

SAS[®] Econometrics 8.5

Econometrics Procedures

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2019. SAS® *Econometrics 8.5: Econometrics Procedures*. Cary, NC: SAS Institute Inc.

SAS® Econometrics 8.5: Econometrics Procedures

Copyright © 2019, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

August 2020

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to <http://support.sas.com/thirdpartylicenses>.

Contents

Chapter 1.	What's New in SAS Econometrics 8.5 Procedures	1
Chapter 2.	Introduction	7
Chapter 3.	Shared Concepts	11
Chapter 4.	The CARIMA Procedure	71
Chapter 5.	The CCDM Procedure	93
Chapter 6.	The CCOPULA Procedure	207
Chapter 7.	The CESM Procedure	249
Chapter 8.	The CNTSELECT Procedure	285
Chapter 9.	The CPANEL Procedure	339
Chapter 10.	The CQLIM Procedure	437
Chapter 11.	The CSPATIALREG Procedure	477
Chapter 12.	The ECM Procedure	549
Chapter 13.	The HMM Procedure	597
Chapter 14.	The SASEMOOD Interface Engine	781
Chapter 15.	The SEVSELECT Procedure	807
Chapter 16.	The TSINFO Procedure	985
Chapter 17.	The TSMODEL Procedure	995
Chapter 18.	Time Series Analysis Package	1039
Chapter 19.	Time Series Model Package	1111
Chapter 20.	Utility Package	1193

Subject Index	1215
----------------------	-------------

Syntax Index	1227
---------------------	-------------

Chapter 1

What's New in SAS Econometrics 8.5 Procedures

Contents

Overview	1
Highlights of Changes and Enhancements	2
New Interface Engines and Procedures	2
SASEMOOD Interface Engine	2
TSINFO Procedure	3
Enhancements to Existing Procedures	3
CCDM Procedure	3
CCOPULA Procedure	3
CPANEL Procedure	4
HMM Procedure	4
SEVSELECT Procedure	4
Highlights of SAS Econometrics 8.4	5

Overview

This chapter summarizes the new features available in SAS Econometrics 8.5 procedures and the highlights of SAS Econometrics 8.4 procedures.

If you have used SAS Econometrics procedures in the past, you can review this chapter to learn about the new features that have been added. When you see a new feature that might be useful for your work, go to the appropriate chapter to read about that feature in detail.

Highlights of Changes and Enhancements

The following interface engine and procedure have been added to SAS Econometrics software:

- SASEMOOD interface engine
- TSINFO procedure

New features have been added to the following SAS Econometrics procedures:

- CCDM procedure
- CCOPULA procedure
- CPANEL procedure
- HMM procedure
- SEVSELECT procedure

New Interface Engines and Procedures

SASEMOOD Interface Engine

The SASEMOOD interface engine enables SAS users to retrieve time series data from the Moody's Analytics Data Buffet (Economy.com) website. This website offers access to more than 600 sources of global historical statistical data at the regional, national, and subnational level, in addition to 40 forecast database offerings. Over 220 million time series are updated promptly after release, on topics such as housing, labor, demographics, finance, industry, housing prices, and consumer credit performance forecasts. Time series are offered in yearly, semiannual, quarterly, monthly, semimonthly, biweekly, weekly, workday (business), daily, and indexed frequencies.

- For more information about the Moody's Analytics Data Buffet, view the catalog of Moody's databases at the following URL: <https://www.economy.com/databuffet/>.
- You can select the time series that you want to retrieve by using the mnemonic code that corresponds to the desired data set. Then you can choose the time range of the retrieved data by specifying a start date and an end date, and you can store the resulting time series data in a SAS data set and in a CAS table.
- To use the SASEMOOD interface engine, you must agree to comply with the terms of use, which are shown at the following URL: <https://www.economy.com/about/terms-of-use>.

TSINFO Procedure

The new TSINFO procedure evaluates a variable in an input data table for its suitability as a time ID variable in SAS procedures and solutions that are used for time series analysis. PROC TSINFO assesses how well a time interval specification fits SAS date values, SAS datetime values, or observation numbers that are used to index a time series. The time interval that this analysis uses can be either specified explicitly as input or inferred from values of the time ID variable. The procedure produces diagnostic information in the form of data tables and ODS tables. These diagnostic results summarize characteristics of the time ID variable that can help determine its use as an index in other time series procedures and solutions.

PROC TSINFO is intended for use as a tool either to identify the time interval of a variable or to prepare problematic data sets for use in subsequent time series analyses. In particular, this procedure can be used to investigate inconsistencies between time ID values and the ID statement options that are used in other SAS procedures and solutions.

Enhancements to Existing Procedures

CCDM Procedure

The following features have been added to the CCDM procedure:

- The output style as it relates to adjusted severity symbols is now consistent whether you specify one or more adjusted severity symbols in the ADJUSTEDSEVERITY= option. In previous releases, the output style with one adjusted severity symbol differed from the output style with more than one adjusted severity symbol. To revert to the older style for the case of one adjusted severity symbol, you can specify the new ONEADJSEVCOMPAT option in the PROC CCDM statement.

CCOPULA Procedure

The following features have been added to the CCOPULA procedure:

- The new PLOTS= option enables you to request correlation plots for pairs of variables in the DATA= data table for the FIT statement. Both the original data and their transformed uniform values can be plotted. The PLOTS= option is also available to plot the correlations between variables that are generated by the SIMULATE statement.
- Additional options are available in the MARGINALS=EMPIRICAL option in the FIT statement. These options support a more accurate algorithm for computing the cumulative distribution function for variables in large data tables and for variables that have nonsmooth distributions and extreme values.
- A new SimulateModelSummary ODS table has been added to describe results of the SIMULATE statement.

CPANEL Procedure

The following features have been added to the CPANEL procedure:

- The heteroscedasticity-corrected standard error estimator (HCCME) is supported.
- The cross-sectional-correlation-corrected (CLUSTER) standard error estimator is supported.
- The heteroscedasticity- and autocorrelation-consistent (HAC) standard error estimator is supported.

HMM Procedure

The following features have been added to the HMM procedure:

- State-independent parameter estimation is supported.
- The analytic store technology is supported.

SEVSELECT Procedure

The following features have been added to the SEVSELECT procedure:

- The parameter estimation phase now balances the available work in each BY group equitably among all available threads of computation. This new default behavior can lead to significantly faster estimation times, but it can also cause the numeric results to differ from those in previous releases. To revert to the numeric results of older releases, you can specify the new NOBALANCETHREADS option in the PROC SEVSELECT statement.

Highlights of SAS Econometrics 8.4

Some users might be unfamiliar with updates made in the previous releases of SAS Econometrics. Following are highlights of enhancements in SAS Econometrics 8.4:

- The CCOPULA procedure adds the BY statement to perform separate copula analyses in a single PROC CCOPULA step. The CCOPULA procedure adds the FIT statement to estimate copula parameters. The CCOPULA procedure adds the STORE= and RESTORE= options and the VIEWSTORE statement to preserve, use, and view the results of a FIT statement between PROC CCOPULA steps.
- The CCDM procedure enables you to prepare various types of plots (PLOTS= option); to specify two new simulation modes, pure premium and custom, in addition to the default collective risk simulation mode (SIMULATIONMODE= option); to specify externally fitted parametric frequency models (COUNTMODEL statement); to specify stochastic symbols to use for computing adjusted severity symbols (SIMULATEDSYMBOL statement); and to generate perturbed samples with different methods (PERTURBMODE= option).
- The CPANEL procedure enables you to fit FIXONETIME, BTWNT, IVFIXONETIME, IVFIXTWO, IVG2SLS, and IVRANTWO models. It also adds the two-step GMM estimation method for the instrumental variables (IV) models. In addition, IV models can use the ROBUST option and the BIASCORRECTED option to obtain different covariance matrices.
- The CQLIM procedure enables you to fit bivariate and multivariate models by specifying multiple MODEL statements, one for each response variable.
- The CSPATIALREG procedure adds the support of impact estimation, which enables you to summarize the average direct impacts, the average indirect impacts, and the average total impacts from changes in an explanatory variable.
- The HMM procedure adds regime-switching autoregression models in the mean-adjusted form, support for the stochastic gradient descent (SGD) optimization algorithm, the solution for the label switching problem, the output of the covariance matrix of parameter estimates, and the output of the standard errors and covariance matrix of forecasts of dependent variables.
- The SEVSELECT procedure enables you to prepare conditional probability density function plots (PLOTS=CONDITIONALPDF and PLOTS=CONDITIONALPDFPERDIST options), to display the parameter covariance and parameter correlation tables (PRINT=COVB and PRINT=CORRB options), and to ignore constant distribution parameters from the computation of likelihood-based fit statistics (NOCONSTFITSTATS option).
- The TSMODEL procedure enables you to specify the method to use for distributing the input time series observations in the input tables across the nodes in the CAS session (DISTRIBUTE= option) and also to select the best international SAS format for the time ID values that are stored in CAS output tables (NLFORMAT= option). In addition, the time series model (TSM) package for the TSMODEL procedure adds the new CFC object, which enables you to create optimized combinations of time series model forecasts.

For more information about these procedures, see the appropriate chapter in *SAS Econometrics: Econometrics Procedures*.

Chapter 2

Introduction

Contents	
Overview of SAS Econometrics Procedures	7
About This Book	8
Chapter Organization	8
Typographical Conventions	9
Options Used in Examples	9
Where to Turn for More Information	10
Online Documentation	10
SAS Technical Support Services	10

Overview of SAS Econometrics Procedures

SAS Econometrics procedures include predictive modeling tools that have been specially developed to take advantage of the distributed environment that SAS Viya provides. Methods include time series, compound distribution modeling, copula fitting and simulation, count data, panel data, limited dependent variables, spatial regression, economic capital modeling, hidden Markov models and severity modeling. In addition, these procedures provide many diagnostic tests, variable and model selection methods, visualization and distribution fitting comparisons that can help you to find the best model for your analysis.

NOTE: When you license SAS Econometrics, you also have access to SAS/ETS software. For more information about SAS/ETS procedures, see *SAS/ETS User's Guide*.

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Language Reference: Concepts* and *Base SAS Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts, such as using the DATA step to create SAS data sets and using Base SAS procedures (such as the PRINT and SORT procedures) to manipulate SAS data sets.

Chapter Organization

This book contains the following chapters:

- **Chapter 2**, this chapter, provides an overview of SAS Econometrics procedures and summarizes related information, products, and services.
- **Chapter 3** provides information about topics that are common to multiple procedures. Topics include how to use SAS Cloud Analytic Services (CAS) sessions and how to load a SAS data set onto a CAS server. This chapter also documents the following statements that are used across a number of procedures: CLASS, DISPLAY, DISPLAYOUT, EFFECT, and SELECTION.
- Next are chapters that describe the procedures; they appear in alphabetical order by procedure name and are organized as follows:
 - The “Overview” section briefly describes the analysis provided by the procedure.
 - The “Getting Started” section provides a quick introduction to the procedure through a simple example.
 - The “Syntax” section describes the SAS statements and options that control the procedure.
 - The “Details” section discusses methodology and other topics, such as ODS tables.
 - The “Examples” section contains examples that use the procedure.
 - The “References” section contains references for the methodology.
- The last chapters in the book describe packages that you can use with the TSMODEL procedure. For more information about using packages, see *SAS Visual Forecasting: Time Series Packages*.

Typographical Conventions

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

<i>roman</i>	is the standard type style used for most text.
UPPERCASE ROMAN	is used for SAS statements, options, and other SAS language elements when they appear in text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
UPPERCASE BOLD	is used in the “Syntax” sections’ initial lists of SAS statements and options.
<i>oblique</i>	is used in the syntax definitions and in text to represent arguments for which you supply a value.
VariableName	is used for the names of variables and data sets when they appear in text.
bold	is used for matrices and vectors.
<i>italic</i>	is used for terms that are defined in text, for emphasis, and for references to publications.
monospace bold	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

The HTMLBLUE style is used to create the graphs and the HTML tables that appear in the online documentation. The PEARLJ style is used to create the PDF tables that appear in the documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. You can specify a style template for an HTML ODS destination as follows:

```
ods html style=HTMLBlue;
```

You can also specify a style template for a PDF ODS destination as follows:

```
ods pdf style=PearlJ;
```

Most of the PDF tables are produced by using the following SAS System option:

```
options papersize=(6.5in 9in);
```

If you run the examples, you might get slightly different output. This is a function of the SAS System options that are used and the precision that your computer uses for floating-point calculations.

Where to Turn for More Information

Online Documentation

You can access the documentation by going to <http://support.sas.com/documentation>.

SAS Technical Support Services

The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of procedures in this book. Go to <http://support.sas.com/techsup> for more information.

Chapter 3

Shared Concepts

Contents

Introduction to Shared Concepts	12
Using CAS Sessions and CAS Engine Librefs	12
Loading a SAS Data Set onto a CAS Server	13
Syntax Common to SAS Econometrics Procedures	14
CLASS Statement	14
DISPLAY Statement	18
DISPLAYOUT Statement	19
EFFECT Statement	20
Collection Effects	22
Multimember Effects	22
Polynomial Effects	24
Spline Effects	27
Splines and Spline Bases	30
ODS Table Names	35
SELECTION Statement	36
Optimization Options	42
Details for SAS Econometrics Procedures	46
Levelization of Classification Variables	46
Specification and Parameterization of Model Effects	49
Effect Operators	50
GLM Parameterization of Classification Variables and Effects	52
Nonsingular Parameterization	56
Class Variable Parameterization with Unbalanced Designs	59
Model Selection Methods	59
Forward Selection	59
Backward Elimination	61
Stepwise Selection	62
Forward-Swap Selection	63
Informative Missingness	63
Multithreading	65
Choosing an Optimization Algorithm	65
First- or Second-Order Algorithms	65
Algorithm Descriptions	66
References	69

Introduction to Shared Concepts

SAS Econometrics procedures run on SAS Viya. One component of SAS Viya is SAS Cloud Analytic Services (CAS), which is the analytic server and associated cloud services. The following subsections describe how to set up and use CAS sessions.

The section “[Syntax Common to SAS Econometrics Procedures](#)” on page 14 describes the common syntax elements that are supported by some of the procedures in this book. In some cases, individual procedures implement these common elements in slightly different ways. When this occurs, the differences are described in the respective procedure chapters.

The section “[Details for SAS Econometrics Procedures](#)” on page 46 provides details that are common to some of the procedures in this book.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see [SAS Cloud Analytic Services: User's Guide](#). For general information about CAS and CAS sessions, see [SAS Cloud Analytic Services: Fundamentals](#).

Loading a SAS Data Set onto a CAS Server

Procedures in this book require the input data to reside on a CAS server. To work with a SAS data set, you must first load the data set onto the CAS server. Data loaded on the CAS server are called *data tables*. This section lists three methods of loading a SAS data set onto a CAS server. In this section, *mycas* is the name of the caslib that is connected to the *mysess* CAS session.

- You can use a single DATA step to create a data table on the CAS server as follows:

```
data mycas.Sample;
  input y x @@;
  datalines;
.46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7
;
```

Note that DATA step operations might not work as intended when you perform them on the CAS server instead of the SAS client.

- You can create a SAS data set first, and when it contains exactly what you want, you can use another DATA step to load it onto the CAS server as follows:

```
data Sample;
  input y x @@;
  datalines;
.46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 .78 8
;
data mycas.Sample;
  set Sample;
run;
```

- You can use the CASUTIL procedure as follows:

```
proc casutil sessref=mysess;
  load data=Sample casout="Sample";
quit;
```

The CASUTIL procedure can load data onto a CAS server more efficiently than the DATA step. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: User's Guide*.

The *mycas* caslib stores the *Sample* data table, which can be distributed across many machine nodes. You must use a caslib reference in procedures in this book to enable the SAS client machine to communicate with the CAS session. For example, the following SEVSELECT procedure statements use a data table that resides in the *mycas* caslib:

```
proc sevselect data = mycas.Sample;
  ...statements...;
run;
```

You can delete your data table by using the DELETE procedure as follows:

```
proc delete data = mycas.Sample;
run;
```

The Sample data table is accessible only in the mysess session. When you terminate the mysess session, the Sample data table is no longer accessible from the CAS server. If you want your Sample data table to be available to other CAS sessions, then you must *promote* your data table. For more information about data tables, see *SAS Cloud Analytic Services: User's Guide*.

Syntax Common to SAS Econometrics Procedures

CLASS Statement

```
CLASS variable <(options)> . . . <variable <(options)> > > </ global-options> ;
```

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. For more information about these mappings, see the section “[Levelization of Classification Variables](#)” on page 46.

If the procedure permits a classification variable as a response (dependent variable or target), the response does not need to be specified in the CLASS statement.

You can specify options either as individual variable *options*, by enclosing the options in parentheses after the variable name, or as *global-options*, by placing them after a slash (/). *Global-options* are applied to all variables that are specified in the CLASS statement. If you specify more than one CLASS statement, the *global-options* that are specified in any one CLASS statement apply to all CLASS statements. However, individual CLASS variable *options* override the *global-options*.

Table 3.1 summarizes the values you can use for either an *option* or a *global-option*. The options are described in detail in the list that follows Table 3.1.

Table 3.1 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DESCENDING**DESC**

reverses the sort order of the classification variable. If both the DESCENDING and **ORDER=** options are specified, the procedure orders the categories according to the **ORDER=** option and then reverse that order.

MISSING

treats missing values (“.”, “.A”, . . . , “.Z” for numeric variables and blanks for character variables) as valid values for the CLASS variable.

If you do not specify the MISSING option, observations that have missing values for CLASS variables are removed from the analysis.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of classification variables. This ordering determines which parameters in the model correspond to each level in the data.

The following table shows how values of the **ORDER=** option are interpreted.

Value of ORDER=	Levels Sorted By
FORMATTED	External formatted values, except for numeric variables that have no explicit format, which are sorted by their unformatted (internal) values. The sort order is machine-dependent. For numeric variables for which you have supplied no explicit format, the levels are ordered by their internal values.
FREQ	Descending frequency count (levels that have more observations come earlier in the order)
INTERNAL	Unformatted value. The sort order is machine-dependent.

For more information about sort order, see the chapter about the SORT procedure in *Base SAS Procedures Guide* and the discussion of BY-group processing in *SAS Language Reference: Concepts*. By default, **ORDER=FORMATTED**.

PARAM=keyword

specifies the parameterization method for the classification variable or variables. You can specify any of the *keywords* shown in the following table; design matrix columns are created from CLASS variables according to the corresponding coding schemes.

Table 3.2 Value of **PARAM=**

Value of PARAM=	Coding
EFFECT	Effect coding. The REF= option in the CLASS statement determines the reference level.
GLM	Less-than-full-rank reference cell coding. This <i>keyword</i> can be used only as a <i>global-option</i> and is applied to all CLASS variables; all other individual variable parameterization specifications are ignored. The REF= option in the CLASS statement indirectly determines the reference level through the order of levels.

Table 3.2 *continued*

Value of PARAM=	Coding
ORDINAL THERMOMETER	Cumulative parameterization for an ordinal CLASS variable
POLYNOMIAL POLY	Polynomial coding. If the classification variable is numeric, then the ORDER= option in the CLASS statement is ignored, and the internal unformatted values are used.
REFERENCE REF	Reference cell coding. The REF= option in the CLASS statement determines the reference level.
ORTHEFFECT	Orthogonalizes PARAM=EFFECT coding. The REF= option in the CLASS statement determines the reference level.
ORTHORDINAL ORTHOTHERM	Orthogonalizes PARAM=ORDINAL coding
ORTHPOLY	Orthogonalizes PARAM=POLYNOMIAL coding. If the classification variable is numeric, then the ORDER= option in the CLASS statement is ignored, and the internal unformatted values are used.
ORTHREF	Orthogonalizes PARAM=REFERENCE coding. The REF= option in the CLASS statement determines the reference level.

All parameterizations are full rank, except for the GLM parameterization. If you specify a full rank parameterization for any CLASS variable, then every CLASS variable without a specified coding is given the EFFECT coding.

By default, PARAM=GLM. For more information about how parameterization of classification variables affects the construction and interpretation of model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49.

REF='level' | *keyword*

REFERENCE='level' | *keyword*

specifies the reference level that is used when you specify a nonsingular parameterization. You can specify the following values:

'level' specifies the *level* of the variable to use as the reference level. Specify the formatted value of the variable if a format is assigned. You can specify this value only for an individual variable *option*.

FIRST designates the first ordered level as reference. You can specify this value either for an individual variable *option* or for a *global-option*.

LAST designates the last ordered level as reference. You can specify this value either for an individual variable *option* or for a *global-option*.

By default, REF=LAST.

SPLIT

specifies that design matrix columns that correspond to any effect that contains a split classification variable can be selected to enter or leave a model independently of the other design columns of that effect. This option applies to procedures that perform model selection.

Suppose that the variable `temp` has three levels ('hot', 'warm', and 'cold'), that the variable `gender` has two levels ('M' and 'F'), and that the variables are used in a PROC SEVSELECT run as follows:

```
proc sevselect data=mycas.data;
  loss y;
  class temp gender / split;
  scalemodel gender gender*temp;
run;
```

The two effects in the SCALEMODEL statement are split into eight independent effects. The effect “gender” is split into two effects that are labeled “gender_M” and “gender_F”. The effect “gender*temp” is split into six effects that are labeled “gender_M*temp_hot”, “gender_F*temp_hot”, “gender_M*temp_warm”, “gender_F*temp_warm”, “gender_M*temp_cold”, and “gender_F*temp_cold”. The previous PROC SEVSELECT step is equivalent to the following:

```
proc sevselect data=mycas.data;
  loss y;
  scalemodel gender_M gender_F
             gender_M*temp_hot gender_F*temp_hot
             gender_M*temp_warm gender_F*temp_warm
             gender_M*temp_cold gender_F*temp_cold;
run;
```

The SPLIT option can be used on individual classification variables. For example, consider the following PROC SEVSELECT step:

```
proc sevselect data=mycas.data;
  loss y;
  class temp(split) gender;
  scalemodel gender gender*temp;
run;
```

In this case, the effect “gender” is not split and the effect “gender*temp” is split into three effects, which are labeled “gender*temp_hot”, “gender*temp_warm”, and “gender*temp_cold”. Furthermore, each of these three split effects now has two parameters that correspond to the two levels of “gender.” The PROC SEVSELECT step is equivalent to the following:

```
proc sevselect data=mycas.data;
  loss y;
  class gender;
  scalemodel gender gender*temp_hot gender*temp_warm gender*temp_cold;
run;
```

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

This section applies to the following procedures: CCDM, CCOPULA, CNTSELECT, CPANEL, CQLIM, CSPATIALREG, ECM, HMM, and SEVSELECT.

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the specific procedure chapters. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

This section applies to the following procedures: CCDM, CCOPULA, CNTSELECT, CPANEL, CQLIM, CSPATIALREG, ECM, HMM, and SEVSELECT.

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The table names that you can specify are listed in the specific procedure chapters.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

EFFECT Statement

EFFECT *effect-name* = *effect-type* (*var-list* < / *effect-options* >) ;

This section applies to the following procedures: CNTSELECT and SEVSELECT.

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are called *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 52. For example, the terms A, B, x, A*x, and A*B in the following statements define fixed effects of the usual type in a generalized linear model:

```
proc sevselect;
  loss y;
  class A B;
  scalemodel A B x A*x;
run;
```

A constructed effect, on the other hand, is assigned through the EFFECT statement. For example, in the following program, the EFFECT statement defines a constructed effect named spl:

```
proc sevselect;
  loss y;
  class A B;
  effect spl = spline(x);
  scalemodel A B A*spl;
run;
```

The columns of spl are formed from the data set variable x as a cubic B-spline basis with three equally spaced interior knots.

Each constructed effect corresponds to a collection of columns that are referred to by the name you supply. You can specify multiple EFFECT statements, and all EFFECT statements must precede the SCALEMODEL statement in the SEVSELECT procedure or the MODEL statement in other procedures.

You must specify the following arguments:

effect-name

names the effect. This name can appear in only one EFFECT statement and cannot be the name of a variable in the input data set.

effect-type

specifies the type of effect. You can specify the following *effect-types*:

COLLECTION

specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference. For more information, see the section “Collection Effects” on page 22.

MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement. For more information, see the section “ Multimember Effects ” on page 22.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables. For more information, see the section “ Polynomial Effects ” on page 24.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables. For more information, see the section “ Spline Effects ” on page 27.

var-list

specifies a list of variables that are used in constructing the effect. You can also specify any of the *effect-options* that are shown in [Table 3.3](#) after a slash following the *var-list*.

Table 3.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 3.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

Collection Effects

EFFECT *effect-name*=**COLLECTION** (*var-list* < / **DETAILS** >) ;

You use a collection effect to define a set of variables that are treated as a single effect that has multiple degrees of freedom. The variables in *var-list* can be continuous or classification variables. Columns in the design matrix that are contributed by a collection effect are the design columns of its constituent variables in the order in which they appear in the definition of the collection effect. If you specify the **DETAILS** option, then a table that shows the constituents of the collection effect is displayed.

Multimember Effects

EFFECT *effect-name*=**MULTIMEMBER** (*var-list* < / *mm-options* >) ;

EFFECT *effect-name*=**MM** (*var-list* < / *mm-options* >) ;

A multimember effect is formed from one or more classification variables in such a way that each observation can be associated with one or more levels of the union of the levels of the classification variables. In other words, a multimember effect is a classification-type effect with possibly more than one nonzero column entry for each observation. Multimember effects are useful, for example, in modeling the following:

- nurses' effects on patient recovery in hospitals
- teachers' effects on student scores
- lineage effects in genetic studies

The levels of a multimember effect consist of the union of formatted values of the variables that define this effect. Each such level contributes one column to the design matrix. For each observation, the value that

corresponds to each level of the multimember effect in the design matrix is the number of times that this level occurs for the observation.

For example, the following data provide teacher information and end-of-year test scores for students after two semesters:

Student	Score	Teacher1	Teacher2
Mary	87	Tobias	Cohen
Tom	89	Rodriguez	Tobias
Fred	82	Cohen	Cohen
Jane	88	Tobias	.
Jack	99	.	.

For example, Mary had different teachers in the two semesters, Fred had the same teacher in both semesters, and Jane received instruction only in the first semester.

You can model the effect of the teachers on student performance by specifying a multimember effect as follows:

```
CLASS teacher1 teacher2;
EFFECT teacher = MM(teacher1 teacher2);
```

The levels of the teacher effect are Cohen, Rodriguez, and Tobias, and the associated design matrix columns are as follows:

Student	Cohen	Rodriguez	Tobias
Mary	1	0	1
Tom	0	1	1
Fred	2	0	0
Jane	0	0	1
Jack	.	.	.

You can specify the following *mm-options* after a slash (/):

DETAILS

displays a table that shows the levels of the multimember effect.

NOEFFECT

specifies that, for observations of the multimember variables whose levels are all missing, the values in the corresponding design matrix columns be set to 0. If, in the preceding example, the teacher effect is defined by the following statement, then the associated design matrix columns' values for Jack are all 0:

```
EFFECT teacher = MM(teacher1 teacher2 / noeffect);
```

This statement enables you to include Jack in the analysis even though there is no effect of teachers on his performance.

A situation where it is important to designate observations as having no effect that can be attributed to a classification variable is the analysis of crossover designs, where lagged treatment levels are used to model the carryover effects of treatments between periods. Because there is no carryover effect for the first period, the treatment lag effect in a crossover design can be modeled by using a multimember effect that consists of a single classification variable and the NOEFFECT option, as in the following statements:

```
CLASS Treatment lagTreatment;
EFFECT Carryover = MM(lagTreatment / noeffect);
```

The lagTreatment variable contains a missing value for the first period. Otherwise, it contains the value of the treatment variable for the preceding period.

STDIZE

specifies that for each observation, the entries in the design matrix that corresponds to the multimember effect be scaled to have a sum of 1.

WEIGHT=*weight-list*

specifies numeric variables used to weigh the contributions of each of the classification effects that define the constructed multimember effect. The number of variables in *weight-list* must match the number of classification variables that define the effect.

Polynomial Effects

```
EFFECT effect-name=POLYNOMIAL (var-list </ polynomial-options>);
```

```
EFFECT effect-name=POLY (var-list </ polynomial-options>);
```

The variables in *var-list* must be numeric. A design matrix column is generated for each term of the specified polynomial. By default, each of these terms is treated as a separate effect for the purpose of model building. For example, the following two PROC sevselect,upper() steps yield the identical analysis:

```
proc sevselect;
  loss y;
  effect MyPoly = polynomial(x1-x3/degree=2);
  scalemodel MyPoly;
run;

proc sevselect;
  loss y;
  scalemodel x1 x2 x3 x1*x1 x1*x2 x1*x3 x2*x2 x2*x3 x3*x3;
run;
```

You can specify the following *polynomial-options* after a slash (/):

DEGREE=*n*

specifies the degree of the polynomial, where *n* must be a positive integer and is typically a small integer, such as 1, 2, or 3. By default, DEGREE=1.

DETAILS

displays a table that shows the details of the specified polynomial, including the number of terms generated. If you also specify the [STANDARDIZE](#) option, then a table that shows the standardization details is also produced.

MDEGREE=*n*

specifies the maximum degree of any variable in a term of the polynomial, where *n* must be a positive integer. The default is the degree of the specified polynomial. For example, the following statement generates the terms x_1 , x_2 , x_1^2 , x_1x_2 , x_2^2 , $x_1^2x_2$, $x_1x_2^2$ and $x_1^2x_2^2$:

```
EFFECT MyPoly=POLYNOMIAL(x1 x2/degree=4 MDEGREE=2);
```

NOSEPARATE

treats the polynomial as a single effect that has multiple degrees of freedom. The *effect-name* that you specify is used as the constructed effect name, and the labels of the terms are used as labels of the corresponding parameters.

STANDARDIZE <(centerscale-opts)> <= standardize-opt >

standardizes the variables that define the polynomial. By default, the standardized variables receive prefix “s_” in the variable names.

You can use the following *centerscale-opts* to specify how the center and scale are estimated:

METHOD=MOMENTS

estimates the center by the variable mean and the scale by the standard deviation. If a weight variable is specified using a WEIGHT statement, the weights are not used in the computation of the mean and standard deviation, and observations that have invalid weights are ignored. Only observations that are used in performing the analysis are used for the standardization.

METHOD=RANGE

estimates the center by the midpoint of the variable range and the scale as half the variable range. Any observation that has a missing value for any regressor used in the model is ignored when the range of variables in a polynomial effect is computed. Observations that have valid regressor values but missing or invalid values of frequency variables, weight variables, or dependent variables are used in computing variable ranges. By default, METHOD=RANGE.

METHOD=WMOMENTS

is the same as METHOD=MOMENTS except that weighted means and weighted standard deviations are used.

PREFIX=NONE | *quoted-string*

specifies the prefix that is appended to standardized variables when forming the term labels. You can specify the following values:

quoted-string specifies the prefix
NONE does not apply a prefix

By default, PREFIX="s_".

Let

n = number of observations used in the analysis
 w = weight variable
 f = frequency variable
 x = variable to be standardized
 $x_{(n)}$ = $\text{Max}_{i=1}^n(x_i)$
 $x_{(1)}$ = $\text{Min}_{i=1}^n(x_i)$
 F = sum of frequencies
 $= \sum_{i=1}^n f_i$
WF = sum of weighted frequencies
 $= \sum_{i=1}^n w_i f_i$

Table 3.4 shows how the center and scale are computed for each of the supported methods.

Table 3.4 Center and Scale Estimates by Method

Method	Center	Scale
RANGE	$(x_{(n)} + x_{(1)})/2$	$(x_{(n)} - x_{(1)})/2$
MOMENTS	$\bar{x} = \sum_{i=1}^n f_i x_i / F$	$\sqrt{\sum_{i=1}^n f_i (x_i - \bar{x})^2 / (F - 1)}$
WMOMENTS	$\bar{x}_w = \sum_{i=1}^n w_i f_i x_i / \text{WF}$	$\sqrt{\sum_{i=1}^n w_i f_i (x_i - \bar{x}_w)^2 / (F - 1)}$

You can control whether the standardization is to center, scale, or both center and scale by specifying one of the following *standardize-opts*:

CENTER

centers but does not scale the variables. For a variable x ,

$$s_x = x - \text{center}$$

CENTERSCALE

centers and scales the variables. This is the default if you do not specify a *standardization-opt*.

For a variable x ,

$$s_x = \frac{x - \text{center}}{\text{scale}}$$

NONE

performs no standardization.

SCALE

scales but does not center the variables. For a variable x ,

$$s_x = \frac{x}{\text{scale}}$$

Spline Effects

This section discusses the construction of spline effects through the EFFECT statement. A spline effect expands variables into spline bases whose form depends on the options that you specify. For more information about regression splines and spline bases, see the section “[Splines and Spline Bases](#)” on page 30. You request a spline effect with the syntax

EFFECT *name*=**SPLINE** (*var-list* < / *spline-options* >) ;

The variables in *var-list* must be numeric. Design matrix columns are generated separately for each of these variables, and the set of columns is collectively referred to with the specified name. By default, the spline basis that is generated for each variable is a cubic B-spline basis that has three equally spaced knots positioned between the minimum and maximum values of that variable. This yields by default seven design matrix columns for each of the variables in the SPLINE effect.

You can specify the following *spline-options* after a slash (/):

BASIS=BSPLINE

uses a B-spline basis for the spline expansion. For splines of degree d that are defined with n knots, this basis consists of $n + d + 1$ columns. In order to completely specify the B-spline basis, d left-side boundary knots and $\max\{d, 1\}$ right-side boundary knots are also required. For information about how to specify the positions of both the internal and boundary knots, see the [KNOTMETHOD=](#), [DATABOUNDARY](#), [KNOTMIN=](#), and [KNOTMAX=](#) suboptions.

BASIS=TPF(*options*)

specifies a truncated power function basis for the spline expansion. For splines of degree d that are defined with n knots for a variable x , this basis consists of an intercept, polynomials $x, x^2, \dots, x^d$, and one truncated power function for each of the n knots. Unlike the B-spline basis, no boundary knots are required. For information about how you can specify the position of the internal knots, see the [KNOTMETHOD=](#) suboption.

You can modify the number of columns when you request BASIS=TPF by specifying the following *options*:

NOINT excludes the intercept column.

NOPOWERS excludes the intercept and polynomial columns.

By default, BASIS=BSPLINE.

DATABOUNDARY

uses the extremes of the data as boundary knots when building a B-spline basis.

DEGREE=*n*

specifies the degree of the spline transformation, where *n* must be a nonnegative integer and is typically a small integer, such as 0, 1, 2, or 3. By default, DEGREE=3.

DETAILS

displays tables that show the knot locations and the knots associated with each spline basis function.

KNOTMAX=*value*

requests that, for each variable in the EFFECT statement, the right-side boundary knots be equally spaced starting at the maximum of the variable and ending at the specified value. This option is ignored for variables whose maximum value is greater than the specified value or if the **DATABOUNDARY** option is also specified.

KNOTMETHOD=*knot-method* < (*knot-options*) >

specifies how to construct the knots for spline effects. You can choose from the following *knot-methods* and affect the knot construction further with the method-specific *knot-options*:

EQUAL < (*n*) >

specifies that *n* equally spaced knots be positioned between the extremes of the data. By default, *n* = 3. For a B-spline basis, any needed boundary knots continue to be equally spaced unless the DATABOUNDARY option has also been specified. By default, KNOTMETHOD=EQUAL(3).

LIST(*number-list*)

specifies the list of internal knots to be used in forming the spline basis columns. For a B-spline basis, the data extremes are used as boundary knots.

LISTWITHBOUNDARY(*number-list*)

specifies the list of all knots that are used in forming the spline basis columns. When you use a truncated power function basis, this list is interpreted as the list of internal knots. When you use a B-spline basis of degree *d*, then the first *d* entries are used as left-side boundary knots and the last max(*d*, 1) entries in the list are used as right-side boundary knots.

MULTISCALE < (*multiscale-options*) >

generates multiple B-spline bases, which correspond to sets that have an increasing number of internal knots. As the number of internal knots increases, the spline basis that is generated can approximate features of the data at finer scales. So generating bases at multiple scales facilitates the modeling of both coarse- and fine-grained features of the data. For scale *i*, the spline basis corresponds to 2^i equally spaced internal knots. By default, the bases for scales 0–7 are generated. For each scale, a separate spline effect is generated. The name of the constructed spline effect at scale *i* is formed by appending *_Si* to the effect name that you specify in the EFFECT statement. If you specify multiple variables in the EFFECT statement, then spline bases are generated separately for each variable at each scale and the name of the corresponding effect is obtained by appending the variable name followed by *_Si* to the name in the EFFECT statement. For example, the following statement generates effects named spl_x1_S0, spl_x1_S1, spl_x1_S2, . . . , spl_x1_S7 and spl_x2_S1, spl_x2_S2, . . . , spl_x2_S7:

```
EFFECT spl = spline(x1 x2 / knotmethod=multiscale);
```

The MULTISCALE option is ignored if you specify the BASIS=TPF *spline-option*.

The following *multiscale-options* control which scales are included:

STARTSCALE=*n* specifies the start scale, where *n* is a positive integer. By default, STARTSCALE=0.

ENDSCALE=*n* specifies the end scale, where *n* is a positive integer. By default, ENDSCALE=7.

PERCENTILES(*n*)

requests that internal knots be placed at *n* equally spaced percentiles of the variable or variables named in the EFFECT statement. For example, the following statement positions internal knots at the deciles of the variable x. For a B-spline basis, the extremes of the data are used as boundary knots:

```
EFFECT spl = spline(x / knotmethod=percentiles(9));
```

RANGEFRACTIONS(*fraction-list*)

places internal knots at each fraction of the ranges of the variables in the EFFECT statement. For example, if variable x1 ranges between 1 and 3 and variable x2 ranges between 0 and 20, then the following EFFECT statement uses internal knots 1.2, 2, and 2.5 for variable x1 and internal knots 2, 10, and 15 for variable x2:

```
EFFECT spl = spline(x1 x2 / knotmethod=rangefractions(.1 .5 .75));
```

For a B-spline basis, the data extremes are used as boundary knots.

KNOTMIN=*value*

requests that, for each variable in the EFFECT statement, the left-side boundary knots be equally spaced starting at the specified value and ending at the minimum of the variable. This option is ignored for variables whose minimum value is less than the specified value or if the DATABOUNDARY option is also specified.

NATURALCUBIC

uses a natural cubic spline basis for the spline expansion. Natural cubic splines, also known as restricted cubic splines, are cubic splines that are constrained to be linear beyond the extreme knots. The natural cubic spline basis that is produced by the EFFECT statement is obtained by starting from the unrestricted truncated power function cubic spline basis that is defined with *n* distinct knots and imposes the linearity constraints beyond the extreme knots. This basis consists of an intercept, the polynomial *x*, and *n* – 2 functions that are all linear beyond the largest knot. The *i*th function, *i* = 1, 2, ..., *n* – 2, is 0 to the left of the *i*th knot, which is called the “break knot.” For more information about this basis, see the section “[Splines and Spline Bases](#)” on page 30. You can use the NOINT and NOPOWERS suboptions of the BASIS=TPF option to suppress the intercept and polynomial *x* when the columns of the natural cubic spline basis are formed. When you specify the NATURALCUBIC option, the options BASIS=BSPLINE, DATABOUNDARY, DEGREE=, and KNOTMETHOD=MULTISCALE are not applicable.

SEPARATE

requests that, when multiple variables are specified in the EFFECT statement, the spline basis for each variable be treated as a separate effect. The names of these separated effects are formed by appending an underscore followed by the name of the variable to the name that you specify in the EFFECT statement. For example, the following statement generates the effect names spl_x1 and spl_x2:

```
EFFECT spl = spline(x1 x2 / separate);
```

In procedures that support variable selection, such as the SEVSELECT procedure, these two effects can enter or leave the model independently during the selection process.

SPLIT

treats each individual column in the design matrix that corresponds to the spline effect as a separate effect that can enter or leave the model independently. Names for these split effects are generated by appending the variable name and an index for each column to the name that you specify in the EFFECT statement. For example, the effects generated for the spline effect in the following statement are spl_x1:1, spl_x1:2, ..., spl_x1:7 and spl_x2:1, spl_x2:2, ..., spl_x2:7:

```
EFFECT spl = spline(x1 x2 / split);
```

Splines and Spline Bases

This section provides details about how the EFFECT statement constructs spline bases. A spline function is a piecewise polynomial function in which the individual polynomials have the same degree and connect smoothly at join points whose abscissa values, called knots, are prespecified. You can use spline functions to fit curves to a wide variety of data.

A spline of degree 0 is a step function with steps located at the knots. A spline of degree 1 is a piecewise linear function where the lines connect at the knots. A spline of degree 2 is a piecewise quadratic curve whose values and slopes coincide at the knots. A spline of degree 3 is a piecewise cubic curve whose values, slopes, and curvature coincide at the knots. Visually, a cubic spline is a smooth curve, and it is the most commonly used spline when a smooth fit is desired. Note that when no knots are used, splines of degree d are simply polynomials of degree d .

More formally, suppose you specify knots $k_1 < k_2 < k_3 < \dots < k_n$. Then a spline of degree $d \geq 0$ is a function $S(x)$ with $d-1$ continuous derivatives such that

$$S(x) = \begin{cases} P_0(x) & x < k_1 \\ P_i(x) & k_i \leq x < k_{i+1}; i = 1, 2, \dots, n-1 \\ P_n(x) & x \geq k_n \end{cases}$$

where each $P_i(x)$ is a polynomial of degree d . The requirement that $S(x)$ has $d-1$ continuous derivatives is satisfied by requiring that the function values and all derivatives up to order $d-1$ of the adjacent polynomials at each knot match.

A counting argument yields the number of parameters that define a spline with n knots. There are $n+1$ polynomials of degree d , producing $(n+1)(d+1)$ coefficients. However, there are d restrictions at each of the n knots, so the number of free parameters is $(n+1)(d+1) - nd = n + d + 1$. In mathematical terminology this says that the dimension of the vector space of splines of degree d on n distinct knots is $n + d + 1$.

+ 1. If you have $n + d + 1$ basis vectors, then you can fit a curve to your data by regressing your dependent variable by using this basis for the corresponding design matrix columns. In this context, such a spline is known as a regression spline. The EFFECT statement provides a simple mechanism for obtaining such a basis.

If you remove the restriction that the knots of a spline must be distinct and allow repeated knots, then you can obtain functions that have less smoothness and even discontinuities at the repeated knot location. For a spline of degree d and a repeated knot that has multiplicity $m \leq d$, the piecewise polynomials that join such a knot are required to have only $d - m$ matching derivatives. Note that this increases the number of free parameters by $m - 1$ but also decreases the number of distinct knots by $m - 1$. Hence the dimension of the vector space of splines of degree d with n knots is still $n + d + 1$, provided that any repeated knot has a multiplicity less than or equal to d .

The EFFECT statement supports the commonly used *truncated power function* basis and *B-spline* basis. With exact arithmetic and by using the complete basis, you obtain the same fit with either of these bases. The following subsections provide details about constructing spline bases for the space of splines of degree d with n knots that satisfies $k_1 \leq k_2 \leq k_3 < \dots \leq k_n$.

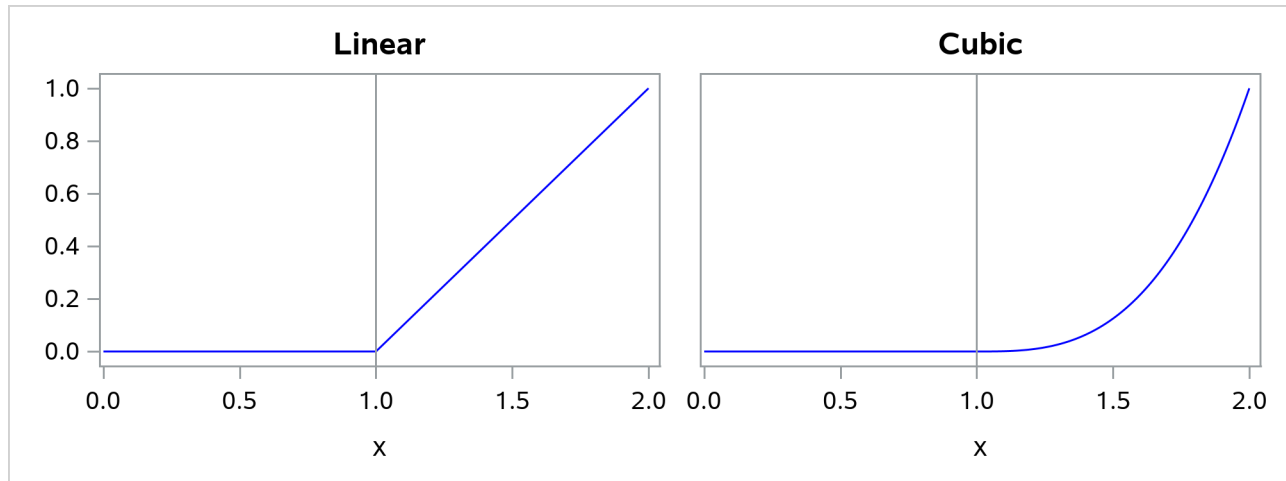
Truncated Power Function Basis

A truncated power function for a knot k_i is a function defined by

$$t_i(x) = \begin{cases} 0 & x < k_i \\ (x - k_i)^d & x \geq k_i \end{cases}$$

Figure 3.1 shows such functions for $d = 1$ and $d = 3$ with a knot at $x = 1$.

Figure 3.1 Truncated Power Functions with Knot at $x = 1$



The name is derived from the fact that these functions are shifted power functions that are truncated to 0 to the left of the knot. These functions are piecewise polynomial functions that have two pieces whose function values and derivatives of all orders up to $d-1$ are 0 at the defining knot. Hence these functions are splines of degree d . It is easy to see that these n functions are linearly independent. However, they do not form a basis, because such a basis requires $n+d-1$ functions. The usual way to add $d+1$ additional basis functions is to use the polynomials $1, x, x^2, \dots, x^d$. These $d+1$ functions together with the n truncated power functions $t_i(x), i = 1, 2, \dots, n$ form the truncated power basis.

Note that each time a knot is repeated, the associated exponent used in the corresponding basis function is reduced by 1. For example, for splines of degree d with three repeated knots $k_i = k_{i+1} = k_{i+2}$, the corresponding basis functions are $t_i(x) = (x - k_i)_+^d$, $t_{i+1}(x) = (x - k_i)_+^{d-1}$, and $t_{i+2}(x) = (x - k_i)_+^{d-2}$. Provided that the multiplicity of each repeated knot is less than or equal to the degree, this construction continues to yield a basis for the associated space of splines.

The main advantage of the truncated power function basis is the simplicity of its construction and the ease of interpreting the parameters in a model that corresponds to these basis functions. However, there are two weaknesses when you use this basis for regression. These functions grow rapidly without bound as x increases, resulting in numerical precision problems when the x data span a wide range. Furthermore, many or even all of these basis functions can be nonzero when evaluated at some x value, resulting in a design matrix that has few zeros and precludes the use of sparse matrix technology to speed up computation. This weakness can be addressed by using a B-spline basis.

B-Spline Basis

A B-spline basis can be built by starting with a set of Haar basis functions, which are functions that are 1 between adjacent knots and 0 elsewhere, and then applying a simple linear recursion relationship d times, yielding the $n + d + 1$ needed basis functions. For the purpose of building the B-spline basis, the n prespecified knots are called internal knots. This construction requires d additional knots, known as boundary knots, to be positioned to the left of the internal knots, and $\max(d, 1)$ boundary knots to be positioned to the right of the internal knots. The actual values of these boundary knots can be arbitrary. The EFFECT statement provides several methods for placing the necessary boundary knots, including the common method of using repeated values of the data extremes as the boundary knots. The boundary knot placement affects the precise form of the basis functions that are generated, but it does not affect the following two desirable properties:

- The B-spline basis functions are nonzero over an interval that spans at most $d + 2$ knots. This yields design matrix columns each of whose rows contain at most $d + 2$ adjacent nonzero entries.
- The computation of the basis functions at any x value is numerically stable and does not require evaluating powers of this value.

The following figures show the B-spline bases that are defined on $[0, 1]$ with four equally spaced internal knots at 0.2, 0.4, 0.6, and 0.8.

Figure 3.2 shows a linear B-spline basis. This basis consists of six functions, each of which is nonzero over an interval that spans at most three knots.

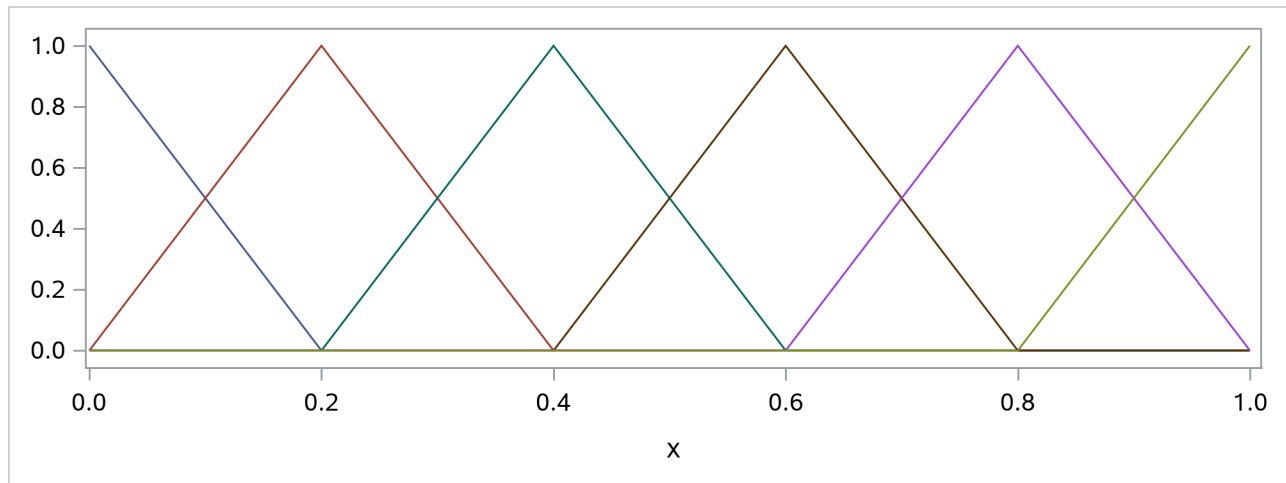
Figure 3.2 Linear B-Spline Basis with Four Equally Spaced Interior Knots

Figure 3.3 shows a cubic B-spline basis where the necessary boundary knots are positioned at $x = 0$ and $x = 1$. This basis consists of eight functions, each of which is nonzero over an interval that spans at most five knots.

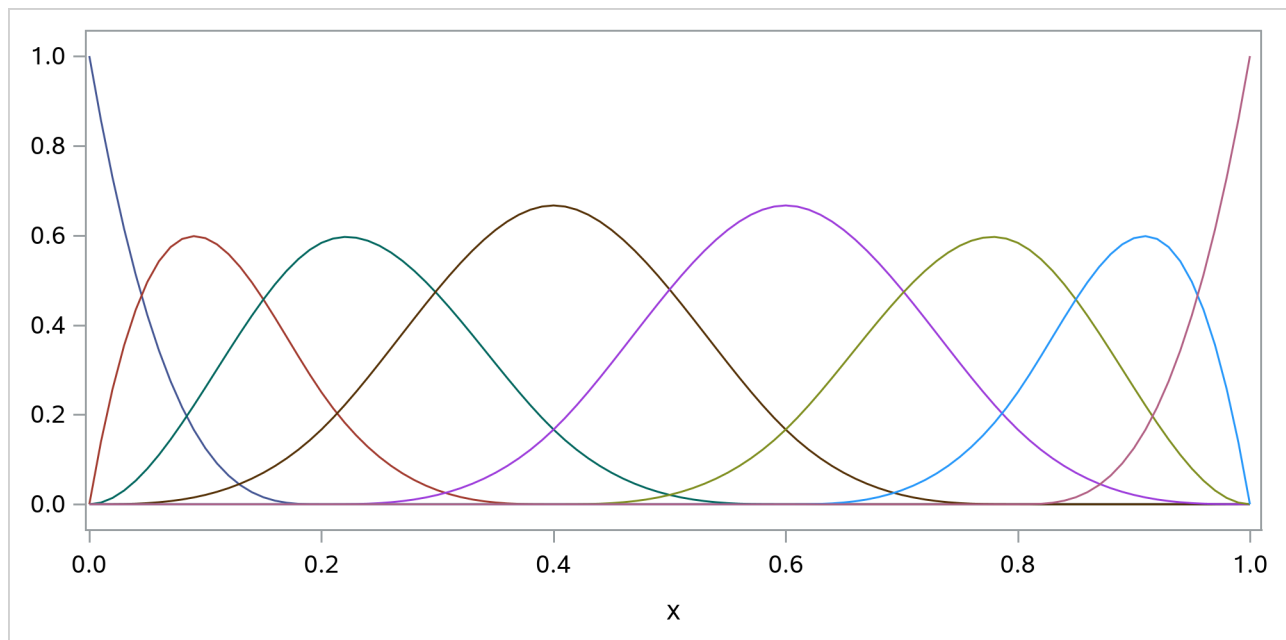
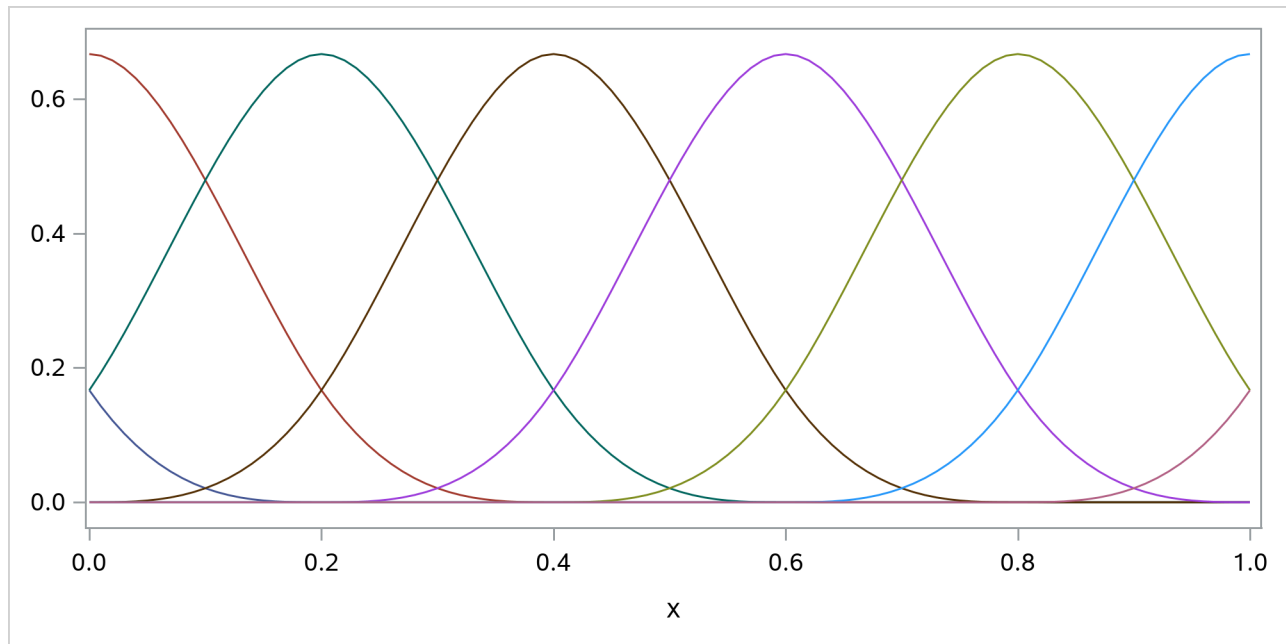
Figure 3.3 Cubic B-Spline Basis with Four Equally Spaced Interior Knots

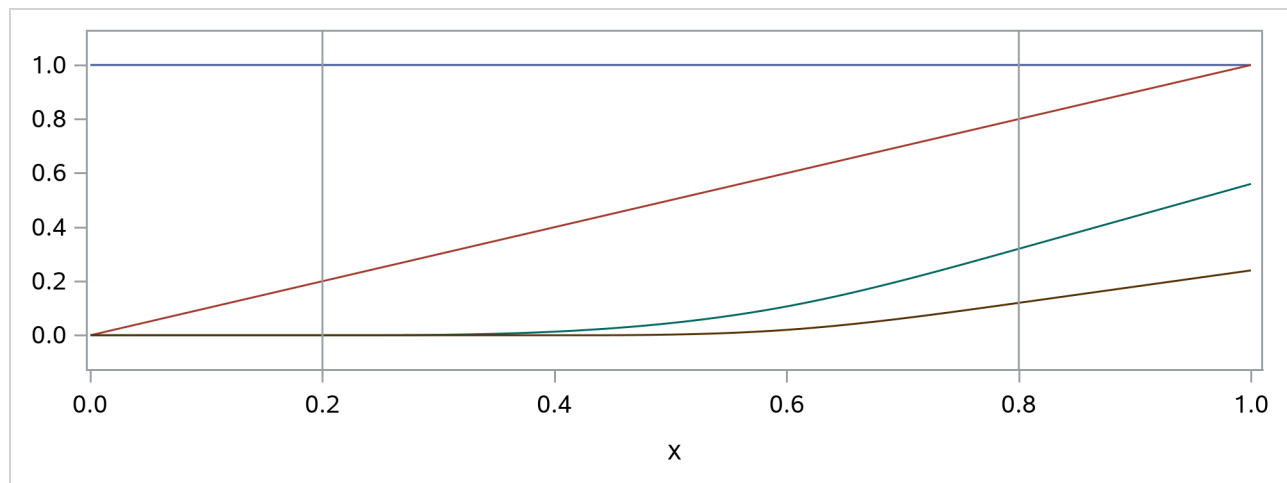
Figure 3.4 shows a different cubic B-spline basis where the necessary left-side boundary knots are positioned at -0.6 , -0.4 , -0.2 , and 0 . The right-side boundary knots are positioned at 1 , 1.2 , 1.4 , and 1.6 . As in the basis shown in Figure 3.3, this basis consists of eight functions, each of which is nonzero over an interval that spans at most five knots. The different positioning of the boundary knots has merely changed the shape of the individual basis functions.

Figure 3.4 Cubic B-Spline Basis with Equally Spaced Boundary and Interior Knots

For more information about this construction, see Hastie, Tibshirani, and Friedman (2001).

Natural Cubic Spline Basis

Natural cubic splines are cubic splines with the additional restriction that the splines are required to be linear beyond the extreme knots. Some authors prefer the terminology “restricted cubic splines” to “natural cubic splines.” The space of unrestricted cubic splines on n knots has the dimension $n + 4$. Imposing the restrictions that the cubic polynomials beyond the first and last knot reduce to linear polynomials reduces the number of degrees of freedom by 4, so a basis for the natural cubic splines consists of n functions. Starting from the truncated power function basis for the unrestricted cubic splines, you can obtain a reduced basis by imposing linearity constraints. For more information about this construction, see Hastie, Tibshirani, and Friedman (2001). [Figure 3.5](#) shows this natural cubic spline basis defined on $[0, 1]$ with four equally spaced internal knots at 0.2, 0.4, 0.6, and 0.8. This basis consists of four basis functions that are all linear beyond the extreme knots at 0.2 and 0.8.

Figure 3.5 Natural Cubic Spline Basis with Four Equally Spaced Knots

ODS Table Names

Each constructed effect produces one or more informative tables when the DETAILS option is also specified. Each table has a name associated with it. This name is used to refer to the table by procedures that support a DISPLAY or DISPLAYOUT statement. You also use this name to refer to the table in ODS statements. These names are listed in [Table 3.5](#).

Table 3.5 ODS Tables Produced by the EFFECT Statement

Table Name	Description	Option
BSplineDetails	B-spline basis details	BASIS=BSPLINE
CollectionLevelInfo	Levels of collection effects	
MMLevelInfo	Levels of multimember effects	
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	
PolyScaling	Centering and scaling details	STANDARDIZE
SplineKnots	knot and boundary knot values	
TPFSplineDetails	Truncated power function spline basis details	BASIS=TPF or NATURALCUBIC

SELECTION Statement

SELECTION < **METHOD**=*method*< (*method-options*) > > < *options* > ;

This section applies to the following procedures: CNTSELECT and SEVSELECT.

Procedures in this book that support model selection use the SELECTION statement to control details about the model selection process.

You can specify the following methods in the SELECTION statement:

METHOD=*method* < (*method-options*) >

specifies the method used to select the model. You can also specify *method-options* that apply to the specified method by enclosing them in parentheses after the *method*.

The following *methods* are available and are explained in detail in the section “[Model Selection Methods](#)” on page 59. By default, METHOD=STEPWISE.

NONE	specifies no model selection.
BACKWARD	specifies backward elimination. This method starts with all effects in the model and deletes effects.
FORWARD	specifies forward selection. This method starts with no effects in the model and adds effects.
STEPWISE	specifies stepwise regression. This method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.
FORWARDSWAP	specifies forward-swap selection, which is an extension of the forward selection method. Before any addition step, the procedure makes all pairwise swaps of one effect in the model and one effect out of the current model that improve the selection criterion.

[Table 3.6](#) lists the applicable *method-options* for each of these methods.

Table 3.6 Applicable *method-options* by *method*

<i>method-option</i>	FORWARD	BACKWARD	STEPWISE
CHOOSE =	x	x	x
COMPETITIVE			x
CRITERION =	x	x	x
MAXEFFECTS =	x		x
MAXSTEPS =	x	x	x
MINEFFECTS =		x	x
SELECT =	x	x	x
STOP =	x	x	x

The syntax of the *method-options* that you can specify in parentheses after the `SELECTION=` option *method* follows. As described in Table 3.6, not all selection *method-options* are applicable to every *method*.

CHOOSE=*criterion*

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. If the optimal value of the specified criterion occurs for models at more than one step, then the model that has the smallest number of parameters is chosen. If you do not specify the `CHOOSE=` option, then the selected model is the model at the final step in the selection process. The criteria that are supported depend on the type of model that is being fit. For the supported values of *criterion*, see the chapters for the relevant procedures.

COMPETITIVE

is applicable as a *method-option* only when `METHOD=STEPWISE`. If you specify the `COMPETITIVE` option, then the `SELECT=` criterion is evaluated for all models in which an effect currently in the model is dropped or an effect not yet in the model is added. The effect whose removal from or addition to the model yields the maximum improvement to the `SELECT=` criterion is dropped or added.

CRITERION=*criterion*

is an alias for the `SELECT=` option.

MAXEFFECTS=*n*

specifies the maximum number of effects in any model that is considered during the selection process. This option is ignored when `METHOD=BACKWARD` is specified. If, at some step of the selection process, the model contains the specified maximum number of effects, then no candidates for addition are considered.

MAXSTEPS=*n*

specifies the maximum number of selection steps that are performed. The default value of *n* is the number of effects in the `MODEL` statement when `METHOD=FORWARD` or `METHOD=BACKWARD`. The default is three times the number of effects when `METHOD=STEPWISE`.

MINEFFECTS=*n*

specifies the minimum number of effects in any model that is considered during backward selection. This option is ignored unless `METHOD=BACKWARD` is specified. The backward selection process terminates if, at some step of the selection process, the model contains the specified minimum number of effects.

SELECT=*criterion*

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. For each step, the effect whose addition to or removal from the current model yields the maximum improvement in the specified criterion is selected. For supported criteria, see the chapter for the relevant procedure.

STOP=NONE | *criterion*

specifies a criterion that is used to stop the selection process. The criteria that are supported depend on the type of model that is being fit. For information about the supported criteria, see the chapter for the relevant procedure.

If you do not specify the STOP= option but do specify the **SELECT=** option, then the criterion specified in the **SELECT=** option is also used as the STOP= criterion.

You can specify the following values:

NONE	stops the selection process if no suitable add or drop candidates can be found or if a size-based limit is reached. For example, if you specify STOP=NONE MAXEFFECTS=5, then the selection process stops at the first step that produces a model that has five effects.
<i>criterion</i>	stops the selection process if the selection process produces a local extremum of this <i>criterion</i> or if a size-based limit is reached. For example, if you specify STOP=AIC MAXSTEPS=5, then the selection process stops before step 5 if the sequence of models has a local minimum of the AIC criterion before step 5. The determination of whether a local minimum is reached is made on the basis of a stop horizon. The default stop horizon is 3, but you can change it by using the STOPHORIZON= option. If the stop horizon is n and the STOP= criterion at any step is better than the stop criterion at the next n steps, then the selection process terminates.

In addition, you can also specify the following *options*:

DETAILS=NONE | **SUMMARY** | **ALL****DETAILS=STEPS**<(CANDIDATES(ALL | n))>

specifies the level of detail to be produced about the selection process. The default is DETAILS=SUMMARY.

The DETAILS=ALL and DETAILS=STEPS options produce the following output:

- tables that provide information about the model that is selected at each step of the selection process.
- entry and removal statistics for inclusion or exclusion candidates at each step. By default, only the top 10 candidates at each step are shown. If you specify STEPS(CANDIDATES(n)), then the best n candidates are shown. If you specify STEPS(CANDIDATES(ALL)), then all candidates are shown.
- a selection summary table that shows by step the effect that is added to or removed from the model in addition to the values of the SELECT, STOP, and CHOOSE criteria for the resulting model.
- a stop reason table that describes why the selection process stopped.
- a selection reason table that describes why the selected model was chosen.
- a selected effects table that lists the effects that are in the selected model.

The DETAILS=SUMMARY option produces only the selection summary, stop reason, selection reason, and selected effects tables. In addition, for METHOD=ELASTICNET without specified L2 value, the DETAILS=SUMMARY option also displays an elastic net summary table, which shows the ridge regularization parameter L2 (in the ratio scaling) and CHOOSE criterion values in each L2 search step.

HIERARCHY=NONE | SINGLE | SINGLECLASS

specifies whether and how the model hierarchy requirement is applied. You can specify that only classification effects, or both classification and continuous effects, be subject to the hierarchy requirement. This option is ignored unless you also specify one of the following options: **METHOD=FORWARD**, **METHOD=BACKWARD**, or **METHOD=STEPWISE**.

Model hierarchy refers to the requirement that, for any term to be in the model, all model effects that are contained in the term must be present in the model. For example, in order for the interaction A*B to enter the model, the main effects A and B must be in the model. Likewise, neither effect A nor effect B can leave the model while the interaction A*B is in the model.

You can specify the following values:

NONE	specifies that model hierarchy not be maintained. Any single effect can enter or leave the model at any step of the selection process.
SINGLE	specifies that only one effect enter or leave the model at one time, subject to the model hierarchy requirement. For example, suppose that the model contains the main effects A and B and the interaction A*B. In the first step of the selection process, either A or B can enter the model. In the second step, the other main effect can enter the model. The interaction effect can enter the model only when both main effects have already entered. Also, before A or B can be removed from the model, the A*B interaction must first be removed. All effects (CLASS and interval) are subject to the hierarchy requirement.
SINGLECLASS	is the same as HIERARCHY=SINGLE except that only CLASS effects are subject to the hierarchy requirement.

By default, **HIERARCHY=NONE**.

ORDERSELECT

specifies that effects in the selected model are displayed in the order in which they first entered the model. If you do not specify the **ORDERSELECT** option, then effects in the selected model are displayed in the order in which they appear in the **SCALEMODEL** or **MODEL** statement.

PLOT <(global-plot-options)> <= plot-request <(options)>>

PLOTS <(global-plot-options)> <= (plot-request <(options)> <...plot-request <(options)>>>

controls the selection process plots that are produced through ODS Graphics. The only procedure in this book that supports this option is the **SEVSELECT** procedure.

When you specify only one plot request, you can omit the parentheses around it. Here are some examples:

```
plots=all
plots=coefficients(unpack)
plots(unpack)=(coefficients criteria)
```

ODS Graphics must be enabled before you can request plots. For example:

```
ods graphics on;

proc sevselect;
  model y = x1-x100;
  selection method=forward plots=all;
run;

ods graphics off;
```

Global Plot Options

Each *global-plot-option* applies to all plots that the **SELECTION** statement generates, unless you alter the option by using a specific plot option that follows a *plot-request*. You can specify the following *global-plot-options*:

ENDSTEP=*n*

terminates the step ranges shown on the horizontal axes of plots at the specified step. By default, the step range that is shown terminates at the final step of the selection process.

MAXPARMLABEL=*n*

specifies the maximum number of characters beyond which all parameter labels on the coefficient progression panel and the coefficient progression plot are suppressed. By default, MAXPARMLABEL=20.

MAXSTEPLABEL=*n*

specifies the maximum number of characters of effect labels to display on the horizontal axes of the plots. By default, MAXSTEPLABEL=20. If the length of any effect label exceeds *n*, then step numbers are used on the horizontal axes of the plots.

STARTSTEP=*n*

specifies that the step ranges shown on the horizontal axes of plots start at the specified step. By default, the step range that is shown starts at the initial step of the selection process.

STEPAXIS=EFFECT | NUMBER

specifies the horizontal axis to be used on the plots, where this axis represents the sequence of entering or departing effects. You can specify the following values:

- | | |
|---------------|---|
| EFFECT | requests that each step be labeled by a prefix followed by the name of the effect that enters or leaves at that step. The prefix consists of the step number, followed by a “+” or a “-” sign, depending on whether the effect enters (+) or leaves (-) at that step. |
| NUMBER | requests that each step be labeled by the step number. |

UNPACK | UNPACKPANEL

suppresses paneling. By default, multiple plots can appear in the coefficient progression panel and the criteria panel. Specify UNPACK to see each plot individually. You can also specify UNPACK as a suboption of the CRITERIA and COEFFICIENTS options.

Specific Plot Options

You can specify the following *plot-requests* and their *options*:

ALL

produces all relevant plots.

COEFFICIENTS | COEFFICIENTPANEL <(UNPACK | UNPACKPANEL)>

creates a panel of two plots. The upper plot shows the progression of the parameter values as the selection process proceeds. The lower plot shows the progression of the CHOOSE= criterion. If no CHOOSE= criterion is in effect, then only the coefficient progression is shown. If you specify the UNPACK option, then the coefficient progression and the CHOOSE= criterion progression are shown in separate plots.

CRITERIA | CRITERIONPANEL <(UNPACK | UNPACKPANEL)>

creates a panel of model fit criteria. If you specify the UNPACK option, then each criterion progression is shown in a separate plot.

NONE

suppress all selection process plots, even if you specify other plot options.

SELECTION=method <(method-options)>

is an alias for the METHOD= option.

STOPHORIZON=n

specifies the number of consecutive steps at which the STOP= criterion must worsen in order for a local extremum to be detected. For example, suppose that STOP=AIC and the sequence of AIC values at steps 1 to 6 of a selection are 10, 7, 4, 6, 5, 2. If STOPHORIZON=2, then the AIC criterion is deemed to have a local minimum at step 3 because the AIC value at the next two steps are greater than the value 4 that occurs at step 3. However, if STOPHORIZON=3, then the value at step 3 is not deemed to be a local minimum because the AIC value at step 6 is lower than the AIC value at step 3. If you specify STOP=NONE then the stop horizon value is ignored. If you specify STOP=SL, then *n* is ignored and STOPHORIZON=1 is used. By default, STOPHORIZON=3 unless otherwise stated in individual procedure chapters.

Optimization Options

This section applies to the following procedures: SEVSELECT.

This section describes options that are typically available in the PROC statement or the NLOPTIONS statement of the procedures in this book that perform optimizations. The following notation is used to describe the options. $\boldsymbol{\beta}$ denotes the $p \times 1$ vector of parameters for the optimization and β_i is its i th element. The objective function being minimized, its $p \times 1$ gradient vector, and its $p \times p$ Hessian matrix are denoted as $f(\boldsymbol{\beta})$, $\mathbf{g}(\boldsymbol{\beta})$, and $\mathbf{H}(\boldsymbol{\beta})$, respectively. The gradient with respect to the i th parameter is denoted as $g_i(\boldsymbol{\beta})$. Superscripts in parentheses denote the iteration count; for example, $f(\boldsymbol{\beta})^{(k)}$ is the value of the objective function at iteration k .

ABSCONV= r

ABSTOL= r

specifies an absolute function convergence criterion. For minimization, termination requires $f(\boldsymbol{\beta}^{(k)}) \leq r$, where $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $f(\cdot)$ is the objective function. The default value of r is the negative square root of the largest double-precision value, which serves only as a protection against overflows.

ABSFCONV= r

ABSFTOL= r

specifies an absolute function difference convergence criterion. For all techniques except NMSIMP, termination requires a small change of the function value in successive iterations:

$$|f(\boldsymbol{\beta}^{(k-1)}) - f(\boldsymbol{\beta}^{(k)})| \leq r$$

Here, $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\boldsymbol{\beta}^{(k)}$ is defined as the vertex that has the lowest function value and $\boldsymbol{\beta}^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex. By default, ABSFCONV=0.

ABSGCONV= r

ABSGTOL= r

specifies an absolute gradient convergence criterion. Termination requires the maximum absolute gradient element to be small:

$$\max_j |g_j(\boldsymbol{\beta}^{(k)})| \leq r$$

Here, $\boldsymbol{\beta}$ is the vector of parameters in the optimization and $g_j(\cdot)$ is the gradient of the objective function with respect to the j th parameter. This criterion is not used by the NMSIMP technique. By default, ABSGCONV=1E-5.

ABSXCONV= r **ABSXTOL= r**

specifies an absolute parameter convergence criterion: For all techniques except NMSIMP, termination requires a small Euclidean distance between successive parameter vectors,

$$\| \boldsymbol{\beta}^{(k)} - \boldsymbol{\beta}^{(k-1)} \|_2 \leq r$$

For the NMSIMP technique, termination requires either a small length $\alpha^{(k)}$ of the vertices of a restart simplex,

$$\alpha^{(k)} \leq r$$

or a small simplex size,

$$\delta^{(k)} \leq r$$

where the simplex size $\delta^{(k)}$ is defined as the L1 distance from the simplex vertex $\boldsymbol{\xi}^{(k)}$ that has the smallest function value to the other p simplex points $\boldsymbol{\beta}_l^{(k)} \neq \boldsymbol{\xi}^{(k)}$:

$$\delta^{(k)} = \sum_{\boldsymbol{\beta}_l^{(k)} \neq \boldsymbol{\xi}^{(k)}} \| \boldsymbol{\beta}_l^{(k)} - \boldsymbol{\xi}^{(k)} \|_1$$

The default is $r = 1\text{E-}8$ for the NMSIMP technique and $r = 0$ otherwise.

FCONV= r **FTOL= r**

specifies a relative function difference convergence criterion. For all techniques except NMSIMP, termination requires a small relative change of the function value in successive iterations,

$$\frac{|f(\boldsymbol{\beta}^{(k)}) - f(\boldsymbol{\beta}^{(k-1)})|}{\max(|f(\boldsymbol{\beta}^{(k-1)})|, \text{FSIZE})} \leq r$$

where FSIZE is defined by the **FSIZE=** option. Here, $\boldsymbol{\beta}$ denotes the vector of parameters that participate in the optimization, and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\boldsymbol{\beta}^{(k)}$ is defined as the vertex that has the lowest function value and $\boldsymbol{\beta}^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex.

The default value is $r=2 \times \epsilon$ where ϵ is the machine precision, which is the smallest double-precision floating-point number such that $1 + \epsilon > 1$.

FCONV2= r **FTOL2= r**

specifies a second function convergence criterion. For all techniques except NMSIMP, termination requires a small predicted reduction of the objective function:

$$df^{(k)} \approx f(\boldsymbol{\beta}^{(k)}) - f(\boldsymbol{\beta}^{(k)} + \mathbf{s}^{(k)})$$

The predicted reduction

$$\begin{aligned} df^{(k)} &= -\mathbf{g}^{(k)'} \mathbf{s}^{(k)} - \frac{1}{2} \mathbf{s}^{(k)'} \mathbf{H}^{(k)} \mathbf{s}^{(k)} \\ &= -\frac{1}{2} \mathbf{s}^{(k)'} \mathbf{g}^{(k)} \leq r \end{aligned}$$

is computed by approximating the objective function f by the first two terms of the Taylor series and substituting the Newton step,

$$\mathbf{s}^{(k)} = -[\mathbf{H}^{(k)}]^{-1} \mathbf{g}^{(k)}$$

For the NMSIMP technique, termination requires a small standard deviation of the function values of the $p + 1$ simplex vertices $\boldsymbol{\beta}_l^{(k)}$, $l = 0, \dots, p$,

$$\sqrt{\frac{1}{n+1} \sum_l \left[f(\boldsymbol{\beta}_l^{(k)}) - \bar{f}(\boldsymbol{\beta}^{(k)}) \right]^2} \leq r$$

where $\bar{f}(\boldsymbol{\beta}^{(k)}) = \frac{1}{p+1} \sum_l f(\boldsymbol{\beta}_l^{(k)})$. If there are p_{act} boundary constraints active at $\boldsymbol{\beta}^{(k)}$, the mean and standard deviation are computed only for the $n + 1 - p_{act}$ unconstrained vertices.

The default value is $r = 1\text{E-}6$ for the NMSIMP technique and $r = 0$ otherwise.

FSIZE= r

specifies the FSIZE parameter of the relative function and relative gradient termination criteria. The default value is $r = 0$. For more information, see the FCONV= and GCONV= options.

GCONV= r

GTOL= r

specifies a relative gradient convergence criterion. For all techniques except CONGRA and NMSIMP, termination requires that the normalized predicted function reduction be small:

$$\frac{\mathbf{g}(\boldsymbol{\beta}^{(k)})' [\mathbf{H}^{(k)}]^{-1} \mathbf{g}(\boldsymbol{\beta}^{(k)})}{\max(|f(\boldsymbol{\beta}^{(k)})|, \text{FSIZE})} \leq r$$

where FSIZE is defined by the FSIZE= option. Here, $\boldsymbol{\beta}$ denotes the vector of parameters that participate in the optimization, $f(\cdot)$ is the objective function, and $\mathbf{g}(\cdot)$ is the gradient. For the CONGRA technique (where a reliable Hessian estimate $\mathbf{H}$ is not available), the following criterion is used:

$$\frac{\|\mathbf{g}(\boldsymbol{\beta}^{(k)})\|_2^2 \|\mathbf{s}(\boldsymbol{\beta}^{(k)})\|_2}{\|\mathbf{g}(\boldsymbol{\beta}^{(k)}) - \mathbf{g}(\boldsymbol{\beta}^{(k-1)})\|_2 \max(|f(\boldsymbol{\beta}^{(k)})|, \text{FSIZE})} \leq r$$

This criterion is not used by the NMSIMP technique. By default, GCONV=1E-8.

GCONV2= r

GTOL2= r

specifies another relative gradient convergence criterion. For the TRUREG, NRRIDG, and NEWRAP techniques, the following criterion of Browne (1982) is used:

$$\max_j \frac{|\mathbf{g}_j(\boldsymbol{\beta}^{(k)})|}{\sqrt{f(\boldsymbol{\beta}^{(k)}) \mathbf{H}_{j,j}^{(k)}}} \leq r$$

This criterion is not used by the other techniques.

By default, GCONV2=0.

MAXFUNC= n **MAXFU= n**

specifies the maximum number n of function calls in the optimization process. The default values are as follows, depending on the optimization technique:

- TRUREG, NRRIDG, and NEWRAP: 125
- QUANEW and DBLDOG: 500
- CONGRA: 1,000
- NMSIMP: 3,000

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number that is specified by this option. You can specify the optimization technique in the **TECHNIQUE=** option.

MAXITER= n **MAXIT= n**

specifies the maximum number n of iterations in the optimization process. The default values are as follows, depending on the optimization technique:

- TRUREG, NRRIDG, and NEWRAP: 50
- QUANEW and DBLDOG: 200
- CONGRA: 400
- NMSIMP: 1,000

These default values also apply when n is specified as a missing value. You can specify the optimization technique in the **TECHNIQUE=** option.

MAXTIME= r

specifies an upper limit of r seconds of CPU time for the optimization process. The time specified by r is checked only once at the end of each iteration. Therefore, the actual running time can be longer than r . The default value is the largest floating-point double representation of your computer.

MINITER= n **MINIT= n**

specifies the minimum number of iterations. If you request more iterations than are actually needed for convergence to a stationary point, the optimization algorithms can behave strangely. For example, the effect of rounding errors can prevent the algorithm from continuing for the required number of iterations. By default, MINITER=0.

TECHNIQUE=*technique***TECH=*technique***

specifies the optimization technique for obtaining maximum likelihood estimates. You can specify one of the following *techniques*:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization with line search.

NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	performs no optimization.
NRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust-region optimization

The default method varies by the procedure. For the SEVSELECT procedure, the default is TECHNIQUE=TRUREG.

For more information, see the section “[Choosing an Optimization Algorithm](#)” on page 65.

XCONV=*r*

XTOL=*r*

specifies the relative parameter convergence criterion. Convergence requires a small relative parameter change in subsequent iterations,

$$\frac{\max_j |\beta_j^{(k)} - \beta_j^{(k-1)}|}{\max(|\beta_j^{(k)}|, |\beta_j^{(k-1)}|, \text{XSIZE})} \leq r$$

where XSIZE is defined by the **XSIZE=** option. and $\beta_j^{(i)}$ is the estimate of the *j*th parameter at iteration *i*. For the NMSIMP technique, the same formula is used, but $\beta_j^{(k)}$ is defined as the vertex that has the lowest function value and $\beta_j^{(k-1)}$ is defined as the vertex that has the highest function value in the simplex. The default value is $r = 1\text{E-}8$ for the NMSIMP technique and $r = 0$ otherwise.

XSIZE=*r*

specifies the XSIZE parameter of the relative parameter termination criterion. The value of *r* must be greater than or equal to 0; the default is $r = 0$. For more information, see the XCONV= option.

Details for SAS Econometrics Procedures

Levelization of Classification Variables

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

A classification variable enters the statistical analysis or model not through its values but through its levels. The process of associating values of a variable with levels is called *levelization*.

During the process of levelization, observations that share the same value are assigned to the same level. The manner in which values are grouped can be affected by the inclusion of formats. The sort order of the levels can be determined by specifying the ORDER= option in the procedure statement. In procedures in this book, you can also control the sorting order separately for each variable in the CLASS statement.

Consider the data on nine observations in [Table 3.7](#). The variable A is integer-valued, and the variable X is a continuous variable that has a missing value for the fourth observation. The fourth and fifth columns of [Table 3.7](#) apply two different formats to the variable X.

Table 3.7 Example Data for Levelization

Obs	A	x	FORMAT x 3.0	FORMAT x 3.1
1	2	1.09	1	1.1
2	2	1.13	1	1.1
3	2	1.27	1	1.3
4	3	.	.	.
5	3	2.26	2	2.3
6	3	2.48	2	2.5
7	4	3.34	3	3.3
8	4	3.34	3	3.3
9	4	3.14	3	3.1

By default, levelization of the variables groups the observations by the formatted value of the variable, except for numerical variables for which no explicit format is provided. Numerical variables for which no explicit format is provided are sorted by their internal value. The levelization of the four columns in [Table 3.7](#) leads to the level assignment in [Table 3.8](#).

Table 3.8 Values and Levels

Obs	A		X		FORMAT x 3.0		FORMAT x 3.1	
	Value	Level	Value	Level	Value	Level	Value	Level
1	2	1	1.09	1	1	1	1.1	1
2	2	1	1.13	2	1	1	1.1	1
3	2	1	1.27	3	1	1	1.3	2
4	3	2	.	.	.	.	.	.
5	3	2	2.26	4	2	2	2.3	3
6	3	2	2.48	5	2	2	2.5	4
7	4	3	3.34	7	3	3	3.3	6
8	4	3	3.34	7	3	3	3.3	6
9	4	3	3.14	6	3	3	3.1	5

The sort order for the levels of CLASS variables can be specified in the **ORDER=** option in the **CLASS** statement.

When **ORDER=FORMATTED** (which is the default) is in effect for numeric variables for which you have supplied no explicit format, the levels are ordered by their internal values. To order numeric classification levels that have no explicit format by their BEST12. formatted values, you can specify the BEST12. format explicitly for the CLASS variables.

[Table 3.9](#) shows how values of the **ORDER=** option are interpreted.

Table 3.9 Interpretation of Values of ORDER= Option

Value of ORDER=	Levels Sorted By
FORMATTED	External formatted value, except for numeric variables that have no explicit format, which are sorted by their unformatted (internal) value. The sort order is machine-dependent.
FREQ	Descending frequency count (levels that have the most observations come first in the order)
INTERNAL	Unformatted value. The sort order is machine-dependent.

For more information about sort order, see the chapter about the SORT procedure in the *Base SAS Procedures Guide* and the discussion of BY-group processing in *SAS Language Reference: Concepts*.

When the MISSING option is specified in the CLASS statement, the missing values (‘.’ for a numeric variable and blanks for a character variable) are included in the levelization and are assigned a level. Table 3.10 displays the results of levelizing the values in Table 3.7 when the MISSING option is in effect.

Table 3.10 Values and Levels When the MISSING Option Is Specified

Obs	A		X		FORMAT x 3.0		FORMAT x 3.1	
	Value	Level	Value	Level	Value	Level	Value	Level
1	2	1	1.09	2	1	2	1.1	2
2	2	1	1.13	3	1	2	1.1	2
3	2	1	1.27	4	1	2	1.3	3
4	3	2	.	1	.	1	.	1
5	3	2	2.26	5	2	3	2.3	4
6	3	2	2.48	6	2	3	2.5	5
7	4	3	3.34	8	3	4	3.3	7
8	4	3	3.34	8	3	4	3.3	7
9	4	3	3.14	7	3	4	3.1	6

When the MISSING option is not specified, it is important to understand the implications of missing values for your statistical analysis. When a procedure in this book levelizes the CLASS variables, an observation for which any CLASS variable has a missing value is excluded from the analysis.

Procedures in this book print a “Number of Observations” table that shows the number of observations that are read from the data set and the number of observations that are used in the analysis. Pay careful attention to this table—especially when your data table contains missing values—to ensure that no observations are unintentionally excluded from the analysis.

Specification and Parameterization of Model Effects

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, HMM, and SEVSELECT.

Procedures in this book that have a MODEL or SCALEMODEL statement support the formation of effects. An *effect* is an element in a linear model structure that is formed from one or more variables. At some point the statistical representations of these models involve linear structures such as

$$\mathbf{X}\boldsymbol{\beta}$$

or

$$\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\boldsymbol{\gamma}$$

The model matrices $\mathbf{X}$ and $\mathbf{Z}$ are formed according to effect-construction rules.

Procedures that also have a CLASS statement support the rich set of effects that is discussed in this section.

Procedures that also have an EFFECT statement enable you to construct special *constructed* effects that are discussed in “EFFECT Statement” on page 20.

In order to correctly interpret the results from a statistical analysis, you need to understand how construction (*parameterization*) rules apply to regression-type models, whether these are scale regression models as in the SEVSELECT procedure or count regression models as in the CNTSELECT procedures.

Effects are specified by a special notation that uses variable names and operators. There are two types of variables: classification (or CLASS) variables and continuous variables. *Classification variables* can be either numeric or character and are specified in a CLASS statement. For more information, see the section “Levelization of Classification Variables” on page 46. An independent variable that is not declared in the CLASS statement is assumed to be *continuous*. For example, the heights and weights of subjects are continuous variables.

Two primary operators (crossing and nesting) are used for combining the variables, and several additional operators are used to simplify effect specification. Operators are discussed in the section “Effect Operators” on page 50.

Procedures in this book that have a CLASS statement support a general linear model (GLM) parameterization and might also support nonsingular parameterizations for the classification variables. The GLM parameterization, commonly called *dummy* parameterization, is the default for all procedures in this book. For more information, see the sections “GLM Parameterization of Classification Variables and Effects” on page 52 and “Nonsingular Parameterization” on page 56.

NOTE: This section uses the MODEL statement in all examples for illustration purposes. All discussion also applies to the SCALEMODEL statement in the SEVSELECT procedure (where you specify only the regression effects) and to other types of model specification statements in other procedures (such as the ZEROMODEL statement in the CNTSELECT procedure).

Effect Operators

Table 3.11 summarizes the operators that are available for selecting and constructing effects. These operators are discussed in the following sections.

Table 3.11 Available Effect Operators

Operator	Example	Description
Interaction	A*B	Crosses the levels of the effects
Nesting	A(B)	Nests A levels within B levels
Bar operator	A B C	Specifies all interactions
At sign operator	A B C@2	Reduces interactions in bar effects
Dash operator	A1-A10	Specifies sequentially numbered variables
Colon operator	A:	Specifies variables with common prefix
Double dash operator	A- -C	Specifies sequential variables in data set order

Bar and At Sign Operators

You can shorten the specification of a large factorial model by using the bar operator. For example, two ways of writing the model for a full three-way factorial model follow:

```
model Y = A B C A*B A*C B*C A*B*C;
```

```
model Y = A|B|C;
```

When the bar (|) is used, the right and left sides become effects, and the cross of them becomes an effect. Multiple bars are permitted. The expressions are expanded from left to right, using rules 2–4 in Searle (1971, p. 390).

- Multiple bars are evaluated from left to right. For example, A | B | C is evaluated as follows:

$$\begin{aligned}
 A | B | C &\rightarrow \{ A | B \} | C \\
 &\rightarrow \{ A \ B \ A*B \} | C \\
 &\rightarrow A \ B \ A*B \ C \ A*C \ B*C \ A*B*C
 \end{aligned}$$

- Crossed and nested groups of variables are combined. For example, A(B) | C(D) generates A*C(B D), among other terms.
- Duplicate variables are removed. For example, A(C) | B(C) generates A*B(C C), among other terms, and the extra C is removed.
- Effects are discarded if a variable occurs on both the crossed and nested parts of an effect. For example, A(B) | B(D E) generates A*B(B D E), but this effect is eliminated immediately.

You can also specify the maximum number of variables involved in any effect that results from bar evaluation by specifying that maximum number, preceded by an at sign (@), at the end of the bar effect. For example, the following specification selects only those effects that contain two or fewer variables:

```
model Y = A|B|C@2;
```

The preceding example is equivalent to specifying the following MODEL statement:

```
model Y = A B C A*B A*C B*C;
```

More examples of using the bar and at operators follow:

A C(B)	is equivalent to	A C(B) A*C(B)
A(B) C(B)	is equivalent to	A(B) C(B) A*C(B)
A(B) B(D E)	is equivalent to	A(B) B(D E)
A B(A) C	is equivalent to	A B(A) C A*C B*C(A)
A B(A) C@2	is equivalent to	A B(A) C A*C
A B C D@2	is equivalent to	A B A*B C A*C B*C D A*D B*D C*D
A*B(C*D)	is equivalent to	A*B(C D)

Colon, Dash, and Double Dash Operators

You can simplify the specification of a large model when some of your variables have a common prefix by using the colon (:) operator and the dash (-) operator. The dash operator enables you to list variables that are numbered sequentially, and the colon operator selects all variables with a given prefix. For example, if your data set contains the variables X1 through X9, the following MODEL statements are equivalent:

```
model Y = X1 X2 X3 X4 X5 X6 X7 X8 X9;
```

```
model Y = X1-X9;
```

```
model Y = X:;
```

If your data set contains only the three covariates X1, X2, and X9, then the colon operator selects all three variables:

```
model Y = X:;
```

However, the following specification returns an error because X3 through X8 are not in the data set:

```
model Y = X1-X9;
```

The double dash (- -) operator enables you to select variables that are stored sequentially in the SAS data set, whether or not they have a common prefix. You can use the CONTENTS procedure (see [Base SAS Procedures Guide](#)) to determine your variable ordering. For example, if you replace the dash in the preceding MODEL statement with a double dash, as follows, then all three variables are selected:

```
model Y = X1--X9;
```

If your data set contains the variables A, B, and C, then you can use the double dash operator to select these variables by specifying the following:

```
model Y = A--C;
```

GLM Parameterization of Classification Variables and Effects

Table 3.12 shows the types of effects that are available in procedures in this book; they are discussed in more detail in the following subsections. Let A, B, and C represent classification variables, and let X and Z represent continuous variables.

Table 3.12 Available Types of Effects

Effect	Example	Description
Intercept	Default	Intercept (unless the NOINT option is specified)
Regression	X Z	Continuous variables
Polynomial	X*Z	Interaction of continuous variables
Main	A B	CLASS variables
Interaction	A*B	Crossing of CLASS variables
Nested	A(B)	Main effect A nested within CLASS effect B
Continuous-by-class	X*A	Crossing of continuous and CLASS variables
Continuous-nesting-class	X(A)	Continuous variable X1 nested within CLASS variable A
General	X*Z*A(B)	Combinations of different types of effects

Table 3.13 shows some examples of MODEL statements that use various types of effects.

Table 3.13 MODEL Statement Effect Examples

Specification	Type of Model
model Y=X;	Simple regression
model Y=X Z;	Multiple regression
model Y=X X*X;	Polynomial regression
model Y=A;	One-way analysis of variance (ANOVA)
model Y=A B C;	Main-effects ANOVA
model Y=A B A*B;	Factorial ANOVA with interaction
model y=A B (A) C (B A) ;	Nested ANOVA
model Y=A X;	Analysis of covariance (ANCOVA)
model Y=A X (A) ;	Separate-slopes regression
model Y=A X X*A;	Homogeneity-of-slopes regression

Intercept

By default, linear models that are created by procedures in this book automatically include a column of 1s in **X**. This column corresponds to an intercept parameter. In many procedures, you can use the NOINT option in the MODEL statement to suppress this intercept. For example, the NOINT option is useful when the MODEL statement contains a classification effect and you want the parameter estimates to be in terms of the mean response for each level of that effect.

Regression Effects

Numeric variables or polynomial terms that involve them can be included in the model as regression effects (covariates). The actual values of such terms are included as columns of the relevant model matrices. You can use the [bar operator](#) along with a regression effect to generate polynomial effects. For example, $X \mid X \mid X$ expands to $X \ X^*X \ X^*X^*X$, which is a cubic model.

Main Effects

If a classification variable has m levels, the GLM parameterization generates m columns for its main effect in the model matrix. Each column is an indicator variable for a particular level. The order of the columns is the sort order of the values of their levels and can be controlled by the [ORDER=](#) option in the [CLASS](#) statement.

Table 3.14 is an example where β_0 denotes the intercept and A and B are classification variables that have two and three levels, respectively.

Table 3.14 Example of Main Effects

Data		I	A		B		
A	B	β_0	A1	A2	B1	B2	B3
1	1	1	1	0	1	0	0
1	2	1	1	0	0	1	0
1	3	1	1	0	0	0	1
2	1	1	0	1	1	0	0
2	2	1	0	1	0	1	0
2	3	1	0	1	0	0	1

There are usually more columns for these effects than there are degrees of freedom to estimate them. In other words, the GLM parameterization of main effects is *singular*.

Interaction Effects

Often a model includes interaction (crossed) effects to account for how the effect of a variable changes along with the values of other variables. With an interaction, the terms are first reordered to correspond to the order of the variables in the [CLASS](#) statement. Thus, B^*A becomes A^*B if A precedes B in the [CLASS](#) statement. Then, the GLM parameterization generates columns for all combinations of levels that occur in the data. The order of the columns is such that the rightmost variables in the interaction change faster than the leftmost variables (Table 3.15).

Table 3.15 Example of Interaction Effects

Data		I	A		B			A*B					
A	B	β_0	A1	A2	B1	B2	B3	A1B1	A1B2	A1B3	A2B1	A2B2	A2B3
1	1	1	1	0	1	0	0	1	0	0	0	0	0
1	2	1	1	0	0	1	0	0	1	0	0	0	0
1	3	1	1	0	0	0	1	0	0	1	0	0	0
2	1	1	0	1	1	0	0	0	0	0	1	0	0
2	2	1	0	1	0	1	0	0	0	0	0	1	0
2	3	1	0	1	0	0	1	0	0	0	0	0	1

In the preceding matrix, main-effects columns are not linearly independent of crossed-effects columns. In fact, the column space for the crossed effects contains the space of the main effect.

When your model contains many interaction effects, you might be able to code them more parsimoniously by using the **bar operator** (`|`). The bar operator generates all possible interaction effects. For example, `A | B | C` expands to `A B A*B C A*C B*C A*B*C`. To eliminate higher-order interaction effects, use the **at sign** (`@`) in conjunction with the bar operator. For example, `A | B | C | D@2` expands to `A B A*B C A*C B*C D A*D B*D C*D`.

Nested Effects

Nested effects are generated in the same manner as crossed effects. Hence, the design columns that are generated by the following two statements are the same (but the ordering of the columns is different):

```
model Y=A B(A) ;
```

```
model Y=A A*B;
```

The nesting operator in procedures in this book is more of a notational convenience than an operation that is distinct from crossing. Nested effects are typically characterized by the property that the nested variables do not appear as main effects. The order of the variables within nesting parentheses is made to correspond to the order of these variables in the **CLASS** statement. The order of the columns is such that variables outside the parentheses index faster than those inside the parentheses, and the rightmost nested variables index faster than the leftmost variables (Table 3.16).

Table 3.16 Example of Nested Effects

Data		I	A		B(A)					
A	B	β_0	A1	A2	B1A1	B2A1	B3A1	B1A2	B2A2	B3A2
1	1	1	1	0	1	0	0	0	0	0
1	2	1	1	0	0	1	0	0	0	0
1	3	1	1	0	0	0	1	0	0	0
2	1	1	0	1	0	0	0	1	0	0
2	2	1	0	1	0	0	0	0	1	0
2	3	1	0	1	0	0	0	0	0	1

Continuous-Nesting-Class Effects

When a continuous variable nests or crosses with a classification variable, the design columns are constructed by multiplying the continuous values into the design columns for the classification effect (Table 3.17).

Table 3.17 Example of Continuous-Nesting-Class Effects

Data		I	A		X(A)	
X	A	β_0	A1	A2	X(A1)	X(A2)
21	1	1	1	0	21	0
24	1	1	1	0	24	0
22	1	1	1	0	22	0
28	2	1	0	1	0	28

Table 3.17 *continued*

Data		I	A		X(A)	
19	2	1	0	1	0	19
23	2	1	0	1	0	23

This model estimates a separate intercept and a separate slope for X within each level of A.

Continuous-by-Class Effects

Continuous-by-class effects generate the same design columns as continuous-nesting-class effects. Table 3.18 shows the construction of the X*A effect. The two columns for this effect are the same as the columns for the X(A) effect in Table 3.17.

Table 3.18 Example of Continuous-by-Class Effects

Data		I	X	A		X*A	
X	A	β_0	X	A1	A2	X*A1	X*A2
21	1	1	21	1	0	21	0
24	1	1	24	1	0	24	0
22	1	1	22	1	0	22	0
28	2	1	28	0	1	0	28
19	2	1	19	0	1	0	19
23	2	1	23	0	1	0	23

You can use continuous-by-class effects together with pure continuous effects to test for homogeneity of slopes.

General Effects

An example that combines all the effects is X1*X2*A*B*C(D E). The continuous list comes first, followed by the crossed list, followed by the nested list in parentheses.

Effects might be renamed by the procedure to correspond to ordering rules. For example, B*A(E D) might be renamed A*B(D E) to satisfy the following:

- Classification variables that occur outside parentheses (crossed effects) are sorted in the order in which they appear in the CLASS statement.
- Variables within parentheses (nested effects) are sorted in the order in which they appear in the CLASS statement.

The sequencing of the parameters that are generated by an effect is determined by the variables whose levels are indexed faster:

- Variables in the crossed list index faster than variables in the nested list.
- Within a crossed or nested list, variables to the right index faster than variables to the left.

For example, suppose that a model includes four effects—A, B, C, and D—each having two levels, 1 and 2, and that the CLASS statement is as follows:

```
class A B C D;
```

Then the order of the parameters for the effect $B*A(C\ D)$, which is renamed $A*B(C\ D)$, is as follows:

$$\begin{aligned} A_1 B_1 C_1 D_1 &\rightarrow A_1 B_2 C_1 D_1 \rightarrow A_2 B_1 C_1 D_1 \rightarrow A_2 B_2 C_1 D_1 \rightarrow \\ A_1 B_1 C_1 D_2 &\rightarrow A_1 B_2 C_1 D_2 \rightarrow A_2 B_1 C_1 D_2 \rightarrow A_2 B_2 C_1 D_2 \rightarrow \\ A_1 B_1 C_2 D_1 &\rightarrow A_1 B_2 C_2 D_1 \rightarrow A_2 B_1 C_2 D_1 \rightarrow A_2 B_2 C_2 D_1 \rightarrow \\ A_1 B_1 C_2 D_2 &\rightarrow A_1 B_2 C_2 D_2 \rightarrow A_2 B_1 C_2 D_2 \rightarrow A_2 B_2 C_2 D_2 \end{aligned}$$

Note that first the crossed effects B and A are sorted in the order in which they appear in the **CLASS** statement so that A precedes B in the parameter list. Then, for each combination of the nested effects in turn, combinations of A and B appear. The B effect changes fastest because it is rightmost in the cross list. Then A changes next fastest, and D changes next fastest. The C effect changes most slowly because it is leftmost in the nested list.

Nonsingular Parameterization

When a parameterization of main effects provides the same number of columns for the effects as there are degrees of freedom to estimate them, the parameterization is called *nonsingular*. A variety of nonsingular parameterizations for classification effects are available for many procedures in this book. In most of these procedures you use the **PARAM=** option in the **CLASS** statement to specify the parameterization.

Consider a model with one **CLASS** variable A that has four levels, 1, 2, 5, and 7. Details of the possible choices for the **PARAM=** option follow.

REFERENCE

Three columns are created to indicate group membership of the nonreference levels. For the reference level, all three dummy variables have a value of 0. For example, if the reference level is 7 (REF=7), the design matrix columns for A are as follows.

Reference Coding			
Design Matrix			
A	A1	A2	A5
1	1	0	0
2	0	1	0
5	0	0	1
7	0	0	0

Parameter estimates of **CLASS** main effects that use the reference coding scheme estimate the difference in the effect of each nonreference level compared to the effect of the reference level.

EFFECT

Three columns are created to indicate group membership of the nonreference levels. For the reference level, all three dummy variables have a value of -1. For example, if the reference level is 7 (REF=7), the design matrix columns for A are as follows.

Effect Coding			
A	Design Matrix		
	A1	A2	A5
1	1	0	0
2	0	1	0
5	0	0	1
7	-1	-1	-1

Parameter estimates of CLASS main effects that use the effect coding scheme estimate the difference in the effect of each nonreference level compared to the average effect over all four levels.

ORDINAL | THERMOMETER Three columns are created to indicate group membership of the higher levels of the effect. For the first level of the effect (which for A is 1), all three dummy variables have a value of 0. The design matrix columns for A are as follows.

Ordinal Coding			
A	Design Matrix		
	A2	A5	A7
1	0	0	0
2	1	0	0
5	1	1	0
7	1	1	1

The first level of the effect is a control or baseline level. Parameter estimates of CLASS main effects, using the ORDINAL coding scheme, estimate the differences between effects of successive levels. When the parameters have the same sign, the effect is monotonic across the levels.

POLYNOMIAL | POLY Three columns are created. The first represents the linear term (x), the second represents the quadratic term (x^2), and the third represents the cubic term (x^3), where x is the level value. If the CLASS levels are not numeric, they are translated into 1, 2, 3, ... according to their sort order. The design matrix columns for A are as follows.

Polynomial Coding			
A	Design Matrix		
	APOLY1	APOLY2	APOLY3
1	1	1	1
2	2	4	8
5	5	25	125
7	7	49	343

ORTHEFFECT

The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=EFFECT. The design matrix columns for A are as follows.

Orthogonal Effect Coding			
Design Matrix			
A	AOEFF1	AOEFF2	AOEFF3
1	1.41421	-0.81650	-0.57735
2	0	1.63299	-0.57735
5	0	0	1.73205
7	-1.41421	-0.81649	-0.57735

ORTHORDINAL | ORTHOTHERM The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=ORDINAL. The design matrix columns for A are as follows.

Orthogonal Ordinal Coding			
Design Matrix			
A	AOORD1	AOORD2	AOORD3
1	-1.73205	0	0
2	0.57735	-1.63299	0
5	0.57735	0.81650	-1.41421
7	0.57735	0.81650	1.41421

ORTHPOLY The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=POLY. The design matrix columns for A are as follows.

Orthogonal Polynomial Coding			
Design Matrix			
A	AOPOLY1	AOPOLY2	AOPOLY5
1	-1.15311	0.90712	-0.92058
2	-0.73380	-0.54041	1.47292
5	0.52414	-1.37034	-0.92058
7	1.36277	1.00363	0.36823

ORTHREF The columns are obtained by applying the Gram-Schmidt orthogonalization to the columns for PARAM=REFERENCE. The design matrix columns for A are as follows.

Orthogonal Reference Coding			
Design Matrix			
A	AOREF1	AOREF2	AOREF3
1	1.73205	0	0
2	-0.57735	1.63299	0
5	-0.57735	-0.81650	1.41421
7	-0.57735	-0.81650	-1.41421

Class Variable Parameterization with Unbalanced Designs

Procedures in this book initially parameterize the CLASS variables by looking at the levels of the variables across the complete data set. If you have an *unbalanced* replication of levels across variables or BY groups, then the design matrix and the parameter interpretation might be different from what you expect. For example, suppose you have a model that has one CLASS variable A with three levels (1, 2, and 3), and another CLASS variable B with two levels (1 and 2). If the third level of A occurs only with the first level of B, if you use the EFFECT parameterization, and if your model contains the effect A(B) and an intercept, then the design for A within the second level of B is not a differential effect. In particular, the design looks like the following:

		Design Matrix			
B	A	A(B=1)		A(B=2)	
		A1	A2	A1	A2
1	1	1	0	0	0
1	2	0	1	0	0
1	3	-1	-1	0	0
2	1	0	0	1	0
2	2	0	0	0	1

Procedures in this book detect linear dependency among the last two design variables and set the parameter for A2(B=2) to 0, resulting in an interpretation of these parameters as if they were reference- or dummy-coded. The REFERENCE or GLM parameterization might be more appropriate for such problems.

Model Selection Methods

The model selection methods are specified in the **METHOD=** option in the **SELECTION** statement. The following methods are available, although specific procedures might support only a subset of these methods. Furthermore, the examples in this section refer to fit criteria that might not be supported by a specific procedure.

Forward Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=FORWARD specifies the forward selection technique, which begins with just the intercept and then sequentially adds the effect that most improves the fit. The process terminates when no significant improvement can be obtained by adding any effect.

In the traditional implementation of forward selection, the statistic that is used to determine whether to add an effect is the significance level of a hypothesis test that reflects an effect's contribution to the model if it is included. At each step, the effect that is most significant is added. The process stops when the significance level for adding any effect is greater than some specified entry significance level.

An alternative approach to address the critical problem of when to stop the selection process that is implemented by the procedures in this book is to assess the quality of the models that are produced by the forward

selection method and choose the model from this sequence that “best” balances goodness of fit against model complexity. You can use several criteria for this purpose. These criteria fall into two groups—information criteria and criteria that are based on out-of-sample prediction performance.

It is important to keep in mind that forward selection bases the decision about what effect to add at any step by considering models that differ by one effect from the current model. This search paradigm cannot guarantee reaching a “best” subset model. Furthermore, the add decision is greedy in the sense that the effect that is deemed most significant is the effect that is added. The optimality of this one step choice depends on the optimality of the selection criterion for the model that contains that effect. You can specify the criterion to optimize at each step by using the **SELECT=** option. For example, the following statement requests that at each step the effect that is added be the one that produces a model that has the smallest value of the SBC:

```
selection method=forward(select=SBC);
```

When all effects are variables (that is, effects have one degree of freedom and no hierarchy), using AIC, AICC, or SBC as the selection criterion for forward selection produces the same sequence of additions. However, if the degrees of freedom contributed by different effects are not constant or if an out-of-sample prediction-based criterion is used, then different sequences of additions might be obtained.

If you specify only the **SELECT=** criterion, then this criterion is also used to decide when to stop the selection process. In the previous example, not only do effects enter based on the SBC value, but the selection terminates when SBC has a local minimum.

You use the **CHOOSE=** option to specify the criterion for selecting one model from the sequence of models produced. If you do not specify a **CHOOSE=** criterion, then the model at the final step is the selected model.

For example, if you specify the following statement, then forward selection terminates at the step where the SBC reaches a (local) minimum:

```
selection method=forward(select=SBC choose=AIC);
```

However, the selected model is the first one that has the minimum value of Akaike’s information criterion. In some cases, this minimum value might occur at a step much earlier than the final step. In other cases, the AIC might start increasing only if more steps are performed—that is, a larger value is used for the significance level for entry. If you want to minimize AIC, then too many steps are performed in the former case and too few in the latter case. To address this issue, you can use the **STOP=** option to specify a stopping criterion. When you specify a stopping criterion, forward selection continues until a local extremum of the stopping criterion in the sequence of models generated is reached. To be deemed a local extremum, a criterion value at a particular step must be better than its value at the next n steps, where n is known as the “stop horizon.” By default, the stop horizon is three steps, but you can change this by specifying the **STOPHORIZON=** option.

For example, if you specify the following statement, then forward selection terminates at the step where the effect to be added at the next step would produce a model that has an AIC statistic larger than the AIC statistic of the current model:

```
selection method=forward(select=SBC stop=AIC) stophorizon=1;
```

In most cases, provided that the stop horizon is long enough that the local extremum of the named criterion occurs before the final step, specifying either of the following statements selects the same model, but more steps are done in the first case:

```
selection method=forward(select=SL choose=CRITERION);
selection method=forward(select=SL stop=CRITERION);
```

In some cases, there might be a better local extremum that cannot be reached if you specify the **STOP=** option but can be found if you use the **CHOOSE=** option. Also, you can use the **CHOOSE=** option in preference to the **STOP=** option if you want to examine how the named criterion behaves as you move beyond the step where the first local minimum of this criterion occurs.

You can specify both the **CHOOSE=** and **STOP=** options. You can also use these options together with options that specify size-based limits on the selected model. You might want to consider models that are generated by forward selection and have at most some fixed number of effects, but select from within this set based on a criterion that you specify. For example, specifying the following statements requests that forward selection continue until there are 20 effects in the final model and chooses among the sequence of models the one that has the smallest value of the corrected Akaike's information criterion:

```
selection method=forward(stop=none maxeffects=20 choose=AICC);
```

You can also combine these options to select a model where one of two conditions is met. For example, the following statement chooses whatever occurs first between a local minimum of the Schwarz Bayesian information criterion (SBC) and a local minimum of the corrected Akaike's information criterion (AICC):

```
selection method=forward(stop=AICC choose=SBC);
```

You can find discussion and references to studies about criteria for variable selection in Burnham and Anderson (2002), along with some cautions and recommendations.

Examples of Forward Selection Specifications

The following statement adds effects that at each step produce the lowest value of the SBC statistic and stops at the step where adding any effect would increase the SBC statistic:

```
selection method=forward stophorizon=1;
```

The following statement adds effects that at each step produce the lowest value of the AIC statistic and stops at the first step whose AIC value is smaller than the AIC value at the next three steps:

```
selection=forward(select=AIC);
```

Backward Elimination

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=BACKWARD specifies the backward elimination technique. This technique starts from the full model, which includes all independent effects. Then effects are deleted one by one until a stopping condition is satisfied. At each step, the effect that shows the smallest contribution to the model is deleted.

Just as with forward selection, you can use the **SELECT=** option to change the criterion that is used to assess effect contributions. You can also specify a stopping criterion in the **STOP=** option and use a **CHOOSE=** option to provide a criterion for selecting among the sequence of models produced. For more information, see the discussion in the section “[Forward Selection](#)” on page 59.

Examples of Backward Selection Specifications

The following statement removes effects that at each step produce the largest value of the Schwarz Bayesian information criterion (SBC) statistic and stops at the step where removing any effect increases the SBC statistic:

```
selection method=backward stophorizon=1;
```

The following statement bases removal of effects on the corrected Akaike's information criterion (AICC) and stops at the first step whose AICC value is smaller than the AICC value at the next three steps:

```
selection method=backward(select=AICC) ;
```

Stepwise Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=STEPWISE specifies the stepwise method, which modifies the forward selection technique by allowing effects already in the model to be removed.

Just as you can in forward selection and backward elimination, you can use the **SELECT=** option to change the criterion that is used to assess effect contributions. You can also use the **STOP=** option to specify a stopping criterion and use a **CHOOSE=** option to provide a criterion for selecting among the sequence of models produced. For more information, see the section “[Forward Selection](#)” on page 59.

The procedures in this book support further modification to the stepwise method. In the standard stepwise method, no effect can enter the model if removing any effect currently in the model would yield an improved value of the selection criterion. In the modification, you can use the **COMPETITIVE** option to specify that addition and deletion of effects should be treated competitively. The selection criterion is evaluated for all models that are produced by deleting an effect from the current model or by adding an effect to this model. The action that most improves the selection criterion is the action taken.

Examples of Stepwise Selection Specifications

The following statement requests stepwise selection based on the SBC criterion:

```
selection method=stepwise;
```

First, if removing any effect yields a model that has a lower SBC statistic than the current model, then the effect that produces the smallest SBC statistic is removed. If removing any effect increases the SBC statistic, then provided that adding some effect lowers the SBC statistic, the effect that produces the model that has the lowest SBC is added.

The following statement requests the stepwise method where effects enter and leave based on the AIC statistic, but with the following extra check: if any effect to be added or removed yields a model whose SBC statistic is greater than the SBC statistic of the current model, then the stepwise method terminates at the current model.

```
selection method=stepwise(select=AIC stop=SBC) stophorizon=1;
```

The following statement requests stepwise selection that is based on the AICC criterion and treats additions and deletions competitively:


```
selection method=stepwise(select=AICC competitive);
```

Each step evaluates the AICC statistics that correspond to the removal of any effect in the current model or the addition of any effect to the current model and chooses the addition or removal that produced the minimum value, provided that this minimum is lower than the AICC statistic of the current model.

Forward-Swap Selection

This section applies to the following procedures: CNTSELECT and SEVSELECT.

METHOD=FORWARDSSWAP specifies the forward-swap selection method, which is an extension of the forward selection method. The forward-swap selection method incorporates steps that improve a model by replacing an effect in the model with an effect that is not in the model.

The forward-swap selection method begins by finding the one-effect model that produces the best value of the selection criterion. Then another effect (the one that yields the greatest improvement in the selection criterion) is added. After the two-effect model is obtained, each of the effects in the model is compared to each effect that is not in the model. For each comparison, the forward-swap method determines whether removing one effect and replacing it with the other effect improves the selection criterion. After comparing all possible swaps, the forward-swap method makes the swap that produces the greatest improvement in the selection criterion. Comparisons begin again, and the process continues until the forward-swap method finds that no other swap could improve the selection criterion. Thus, the two-variable model that is produced is considered the “best” two-variable model that the technique can find. Another variable is then added to the model, and the comparing-and-swapping process is repeated to find the “best” three-variable model, and so on.

The difference between the stepwise selection method and the forward-swap selection method is that all swaps are evaluated before any addition is made in the forward-swap method. In the stepwise selection method, the “worst” effect might be removed without considering what adding the “best” remaining effects might accomplish. Because the forward-swap method needs to examine all possible pairwise effect swaps at each step of the selection process, the forward-swap method is much more computationally expensive than the stepwise selection method; it might not be appropriate for models that contain a large number of effects.

Informative Missingness

This section applies to the following procedures: CNTSELECT, CPANEL, CQLIM, CSPATIALREG, and SEVSELECT.

The concept of informative missingness is one way to account for missing values in statistical analyses and, in particular, statistical modeling. Missing values can be a problem because they reduce the amount of available data. When you work with classification variables (factors, which are leveled variables), you can treat a missing value as an actual level of the variable and allow it to participate in the analysis.

However, when continuous variables have missing values, the observation is removed from the analysis. In data that have many missing values, removing observations can reduce the amount of available data greatly,

and the sets of observations used in one model versus another model can vary based on which variables are included in the model.

Of course, there are many reasons for missing values, and substituting values for missing values has to be done with caution. For example, the famous Framingham Heart study data set contains 5,209 observations on subjects in a longitudinal study that helped understand the relationship between smoking, cholesterol, and coronary heart disease. One of the variables in the data set is `AgeCHDdiag`. This variable represents the age at which a patient was diagnosed with coronary heart disease (CHD). If you include this variable in a statistical model, only 1,449 observations are available, because the value cannot be observed unless a patient has experienced CHD. Including this variable acts as a filter that reduces the analysis set to the subjects who have CHD. You cannot impute the value for subjects where the variable has a missing value, because you cannot impute an age at which someone who has not had CHD would have contracted coronary heart disease.

With informative missingness, you are not so much substituting imputed values for the missing values as you are modeling the missingness. Consider a simple linear regression model:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Suppose that some of the values for the regressor variable x are missing. The fitted model uses only observations for which y and x have been observed.

In order to predict the outcome y for an observation that has a missing x , either you assume that y is missing or you substitute a value (such as the average value, $\bar{x}$) for the missing x . Because the estimate for the intercept is in the simple linear regression model, the predicted value would be the average response of the nonmissing values, $\bar{y}$.

With informative missingness, you extend the model by adding extra effects for each effect that contains at least one continuous variable. In the simple linear regression model, you add one column to the model and slightly change the content of the x variable:

$$y = \beta_0 + \beta_1 x^* + \beta_2 x_{miss} + \epsilon_1$$

The variable x^* contains the original values of x if they are not missing, and the average of x otherwise:

$$x^* = \begin{cases} x & \text{if } x \text{ is not missing} \\ \bar{x} & \text{otherwise} \end{cases}$$

The variable x_{miss} is a dummy variable whose value is 1 when x is missing, and 0 otherwise:

$$x_{miss} = \begin{cases} 1 & \text{if } x \text{ is missing} \\ 0 & \text{otherwise} \end{cases}$$

The fitted model is not the same model that results from substituting $\bar{x}$ for the missing values during training, because the model that simply substitutes $\bar{x}$ for the missing values is

$$y = \beta_0 + \beta_1 x^* + \epsilon_2$$

The informative missing model has an extra parameter, and unless all values of x_{miss} are 0 (in which case there are no missing values), the informative missing model has a higher R-square value, because it picks up more variation.

The parameter estimate for β_2 measures the amount by which the predicted value differs from a predicted value at $\bar{x}$.

Multithreading

Threading refers to the organization of computational work into multiple tasks (processing units that can be scheduled by the operating system). A task is associated with a thread. Multithreading refers to the concurrent execution of threads. When multithreading is possible, substantial performance gains can be realized compared to sequential (single-threaded) execution. The number of threads spawned by a procedure in this book is determined by your installation.

The tasks that are multithreaded by procedures in this book are primarily defined by dividing the data that are processed on a single machine among the threads—that is, the procedures implement multithreading through a data-parallel model. For example, if the input data table has 1,000 observations and the procedure is running on four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following (not all operations are required for all procedures):

- variable levelization
- effect levelization
- formation of the initial crossproducts matrix
- formation of approximate Hessian matrices for candidate evaluation during model selection
- objective function calculation
- gradient calculation
- Hessian calculation
- scoring of observations

In addition, operations on matrices such as sweeps can be multithreaded provided that the matrices are of sufficient size to realize performance benefits from managing multiple threads for the particular matrix operation.

Choosing an Optimization Algorithm

This section applies to the following procedures: SEVSELECT.

First- or Second-Order Algorithms

The factors that go into choosing a particular optimization technique for a particular problem are complex. Trial and error can be involved.

For many optimization problems, computing the gradient takes more computer time than computing the function value. Computing the Hessian sometimes takes *much* more computer time and memory than computing the gradient, especially when there are many decision variables. Unfortunately, optimization techniques that do not use some kind of Hessian approximation usually require many more iterations than techniques that do use a Hessian matrix, and as a result the total run time of these techniques is often longer. Techniques that do not use the Hessian also tend to be less reliable. For example, they can terminate more easily at stationary points than at global optima.

Table 3.19 shows which derivatives are required for each optimization technique.

Table 3.19 Derivatives Required

Algorithm	First-Order	Second-Order
TRUREG	x	x
NEWRAP	x	x
NRRIDG	x	x
QUANEW	x	-
DBLDOG	x	-
CONGRA	x	-
NMSIMP	-	-

The second-derivative methods TRUREG, NEWRAP, and NRRIDG are best for small problems for which the Hessian matrix is not expensive to compute. Sometimes the NRRIDG algorithm can be faster than the TRUREG algorithm, but TRUREG can be more stable. The NRRIDG algorithm requires only one matrix with $p(p + 1)/2$ double words; TRUREG and NEWRAP require two such matrices. Here, p denotes the number of parameters in the optimization.

The first-derivative methods QUANEW and DBLDOG are best for medium-sized problems for which the objective function and the gradient can be evaluated much faster than the Hessian. In general, the QUANEW and DBLDOG algorithms require more iterations than TRUREG, NRRIDG, and NEWRAP, but each iteration can be much faster. The QUANEW and DBLDOG algorithms require only the gradient to update an approximate Hessian, and they require slightly less memory than TRUREG or NEWRAP.

The first-derivative method CONGRA is best for large problems for which the objective function and the gradient can be computed much faster than the Hessian and for which too much memory is required to store the (approximate) Hessian. In general, the CONGRA algorithm requires more iterations than QUANEW or DBLDOG, but each iteration can be much faster. Because CONGRA requires only a factor of p double-word memory, many large applications can be solved only by CONGRA.

The no-derivative method NMSIMP is best for small problems for which derivatives are not continuous or are very difficult to compute.

Each optimization method uses one or more convergence criteria that determine when it has converged. An algorithm is considered to have converged when any one of the convergence criteria is satisfied. For example, under the default settings, the QUANEW algorithm converges if `ABSGCONV` < 1E-5, `FCONV` < $2 \times \epsilon$, or `GCONV` < 1E-8.

By default, procedures in this book apply the NRRIDG algorithm because it can take advantage of multi-threading in Hessian computations and inversions. If the number of parameters becomes large, specifying the `TECHNIQUE=QUANEW` option, which is a first-order method with good overall properties, is recommended.

Algorithm Descriptions

The following subsections provide details about each optimization technique and follow the same order as Table 3.19.

Trust Region Optimization (TRUEG)

The trust region method uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function $f(\boldsymbol{\beta})$ have continuous first- and second-order derivatives inside the feasible region.

The trust region method iteratively optimizes a quadratic approximation to the nonlinear objective function within a hyperelliptic trust region with radius Δ that constrains the step size that corresponds to the quality of the quadratic approximation. The trust region method is implemented based on Dennis, Gay, and Welsch (1981), Gay (1983), and Moré and Sorensen (1983).

The trust region method performs well for small- to medium-sized problems, and it does not need many function, gradient, and Hessian calls. However, if the computation of the Hessian matrix is computationally expensive, one of the dual quasi-Newton or conjugate gradient algorithms might be more efficient.

Newton-Raphson Optimization with Line Search (NEWRAP)

The NEWRAP technique uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function have continuous first- and second-order derivatives inside the feasible region. If second-order derivatives are computed efficiently and precisely, the NEWRAP method can perform well for medium-sized to large problems, and it does not need many function, gradient, and Hessian calls.

This algorithm uses a pure Newton step when the Hessian is positive-definite and when the Newton step reduces the value of the objective function successfully. Otherwise, a combination of ridging and line search is performed to compute successful steps. If the Hessian is not positive-definite, a multiple of the identity matrix is added to the Hessian matrix to make it positive-definite (Eskow and Schnabel 1991).

In each iteration, a line search is performed along the search direction to find an approximate optimum of the objective function. The line-search method uses quadratic interpolation and cubic extrapolation.

Newton-Raphson Ridge Optimization (NRRIDG)

The NRRIDG technique uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$ and the Hessian matrix $\mathbf{H}(\boldsymbol{\beta}^{(k)})$; thus, it requires that the objective function have continuous first- and second-order derivatives inside the feasible region.

This algorithm uses a pure Newton step when the Hessian is positive-definite and when the Newton step reduces the value of the objective function successfully. If at least one of these two conditions is not satisfied, a multiple of the identity matrix is added to the Hessian matrix.

Because the NRRIDG technique uses an orthogonal decomposition of the approximate Hessian, each iteration of NRRIDG can be slower than that of the NEWRAP technique, which works with a Cholesky decomposition. However, NRRIDG usually requires fewer iterations than NEWRAP.

The NRRIDG method performs well for small- to medium-sized problems, and it does not require many function, gradient, and Hessian calls. However, if the computation of the Hessian matrix is computationally expensive, one of the dual quasi-Newton or conjugate gradient algorithms might be more efficient.

Quasi-Newton Optimization (QUANEW)

The dual quasi-Newton method uses the gradient $\mathbf{g}(\boldsymbol{\beta}^{(k)})$, and it does not need to compute second-order derivatives because they are approximated. It works well for medium-sized to moderately large optimization problems, where the objective function and the gradient can be computed much faster than the Hessian. However, in general the QUANEW technique requires more iterations than the TRUEG, NEWRAP, and NRRIDG techniques, which compute second-order derivatives. The QUANEW technique provides an appropriate balance between the speed and stability required for most nonlinear mixed model applications.

The QUANEW technique implemented by procedures in this book is the dual quasi-Newton algorithm, which updates the Cholesky factor of an approximate Hessian.

In each iteration, a line search is performed along the search direction to find an approximate optimum. The line-search method uses quadratic interpolation and cubic extrapolation to obtain a step size α that satisfies the Goldstein conditions (Fletcher 1987). One of the Goldstein conditions can be violated if the feasible region defines an upper limit of the step size. Violating the left-side Goldstein condition can affect the positive-definiteness of the quasi-Newton update. In that case, either the update is skipped or the iterations are restarted with an identity matrix, resulting in the steepest descent or ascent search direction.

Double-Dogleg Optimization (DBLDOG)

The double-dogleg optimization method combines the ideas of the quasi-Newton and trust region methods. In each iteration, the double-dogleg algorithm computes the step $s^{(k)}$ as the linear combination of the steepest descent or ascent search direction $s_1^{(k)}$ and a quasi-Newton search direction $s_2^{(k)}$:

$$s^{(k)} = \alpha_1 s_1^{(k)} + \alpha_2 s_2^{(k)}$$

The step is requested to remain within a prespecified trust region radius (Fletcher 1987, p. 107). Thus, the DBLDOG subroutine uses the dual quasi-Newton update but does not perform a line search.

The double-dogleg optimization technique works well for medium-sized to moderately large optimization problems, where the objective function and the gradient can be computed much faster than the Hessian. The implementation is based on Dennis and Mei (1979) and Gay (1983), but it is extended for dealing with boundary and linear constraints. The DBLDOG technique generally requires more iterations than the TRUREG, NEWRAP, and NRRIDG techniques, which require second-order derivatives; however, each of the DBLDOG iterations is computationally cheap. Furthermore, the DBLDOG technique requires only gradient calls for the update of the Cholesky factor of an approximate Hessian.

Conjugate Gradient Optimization (CONGRA)

Second-order derivatives are not required by the CONGRA algorithm and are not even approximated. The CONGRA algorithm can be expensive in function and gradient calls, but it requires only $O(p)$ memory for unconstrained optimization. In general, many iterations are required to obtain a precise solution, but each of the CONGRA iterations is computationally cheap.

The CONGRA subroutine should be used for optimization problems with large p . For the unconstrained or boundary-constrained case, CONGRA requires only $O(p)$ bytes of working memory, whereas all other optimization methods require order $O(p^2)$ bytes of working memory. During p successive iterations, uninterrupted by restarts or changes in the working set, the conjugate gradient algorithm computes a cycle of p conjugate search directions. In each iteration, a line search is performed along the search direction to find an approximate optimum of the objective function. The line-search method uses quadratic interpolation and cubic extrapolation to obtain a step size α that satisfies the Goldstein conditions. One of the Goldstein conditions can be violated if the feasible region defines an upper limit for the step size.

Nelder-Mead Simplex Optimization (NMSIMP)

The Nelder-Mead simplex method does not use any derivatives and does not assume that the objective function has continuous derivatives. The objective function itself needs to be continuous. This technique is quite expensive in the number of function calls, and it might be unable to generate precise results for $p \gg 40$.

The original Nelder-Mead simplex algorithm is implemented and extended to boundary constraints. This algorithm does not compute the objective for infeasible points, but it changes the shape of the simplex

adapting to the nonlinearities of the objective function. This change contributes to an increased speed of convergence and uses a special termination criterion.

References

- Browne, M. W. (1982). “Covariance Structures.” In *Topics in Applied Multivariate Analysis*, edited by D. M. Hawkins, 72–141. Cambridge: Cambridge University Press.
- Burnham, K. P., and Anderson, D. R. (2002). *Model Selection and Multimodel Inference*. 2nd ed. New York: Springer-Verlag.
- Dennis, J. E., Gay, D. M., and Welsch, R. E. (1981). “An Adaptive Nonlinear Least-Squares Algorithm.” *ACM Transactions on Mathematical Software* 7:348–368.
- Dennis, J. E., and Mei, H. H. W. (1979). “Two New Unconstrained Optimization Algorithms Which Use Function and Gradient Values.” *Journal of Optimization Theory and Applications* 28:453–482.
- Eskow, E., and Schnabel, R. B. (1991). “Algorithm 695: Software for a New Modified Cholesky Factorization.” *ACM Transactions on Mathematical Software* 17:306–312.
- Fletcher, R. (1987). *Practical Methods of Optimization*. 2nd ed. Chichester, UK: John Wiley & Sons.
- Gay, D. M. (1983). “Subroutines for Unconstrained Minimization.” *ACM Transactions on Mathematical Software* 9:503–524.
- Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer-Verlag.
- Moré, J. J., and Sorensen, D. C. (1983). “Computing a Trust-Region Step.” *SIAM Journal on Scientific and Statistical Computing* 4:553–572.
- Searle, S. R. (1971). *Linear Models*. New York: John Wiley & Sons.

Chapter 4

The CARIMA Procedure

Contents

Overview: CARIMA Procedure	72
Using CAS Sessions and CAS Engine Librefs	72
Getting Started: CARIMA Procedure	73
Syntax: CARIMA Procedure	74
Functional Summary	74
PROC CARIMA Statement	76
BY Statement	77
ESTIMATE Statement	77
FORECAST Statement	79
ID Statement	79
IDENTIFY Statement	83
Details: CARIMA Procedure	83
Accumulation	83
Missing Value Interpretation	85
Parameter Estimation	85
Transformations	85
Forecasting	86
Data Table Output	86
Printed Output	88
ODS Table Names	88
Example: CARIMA Procedure	89
Example 4.1: Seasonal Model for the Airline Series	89
References	92

Overview: CARIMA Procedure

The CARIMA procedure analyzes and forecasts equally spaced univariate time series data, by using the autoregressive integrated moving-average (ARIMA) or autoregressive moving-average (ARMA) model.

An ARIMA model predicts a value in a response time series as a linear combination of its own past values and past errors (also called shocks or innovations).

The ARIMA approach was first popularized by Box and Jenkins (1976), and ARIMA models are often referred to as Box-Jenkins models.

The CARIMA procedure provides a comprehensive set of tools for parameter estimation and for forecasting univariate time series. It also offers a high degree of flexibility in the types of ARIMA models that you can analyze; it supports seasonal, subset, and factored ARIMA models. For each fitted model, PROC CARIMA prints parameter estimates along with standard errors and *p*-values, and it prints multistep-ahead variable forecasts along with confidence intervals.

Before you use PROC CARIMA, you should exercise care and judgment and be familiar with Box-Jenkins methods.

The ARIMA class of time series models is complex and powerful, and some degree of expertise is needed to use them correctly.

PROC CARIMA requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CARIMA Procedure

This section outlines the use of the CARIMA procedure and gives a cursory description of the ARIMA modeling process for readers who are less familiar with these methods.

Given an input data table that contains numerous time series variables recorded at a specific frequency, the CARIMA procedure can fit an ARIMA model to the series and produce forecasts.

For example, suppose that the input data table `mycas.Sales` contains sales data that were recorded monthly, the variable that represents time is `Date`, and the forecasts are to be recorded in the output data table `mycas.NextYear`. If you believe that sales in the current month are affected only by sales in the previous month, then you can use the CARIMA procedure to fit an AR(1) model and forecasts sales for the next four months as follows:

```
data mycas.sales;
  format date date9.;
  input date : date9. shoes socks laces dresses
        coats shirts ties belts hats blouses;
  datalines;
01JAN1994 3557 3718 6368.80 575 987 10.8200 15.0000 102.600 12410 15013
01FEB1994 5128 4174 8123.20 565 1000 12.1200 15.1000 99.900 13556 12413
01MAR1994 5222 4482 7807.20 406 1005 11.7800 15.3000 102.000 11063 12752

... more lines ...

01DEC1998 5399 4795 6075.30 614 1239 21.2000 1.9000 79.700 23004 22044
01JAN1999 6405 4981 6812.10 607 1196 20.7000 14.9000 99.900 20583 26093
;

proc carima data=mycas.sales outfor=mycas.nextyear;
  id date interval=month;
  identify _numeric_;
  estimate p =1;
  forecast lead=4;
```

```
run;
```

These statements assume that the CAS engine libref is named `mycas`, but you can substitute any appropriately named CAS engine libref. These statements generate forecasts for every numeric variable in the input data table `mycas.Sales` for the next four months and store these forecasts in the output data table `mycas.NextYear`. Other output data tables can be specified to store the parameter estimates and summary data.

The CARIMA procedure can forecast time series data, whose observations are equally spaced by a specific time interval (for example, monthly or weekly), or transactional data, whose observations are not spaced with respect to any particular time interval.

Given an input data table that contains transactional variables that are not recorded at any specific frequency, the CARIMA procedure accumulates the data to a specific time interval and forecasts the accumulated series. For an example, see “[Example 4.1: Seasonal Model for the Airline Series](#)” on page 89.

Syntax: CARIMA Procedure

The following statements are available in the CARIMA procedure:

```
PROC CARIMA options ;
  BY variables ;
  ID variable INTERVAL= interval < options > ;
  IDENTIFY variable-list ;
  ESTIMATE < options > ;
  FORECAST < options > ;
```

The PROC CARIMA and IDENTIFY statements are required. The CARIMA procedure supports multiple IDENTIFY, ESTIMATE, and FORECAST statements. The FORECAST statement forecasts all variables in the *variable-list* that is specified in the preceding IDENTIFY statement according to the model that is specified in the preceding ESTIMATE statement. If no ESTIMATE statement is specified before a FORECAST statement, then prediction is carried out according to a white noise model.

Functional Summary

The statements and options that control the CARIMA procedure are summarized in [Table 4.1](#).

Table 4.1 Functional Summary

Description	Statement	Option
Statements		
Specify data tables and options	PROC CARIMA	
Request BY-group processing	BY	
Specify the variables to model	IDENTIFY	
Specify the variables to estimate	ESTIMATE	
Specify the variables to forecast	FORECAST	

Table 4.1 *continued*

Description	Statement	Option
Data Table Options		
Specify the input data table	PROC CARIMA	DATA =
Specify the output data table	PROC CARIMA	OUT =
Specify parameter output data table	PROC CARIMA	OUTEST =
Specify forecast output data table	PROC CARIMA	OUTFOR =
Specify summary output data table	PROC CARIMA	OUTSUM=
Accumulation Options		
Specify the accumulation frequency	ID	INTERVAL =
Specify the interval alignment	ID	ALIGN =
Specify the starting time ID value	ID	START =
Specify the ending time ID value	ID	END =
Specify the accumulation statistic	ID, FORECAST	ACCUMULATE =
Specify how to interpret missing values	ID, FORECAST	SETMISSING =
Estimation Options		
Specify the convergence criterion	ESTIMATE	CONVERGE =
Specify the perturbation value for computing numerical derivatives	ESTIMATE	DELTA =
Specify the degrees of differencing	ESTIMATE	DIFF =
Specify the maximum number of iterations	ESTIMATE	MAXITER =
Specify the estimation method	ESTIMATE	METHOD =
Specify a constant term	ESTIMATE	MU =
Specify whether to include an intercept in the model	ESTIMATE	NOINT =
Specify whether to restrict candidate estimates for the AR and MA parameter estimates to stationarity and invertibility regions	ESTIMATE	NOSTABLE =
Specify the AR order	ESTIMATE	P =
Specify the MA order	ESTIMATE	Q =
Specify the singularity criterion	ESTIMATE	SINGULAR =
Specify the transformation type	ESTIMATE	TRANSFORM =
Forecasting Horizon		
Specify the forecast horizon or lead	FORECAST	LEAD =
Specify the confidence level	FORECAST	ALPHA =
Miscellaneous Options		
Specify the time ID format	ID	FORMAT =
Specify the method for trimming the data in the BY groups	ID	TRIMID =

The following sections describe the PROC CARIMA statement and then describe the other statements in alphabetical order.

PROC CARIMA Statement

PROC CARIMA *options* ;

You can specify the following *options*:

DATA=*CAS-libref.data-table*

names the input data table for PROC CARIMA to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 72.

data-table specifies the name of the input data table.

OUT=*CAS-libref.data-table*

names the output data table to contain the forecasts of the variables specified in the subsequent FORECAST statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 72.

If an ID statement is specified, its variable is also included in the OUT= data table and its values are accumulated on the basis of the ACCUMULATE= option in the ID statement.

OUTEST=*CAS-libref.data-table*

names the output data table to contain the model parameter estimates and the associated test statistics and probability values. The OUTEST= data table is useful for evaluating the significance of the model parameters and understanding the model dynamics.

OUTFOR=*CAS-libref.data-table*

names the output data table to contain the forecast time series components (actual, predicted, lower confidence limit, upper confidence limit, prediction error, and prediction standard error). The OUTFOR= data table is useful for displaying the forecasts in tabular or graphical form.

OUTSUM=*CAS-libref.data-table*

names the output data table to contain the summary statistics and the forecast summation. The summary statistics are based on the accumulated time series when the ACCUMULATE= or SETMISSING= option is specified in the ID statement. The OUTSUM= data table is useful for summarizing results when large numbers of series are forecast.

BY Statement

BY *variables* ;

A BY statement can be used with PROC CARIMA to obtain separate dummy variable definitions for groups of observations that are defined by the BY *variables*.

ESTIMATE Statement

ESTIMATE < *options* > ;

The ESTIMATE statement specifies the type of ARIMA model to be fit for the *variable* in the preceding IDENTIFY statement.

You can specify the following *options*:

CONVERGE=*value*

specifies the convergence criterion, where *value* is a number between 0 and 1, exclusive. Convergence is assumed when the largest change in the estimate for any parameter is less than the specified *value*. If the absolute value of the parameter estimate is greater than 0.01, the relative change is used; otherwise, the absolute change in the estimate is used. By default, CONVERGE=0.001.

DELTA=*value*

specifies the perturbation value for computing numerical derivatives, where *value* is a number between 0 and 1, exclusive. By default, DELTA=0.001.

DIFF=(*value-list*)

specifies the degrees of differencing for the *variable-list* that is specified in the preceding IDENTIFY statement. For example, you can specify DIFF=(1) to take a first-order difference for all variables specified in the *variable-list* of the preceding IDENTIFY statement. If you have monthly data and expect seasonal patterns, you can specify DIFF=(1,12) to difference all variables in the *variable-list* of the preceding IDENTIFY statement twice: once at lag 1, and then again at lag 12. Each analyzed series is then the difference between the current period-to-period change in the specified variable and the change 12 periods ago. Each value in the *value-list* is an integer greater than 0. By default, DIFF=(0).

MAXITER=*value*

specifies the maximum number of iterations, where *value* is a positive integer. By default, MAXITER=50.

METHOD=*method-name*

specifies the estimation method to be used.

You can specify the following forecasting *method-names*:

- CLS** specifies the conditional least squares method.
- ML** specifies the maximum likelihood method.
- ULS** specifies the unconditional least squares method.

By default, METHOD=CLS.

MU=value

specifies a constant term for the ARIMA model, where *value* is a number. By default, MU=0.

NOINT=0 | 1

specifies whether to include an intercept in the model. You can specify the following values:

- 0** includes an intercept term in the model.
- 1** omits the intercept term from the model. This suppresses the fitting of a constant (or intercept) parameter in the model. That is, the value specified in the MU value is omitted.

By default, NOINT=0.

NOSTABLE=0 | 1

specifies whether to restrict the candidate estimates for the autoregressive and moving average parameters in the stationary and invertible regions, respectively. You can specify the following values:

- 0** restrict candidate estimates in stable and invertible regions.
- 1** do not restrict candidate estimates.

By default, NOSTABLE=0.

P=value

specifies the degree of the autoregressive polynomial, where *value* is an integer greater or equal than 0. By default, P=0.

Q=value

specifies the degree of the moving average polynomial, where *value* is an integer greater or equal than 0. By default, Q=0.

SINGULAR=value

specifies the criterion for checking singularity, where *value* is a number between 0 and 1, exclusive. If a pivot of a sweep operation is less than *value*, the matrix is deemed singular. Sweep operations are performed on the Jacobian matrix during final estimation and on the covariance matrix when preliminary estimates are obtained. The default is 1E-7.

TRANSFORM=transform-option

specifies the time series transformation to apply to the series in the ESTIMATE statement. You can specify the following *transform-options*:

- BOXCOX(*n*)** applies a Box-Cox transformation with parameter number, *n*, where *n* is a number between -5 and 5.
- LOG** applies a logarithmic transformation.
- LOGISTIC** applies a logistic transformation.
- NONE** does not apply a transformation.
- SQRT** applies a square-root transformation.

By default, TRANSFORM=NONE.

FORECAST Statement

FORECAST < *options* > ;

The FORECAST statement forecasts the future values of the variable list that is specified in the preceding IDENTIFY statement according to the model that is fitted in the preceding ESTIMATE STATEMENT.

You can specify any number of FORECAST statements. You can specify the following *options*:

ALPHA=*number*

specifies the significance level to use in computing the confidence limits of the forecast, where *number* must be between 0 and 1. By default, ALPHA=0.05, which produces 95% confidence intervals.

LEAD=*number*

specifies the *number* of periods ahead to forecast, where *number* must be an integer greater than or equal to 0. By default, LEAD=0.

ID Statement

ID *variable* **INTERVAL**=*interval* < *options* > ;

The ID statement names a numeric *variable* that identifies observations in the input and output data tables. The ID variable's values are assumed to be SAS date or datetime values. In addition, the ID statement specifies the (desired) frequency that is associated with the time series. The ID statement *options* also specify how to accumulate the observations and how to align the time ID values to form the time series to be forecast. The specified information affects all variables that are specified in subsequent IDENTIFY statements. If an ID statement is not specified, the observation number (with respect to the BY group) is used as the time ID.

You must specify the **INTERVAL**=*interval* option:

INTERVAL=*interval*

specifies the frequency of the input time series or the frequency for the time series to be accumulated from the input data. For example, if the input data consist of quarterly observations, then specify INTERVAL=QTR. For example, INTERVAL=QTR implies a seasonal cycle of length 4. If the ACCUMULATE= option is also specified, the INTERVAL= option determines the time periods for the accumulation of observations.

The basic intervals are YEAR, SEMIYEAR, QTR, MONTH, SEMIMONTH, TENDAY, WEEK, WEEKDAY, DAY, HOUR, MINUTE, SECOND. For more information about the intervals that can be specified, see the “Date Intervals, Formats, and Functions” chapter in *SAS/ETS User's Guide*.

You can also specify the following *options*:

ACCUMULATE=*accumulate-option*

specifies how to accumulate the observations within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID variable contains the time ID values. Each time ID variable value corresponds to a specific time period. The accumulated values form the time series, which is used in subsequent model fitting and forecasting.

This option is particularly useful when there are gaps in the input data or when there are multiple input observations that coincide with a particular time period (for example, transactional data). The EXPAND

procedure in *SAS/ETS User's Guide* offers additional frequency conversions and transformations that can also be useful in creating a time series.

The following *accumulate-options* determine how the observations are accumulated within each time period on the basis of the ID variable and the frequency that is specified in the INTERVAL= option:

TOTAL SUM	accumulates observations based on the total sum of their values.
AVERAGE AVG	accumulates observations based on the average of their values.
MINIMUM MIN	accumulates observations based on the minimum of their values.
MAXIMUM MAX	accumulates observations based on the maximum of their values.
N	accumulates observations based on the number of nonmissing observations.
NMISS	accumulates observations based on the number of missing observations.
STDDEV STD	accumulates observations based on the standard deviation of their values.
CSS	accumulates observations based on the corrected sum of squares of their values.
USS	accumulates observations based on the uncorrected sum of squares of their values.

By default, ACCUMULATE=TOTAL.

If the ACCUMULATE= option is specified, the SETMISSING= option is useful for specifying how to treat accumulated missing values. If missing values should be interpreted as 0, then specify SETMISSING=0. For more information about accumulation, see the section “[Accumulation](#)” on page 83.

ALIGN=*align-option*

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, the ALIGN= option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations you supply, the ALIGN= option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often you do not supply. You can specify the following *align-options*:

BEGINNING BEG B	represents each time period by using the beginning SAS date or datetime value of the time period.
ENDING END E	represents each time period by using the ending SAS date or datetime value of the time period.
MIDDLE MID M	represents each time period by using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

END=*date | datetime*

specifies a SAS date or datetime literal value that represents the end of the data. If the last time ID *variable* value is less than the END= value, the series is extended with missing values. If the last time ID *variable* value is greater than the END= value, the series is truncated. For example, **END= '1jan2008'**D requests that data for time periods after the first of January 2008 not be used. The option **END= "&sysdate"**D uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. This option and the START= option can be used to ensure that data associated with each BY group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If you omit this option, the default format is implied by the INTERVAL= option.

SETMISSING=*option*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can specify the following values to determine how to interpret missing values:

<i>n</i>	interprets a missing value as having the value <i>n</i> . You can specify any number for <i>n</i> , but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because the absence of recorded data usually implies no activity.
MISSING	interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
AVERAGE AVG	interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
MINIMUM MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*date* | *datetime*

specifies a SAS date or datetime literal value that represents the beginning of the data. If the first time ID variable value is greater than the START= value, the series is prefixed with missing values. If the first time ID variable value is less than the START= value, the series is truncated. This option and the END= option can be used to ensure that data associated with each BY group contain the same number of observations.

TRIMID=*method*

specifies the method for trimming the data in the BY groups. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable is recalculated according to the value of the LEAD= option (if one is specified).

You can specify one of the following *methods*:

- NONE** uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
- LEFT** uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.
- RIGHT** uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.
- BOTH** uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

IDENTIFY Statement

IDENTIFY *variable-list* ;

The identify statement specifies the variables you want to model. Specify one or more variables in a space-separated *variable-list*.

Details: CARIMA Procedure

The CARIMA procedure can be used to fit ARIMA models either to time series data or to as transactional data. If the data are transactional, then the procedure must first accumulate the data into a time series before it can forecast them. The procedure uses the sequential steps in [Table 4.2](#) to produce forecasts, with the options that control the step listed to the right.

Table 4.2 PROC CARIMA Processing Steps and Control Options

Step	Operation	Options	Statement
1	Accumulation	ACCUMULATE=	ID
2	Missing value interpretation	SETMISSING=	ID
3	Parameter estimation	P=, Q=	ESTIMATE
4	Inverse transformation	TRANSFORM, DIFF	ESTIMATE
5	Forecasting	LEAD=	FORECAST

Each of the steps shown in [Table 4.2](#) is described in the following sections.

Accumulation

If the ACCUMULATE= option is specified in the ID statement, data table observations are accumulated within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option, and the ID variable contains the time ID values. Each time ID value corresponds to a specific time period. Accumulation is particularly useful when the input data table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series that is used in subsequent analyses by the CARIMA procedure.

For example, suppose a data table contains the following observations:

```
19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
23MAY1999    20
```

If the INTERVAL=MONTH option is specified in the ID statement, all the preceding observations fall within three time periods: March 1999, April 1999, and May 1999. The observations are accumulated within each time period as follows:

- If the ACCUMULATE=NONE option is specified, an error is generated because the ID variable values are not equally spaced with respect to the specified frequency (MONTH).
- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

```
O1MAR1999    40
O1APR1999    .
O1MAY1999    90
```

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    30
```

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MEDIAN option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=FIRST option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=LAST option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

O1MAR1999	14.14
O1APR1999	.
O1MAY1999	17.32

As you can see from the preceding examples, even though the data table observations contained no missing values, the accumulated time series can have missing values.

Missing Value Interpretation

After the data have been accumulated to form a time series on the basis of the `INTERVAL=` and the `ACCUMULATE=` options, missing values are interpreted. Sometimes missing values should be interpreted as unknown values. In this case, you can specify the `SETMISSING=` option to interpret how to treat missing values. In other situations, missing values are known, such as when missing values are created from accumulation and the absence of an observation should be interpreted the observation having the value 0. In this case, you can specify `SETMISSING=0` in order to treat missing observations as 0 values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Parameter Estimation

The default method for estimating parameters is conditional least squares. You can also use maximum likelihood estimation and unconditional least squares, by specifying the corresponding value in the `METHOD=` option in the `ESTIMATE` statement.

The techniques used in the CARIMA procedure are identical to those used for ARIMA models in the Time Series Forecasting System of SAS/ETS software. For more information, see the “Overview of the Time Series Forecasting System” chapter in *SAS/ETS User’s Guide*.

Transformations

If the `TRANSFORM=` option is specified in the `ESTIMATE` statement, the time series is transformed before the parameters are estimated and the series is forecast. Only strictly positive series can be transformed. An error is generated when the `TRANSFORM=` option is used with a nonpositive series. For more information about forecasting transformed time series, see the “Forecasting Process Details” chapter in *SAS/ETS User’s Guide*.

Forecasting

After the model parameters are estimated, one-step-ahead forecasts are generated for the full range of the accumulated and optionally transformed time series data, and multistep forecasts are generated from the end of the time series to the future time period as specified in the LEAD= option. If there are missing values at the end of the time series, the forecast horizon will be greater than that specified by the LEAD= option.

Data Table Output

The CARIMA procedure can create the OUT=, OUTEST=, OUTFOR=, and OUTSUM= data tables. These data tables contain the variables that are listed in the BY, ID, and IDENTIFY statements, in addition to statistics that are related to the parameters estimated by the ESTIMATE statement. In general, if a forecasting step that is related to an output data table fails, the values of that step are not recorded or are set to missing in the related output data table and appropriate error or warning messages (or both) are recorded in the log. For more information about how the variables in the output data tables are computed, see the section “Smoothing Models” in the chapter “Forecasting Process Details” in *SAS/ETS User’s Guide*.

OUT= Data Table

The OUT= data table contains the variables that are specified in the BY, ID, and IDENTIFY statements. If the ID statement is specified, its variable values are aligned and extended according to the values of ALIGN= and INTERVAL= options. The values of the variables that are specified in the IDENTIFY statements are accumulated according to the values of the ACCUMULATE= option, and missing values are interpreted according to the values of the SETMISSING= option.

If any forecasting steps fail for a particular variable, the variable is extended by missing values.

OUTEST= Data Table

The OUTEST= data table contains the variables that are specified in the BY and IDENTIFY statements and the variables in the following list.

NAME	Variable name
MODEL	Forecasting model
MODELVAR	Model variable
DSVAR	Data table variable
VARTYPE	Variable type
TRANSFORM	Transformation
COMPONENT	Component
COMPMODEL	Component model
PARM	Parameter name
FACTOR	Factor
LAG	Lag

<code>_SHIFT_</code>	Shift
<code>_EST_</code>	Value of parameter estimate
<code>_STDERR_</code>	Standard error of parameter estimate
<code>_TVALUE_</code>	t values of parameter estimate
<code>_STDERR_</code>	p -values of parameter estimate

If the parameter estimation step fails for a particular variable, no observations are output to the OUTEST= data table for that variable.

OUTFOR= Data Table

The OUTFOR= data table contains the variables that are specified in the BY and IDENTIFY statements and the variables in the following list.

<code>_NAME_</code>	Variable name
ID Variable	Values of the variable specified in the ID statement
ACTUAL	Actual values
PREDICT	Predicted values
ERROR	Prediction errors
STD	Prediction standard errors
UPPER	Upper confidence limits
LOWER	Lower confidence limits

If the forecasting step fails for a particular variable, no observations are recorded in the OUTFOR= data table for that variable. If the TRANSFORM= option is specified, the values in the preceding IDENTIFY statement are the inverse transform forecasts. If the MEDIAN option is specified, the median forecasts are stored; otherwise, the mean forecasts are stored.

OUTSUM= Data Table

The OUTSUM= data table contains the variables that are specified in the BY statement and the variables in the following list. The OUTSUM= data table records the summary statistics for each variable that is specified in an IDENTIFY statement.

The following variables that are related to summary statistics are based on the ACCUMULATE= and SETMISSING= options:

<code>_NAME_</code>	Variable name
NOBS	Number of observations
N	Number of nonmissing observations
NMISS	Number of missing observations
MIN	Minimum value
MAX	Maximum value

MEAN	Mean value
STDDEV	Standard deviation
STATUS	forecasting status (nonzero values imply that no forecast was generated for the series)
START	Starting date of each series
END	Ending date of each series
STARTOBS	Beginning observation number of each series
ENDOBS	Ending observation number of each series

Printed Output

The CARIMA procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC CARIMA.

The CARIMA procedure also prints by default the parameter estimates of fitted models in the ParameterEstimates table and the multistep-ahead variable forecasts in the Forecasts table. The name of the modeled variable is printed in the VariableInfo table.

You can also print parameter estimates, variable forecasts, and other summary statistics by using targeted data queries to subset and display the information from the tables that PROC CARIMA produces. For example, you might want to print results in the OUTFOR= data table only for BY groups that have a particular _STATUS_ value in the OUTSUM= table.

ODS Table Names

The CARIMA procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 4.3](#).

Table 4.3 ODS Tables Produced in the CARIMA Procedure

ODS Table Name	Description	Option
Forecasts	Forecast components, such as prediction, prediction standard error, lower and upper confidence limits	Default
OutInfo	Summary of the processing that is performed on the time series data	Default
ParameterEstimates	Parameter estimates	Default
VariableInfo	Name of the variable specified in the IDENTIFY statement	Default

Example: CARIMA Procedure

Example 4.1: Seasonal Model for the Airline Series

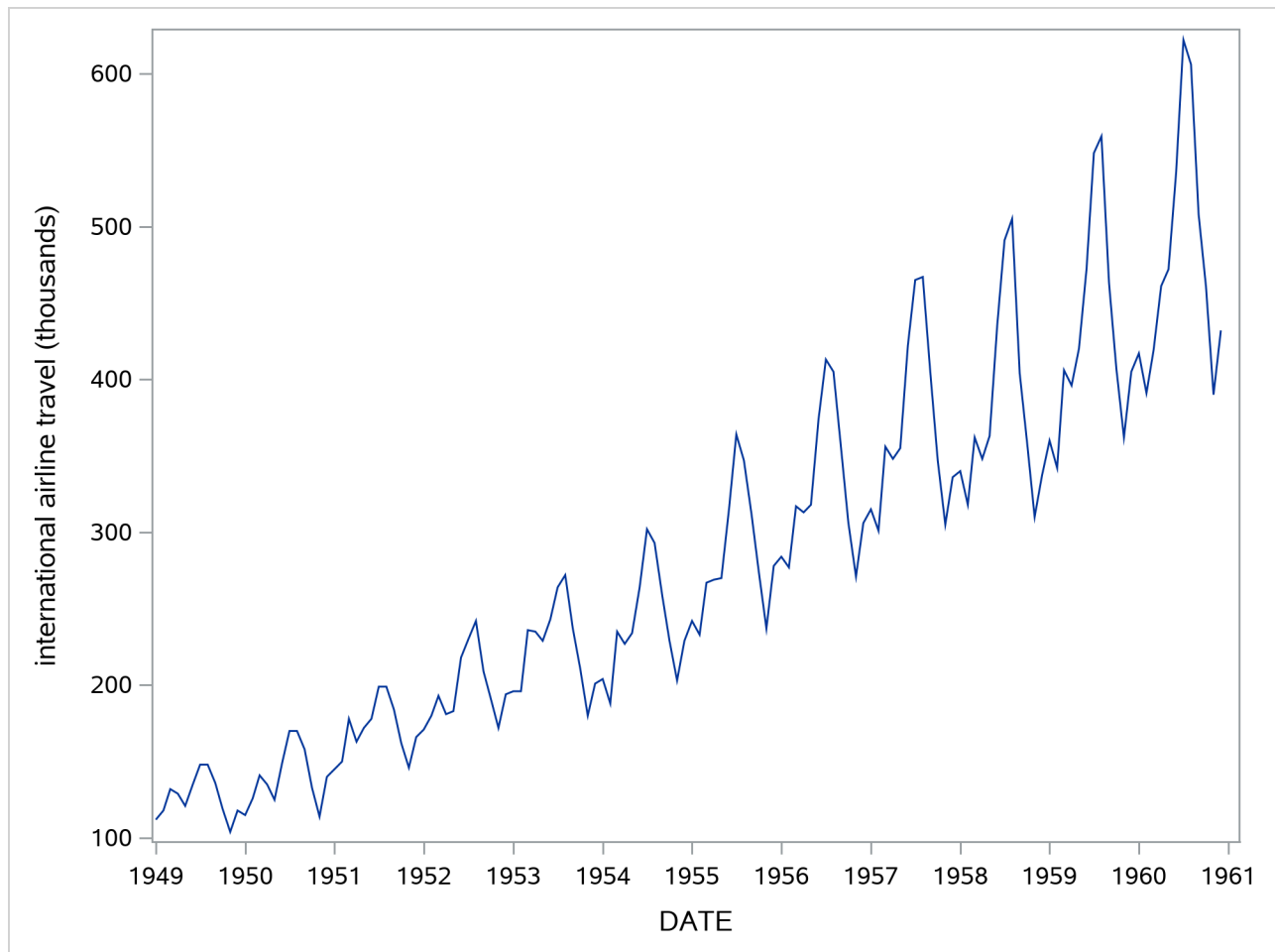
The airline passenger data have been used in time series analysis literature as an example of a nonstationary seasonal time series.

The following statements plot the air data:

```
proc sgplot data=sashelp.air;  
  series x = date y = air;  
run;
```

Output 4.1.1 indicates that the series is nonstationary and seasonal.

Output 4.1.1 Airline Passenger Data



The following DATA step loads the air data into a CAS session. The DATA step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately named CAS engine libref.

```
data mycas.myair;
  set sashelp.air;
run;
```

You can use the following statements to fit an $ARIMA(0,1,1) \times (0,1,1)_{12}$ model without a mean term to the logarithms of the airline passengers series. The model estimates and forecasts are stored in the data tables `mycas.cariest` and `mycas.carifor`.

```
proc carima data=mycas.myair outest=mycas.cariest outfor=mycas.carifor;
  id date interval=month;
  identify air;
  estimate q=(1)(12) noint method = ML transform = log diff = (1,12);
  forecast lead = 4;
run;
```

Output 4.1.2 shows the summary of the time series processing.

Output 4.1.2 Summary of Time Series Processing
The CARIMA Procedure

Summary of time series processing for MYAIR	
Number of analysis variables	1
Number of rows read	288
Number of groups read	1
Memory for group packages (KB)	15
Time to load groups (seconds)	0.0100331306
Minimum time ID	JAN1949
Maximum time ID	DEC1960
Minimum time periods	144
Maximum time periods	144
Number of nodes run	4
Number of nodes with data	3
Number of nodes with groups	1
Number of threads budgeted	1
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	1
Maximum threads active	1
Number of groups processed by submitted code	1
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0490541458

Output 4.1.3 shows the name of the modeled variable.

Output 4.1.3 Variable Information

Variable Info	
Variable Name	AIR

Output 4.1.4 shows the parameter estimates from the fitted model.

Output 4.1.4 Parameter Estimates

Maximum Likelihood Estimates					
Parameter	Estimate	Standard Error	t Value	Approx Pr > t	Lag
MA1_1	0.401936	0.079877	5.03	<.0001	1
MA2_12	0.556862	0.084027	6.63	<.0001	12

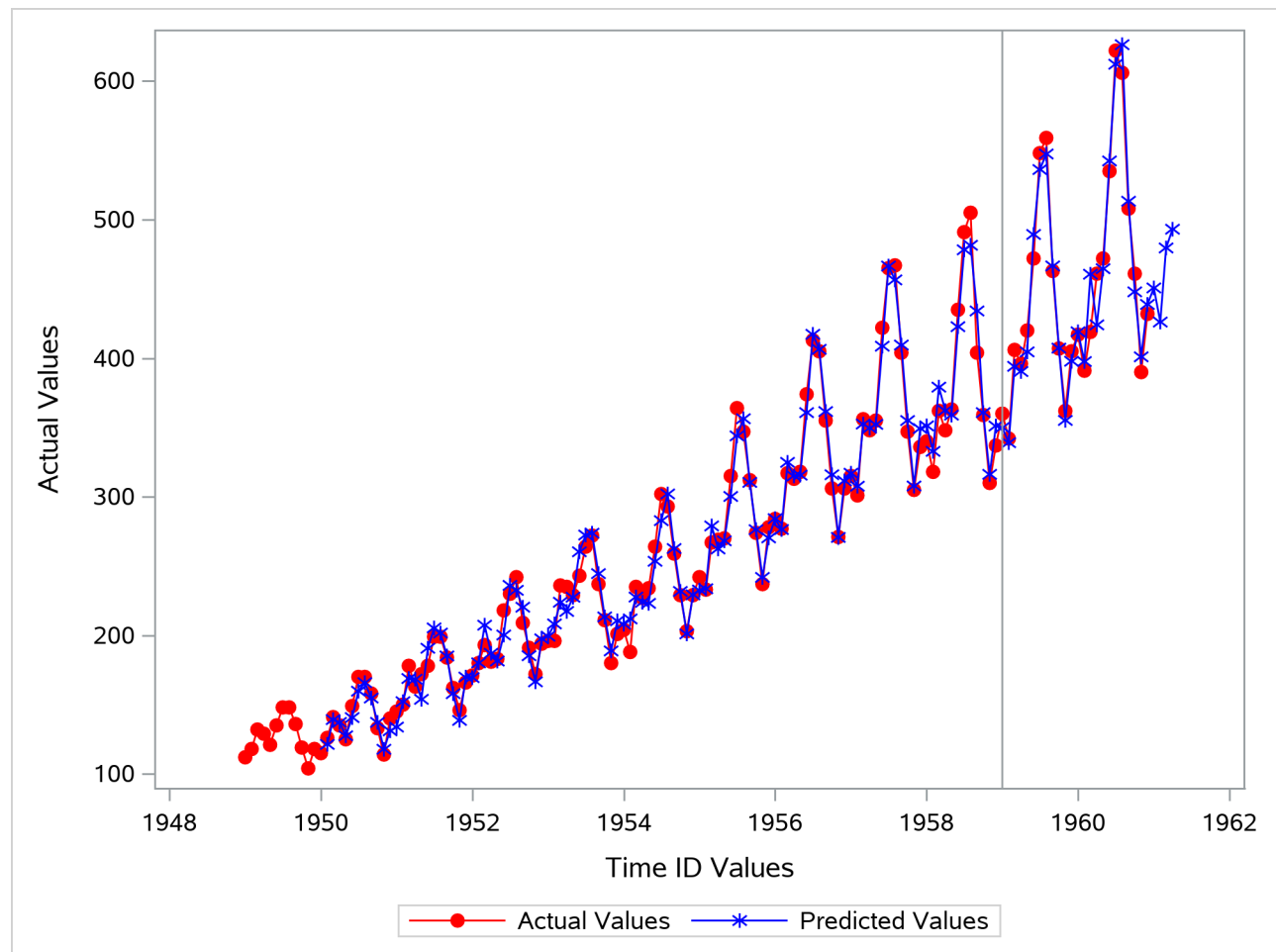
Output 4.1.5 shows the multistep-ahead variable forecasts.

Output 4.1.5 Variable Forecasts

Forecasts for AIR using Model 1				
Time ID	Forecast	Std Error	95% Confidence Limits	
JAN1961	450.7320	16.6829	418.9155	484.3014
FEB1961	426.1129	18.3793	391.2224	463.2533
MAR1961	479.5664	23.2533	435.6018	526.7297
APR1961	493.1052	26.2879	443.5853	546.5997

The following statements plot the forecasts that are produced by the $ARIMA(0,1,1) \times (0,1,1)_{12}$ model. The plot is shown in Output 4.1.6.

```
proc sgplot data=mycas.carifor;
  series x=date y=actual / markers lineattrs=(color=red)
                                markerattrs=(symbol=circlefilled color=red);
  series x=date y=predict / markers lineattrs=(color=blue)
                                markerattrs=(symbol=asterisk color=blue);
  refline '01JAN1959'd / axis=x;
run;
```

Output 4.1.6 Airline Passengers Forecasts

References

Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.

Chapter 5

The CCDM Procedure

Contents

Overview: CCDM Procedure	94
Using CAS Sessions and CAS Engine Librefs	96
Getting Started: CCDM Procedure	97
Estimating a Simple Compound Distribution Model	97
Analyzing the Effect of Parameter Estimate Uncertainty on the Compound Distribution	102
Scenario Analysis	105
Syntax: CCDM Procedure	115
Functional Summary	116
PROC CCDM Statement	118
BY Statement	128
COUNTMODEL Statement	128
DISPLAY Statement	129
DISPLAYOUT Statement	130
EXTERNALCOUNTS Statement	131
OUTPUT Statement	132
OUTSUM Statement	133
SEVERITYMODEL Statement	136
SIMULATEDSYMBOL Statement	137
Programming Statements	138
Details: CCDM Procedure	138
Specifying Scenario Data in the DATA= Data Table	138
Simulation Processes for Collective Risk Simulation Mode	139
Simulation Processes for Pure Premium and Custom Simulation Modes	147
Simulation of Adjusted Compound Distribution Sample	151
Parameter Perturbation Analysis	161
Fixed Parameter-Value Distributions	162
Descriptive Statistics	166
Input Specification	168
Output Data Tables	169
Displayed Output	172
ODS Graphics	173
Examples: CCDM Procedure	174
Example 5.1: Estimating the Probability Distribution of Insurance Payments	174
Example 5.2: Using Externally Simulated Count Data	179
Example 5.3: Scenario Analysis with Rich Regression Effects and BY Groups	189
Example 5.4: Illustrating Pure Premium and Custom Simulation Modes	200
References	206

Overview: CCDM Procedure

In many loss modeling applications, you analyze the loss events by modeling the severity (magnitude) of loss and the frequency (count) of loss separately. The primary goal of creating these models is to estimate the aggregate loss—that is, the total loss that occurs over a period of time for which the frequency model is applicable. For example, an insurance company might want to assess the expected and worst-case losses for a particular business line, such as automobile insurance, over an entire year by using the models for the number of losses in a year and the severity of each loss. A bank might want to assess the value-at-risk (VaR), a measure of the worst-case loss, for a portfolio of assets by using the frequency and severity models for each asset type.

Severity of loss and frequency of loss are random variables, so the aggregate loss is also a random variable. Instead of preparing a point estimate of the expected aggregate loss, it is better to estimate its probability distribution, because this enables you to infer various aspects of the aggregate loss, such as measures of location, scale (variability), and shape, in addition to percentiles. For example, the VaR that banks or insurance companies use to compute regulatory capital requirements is usually the estimate of the 97.5th or 99th percentile from the aggregate loss distribution.

Let N represent the frequency random variable for the number of loss events that occur in the time period of interest. Let X represent the severity random variable for the magnitude of one loss event. Then, the aggregate loss S is defined as

$$S = \sum_{j=1}^N X_j$$

The goal is to estimate the probability distribution of S . Let $F_X(x)$ denote the cumulative distribution function (CDF) of X , let $F_X^{*n}(x)$ denote the n -fold convolution of the CDF of X , and let $\Pr(N = n)$ denote the probability of seeing n losses as per the frequency distribution. The CDF of S is theoretically computable as

$$F_S(s) = \sum_{n=0}^{\infty} \Pr(N = n) \cdot F_X^{*n}(x)$$

This probability distribution model of S , characterized by the CDF $F_S(s)$, is referred to as a *compound distribution model* (CDM). The CCDM procedure computes an estimate of the CDM by using the distribution models of X and N .

Direct computation of F_S is usually a difficult task because of the need to compute the n -fold convolution. Klugman, Panjer, and Willmot (1998, Ch. 4) suggest some relatively efficient recursion and inversion methods for certain combinations of severity and frequency distributions. However, those methods assume that distributions of N and X are fixed and all X s are identically distributed. When the distributions of X and N are conditional on external factors (regressors), each set of regressor values results in a different distribution. So you must repeat the recursion and inversion methods for each combination of regressor values, and this repetition makes these methods prohibitively expensive. PROC CCDM instead estimates the compound distribution by using a Monte Carlo simulation method, which can use all available computational resources to generate a sufficiently large, representative sample of the compound distribution while accommodating the dependence of distributions of X and N on external factors. Conceptually, the simulation method works as follows:

1. Use the specified frequency model to draw a value N , which represents the number of loss events.

2. Use the specified severity model to draw N values, each of which represents the magnitude of loss for each of the N loss events.
3. Add the N severity values from step 2 to compute the aggregate loss S as

$$S = \sum_{j=1}^N X_j$$

This forms one sample point of the CDM.

Steps 1 through 3 are repeated M times, where M is a number that you specify, to obtain the representative sample of the CDM. PROC CCDM analyzes this sample to compute empirical estimates of various summary statistics of the compound distribution such as the mean, variance, skewness, and kurtosis in addition to percentiles such as the median, the 95th percentile, the 99th percentile, and so on. You can also use PROC CCDM to write the entire simulated sample to an output data table.

The simulation process gets more complicated when the frequency and severity models contain regression effects. The CDM is then conditional on the values of regressors. The simulation process essentially becomes a scenario analysis, because you need to specify the expected values of the regressors that together represent the scenario for which you want to estimate the CDM. PROC CCDM enables you to specify an input data table that contains the scenario. If you are modeling a group of entities together (such as a portfolio of multiple assets or a group of insurance policies), and each entity has a different set of characteristics, then the scenario consists of more than one observation, and each observation corresponds to a different entity. PROC CCDM enables you to specify such a group scenario in the input data table and creates a realistic simulation of loss events that each entity can generate.

PROC CCDM accepts the severity model of X as estimated by the SEVSELECT procedure, and it accepts the frequency model of N as estimated by the CNTSELECT procedure. Both the SEVSELECT and CNTSELECT procedures are part of SAS Econometrics software. Both procedures allow models of X and N to be conditional on external factors (regressors). In particular, you can model the severity distribution such that its scale parameter depends on severity regressors, and you can model the frequency distribution such that its mean depends on frequency regressors. The frequency model can also be a zero-inflated model. PROC CCDM uses the estimates of model parameters and the values of severity and frequency regressors to estimate the compound distribution model.

PROC CCDM also enables you to specify externally simulated counts. This is useful if you have an empirical frequency model or if you estimate the frequency model by using a method other than using PROC CNTSELECT. To specify an external count model, you create a data table that contains M replications of counts that are randomly drawn from your frequency model. For each of the replications, in step 1 of the simulation, PROC CCDM uses the count N that is associated with that replication in the count data table. If the severity model contains regression effects, then you can specify the scenario to analyze by specifying the severity regressor data for each of the M replications in the same count data table.

If the estimates of the parameters of your severity and frequency models have uncertainty associated with them, and they usually do, then you can use PROC CCDM to conduct parameter perturbation analysis to assess the effect of the uncertainty in parameter estimates on the estimates of CDM. If you request that P perturbed samples be generated, then the parameter set is perturbed P times, and each time PROC CCDM makes a random draw from either the univariate normal distribution of each parameter or the multivariate normal distribution over all parameters. For each of the P perturbed parameter sets, a full compound distribution sample is simulated and summarized. This process yields P estimates for each summary statistic

and percentile, which are then used to provide you with estimates of the location and variability of each summary statistic and percentile.

You can also use PROC CCDM to compute the distribution of an aggregate *adjusted* loss. For example, in insurance applications, you might want to compute the distribution of the amount paid in a particular time period after applying adjustments such as the deductible and the policy limit to each individual loss. PROC CCDM enables you to specify SAS programming statements to adjust each severity value. If X_j^d represents the d th adjusted severity value, then PROC CCDM computes S^d , the d th aggregate adjusted loss, as

$$S^d = \sum_{j=1}^N X_j^d$$

You can define and analyze as many adjusted losses as you want in one PROC CCDM step. All the analyses that PROC CCDM conducts for the aggregate unadjusted loss, including scenario analysis and parameter perturbation analysis, are also conducted for each aggregate adjusted loss. This gives you a comprehensive picture of the compound distribution model for each adjusted loss.

PROC CCDM is the next-generation version of PROC HPCDM. It requires SAS Cloud Analytic Services (CAS) in order to run. Because PROC CCDM is a next-generation high-performance analytical procedure, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the

CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CCDM Procedure

This section outlines the use of the CCDM procedure to fit compound distribution models. The examples are intended as a gentle introduction to some of the features of the procedure.

Estimating a Simple Compound Distribution Model

This example illustrates the simplest use of PROC CCDM. Assume that you are an insurance company that has used its historical data about the number of losses per year and the severity of each loss to determine that the Poisson distribution is the best distribution for the loss frequency and that the gamma distribution is the best distribution for the severity of each loss. Now, you want to estimate the distribution of an aggregate loss to determine the worst-case loss that your policyholders can incur in a year. In other words, you want to estimate the compound distribution of $S = \sum_{i=1}^N X_i$, where the loss frequency, N , follows the fitted Poisson distribution and the severity of each loss event, X_i , follows the fitted gamma distribution.

To illustrate, let the historical count and severity data be stored in the data sets Work.ClaimCount and Work.ClaimSev, respectively. The following two DATA steps simulate a frequency sample from a Poisson distribution and a severity sample from the gamma distribution, respectively:

```
/* Simulate data for an intercept-only Poisson count model */
data claimcount(keep=numLosses);
  call streaminit(12345);
  label numLosses='Number of Loss Events in a Year';
  Lambda = 2;
  do n = 1 to 500;
    numLosses = rand('POISSON', Lambda);
    output;
  end;
run;

/* Simulate data for a gamma severity model */
data claimsev(keep=lossValue);
  call streaminit(67890);
  label lossValue='Severity of a Loss Event';
  Theta = 1000;
  Alpha = 2;
  do n = 1 to 500;
```

```

        lossValue = quantile('Gamma', rand('UNIFORM'), Alpha, Theta);
        output;
    end;
run;

```

You can load the data sets `Work.ClaimCount` and `Work.ClaimSev` into data tables in your CAS session by using your CAS engine libref with the following DATA steps:

```

/* Load data into the CAS server */
data mycas.claimcount;
    set claimcount;
run;
data mycas.claimsev;
    set claimsev;
run;

```

These statements assume that your CAS engine libref is named `mycas`, as in the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 96, but you can substitute any appropriately defined CAS engine libref.

To create model parameter estimates in a format that PROC CCDM can use, you need to use the following PROC CNTSELECT and PROC SEVSELECT steps to fit and store the parameter estimates of the frequency and severity models:

```

/* Fit an intercept-only Poisson count model and
   write estimates to an item store */
proc cntselect data=mycas.claimcount store=mycas.countStorePoisson;
    model numLosses= / dist=poisson;
run;

/* Fit severity models and write estimates to a data table */
proc sevselect data=mycas.claimsev criterion=aicc
    outest=mycas.sevest covout;
    loss lossValue;
    dist _predefined_;
run;

```

The `STORE=` option in the PROC CNTSELECT statement saves the count model information, including the parameter estimates, in the item store `mycas.CountStorePoisson`. An item store contains the model information in a binary format that cannot be modified after it is created. You can examine the contents of an item store that is created by a PROC CNTSELECT step by using the `VIEWSTORE` statement in another PROC CNTSELECT step. (For more information, see Chapter 8, “[The CNTSELECT Procedure](#).”)

The `OUTEST=` option in the PROC SEVSELECT statement stores the estimates of all the fitted severity models in the data table `mycas.SevEst`. You can verify that the best severity model that the PROC SEVSELECT step chooses is the gamma distribution model.

You can now submit the following PROC CCDM step to simulate an aggregate loss sample of size 10,000 by specifying the count model’s item store in the `COUNTSTORE=` option and the severity model’s data table of estimates in the `SEVERITYEST=` option:

```

/* Simulate and estimate Poisson-gamma compound distribution model */
proc ccdm countstore=mycas.countStorePoisson severityest=mycas.sevest
    seed=13579 nreplicates=10000 plots=(edf density)
    print=(summarystatistics percentiles);
    severitymodel gamma;

```

```

output out=mycas.aggregateLossSample samplevar=aggloss;
outsum out=mycas.aggregateLossSummary mean stddev skewness kurtosis
      p01 p05 p95 p995=var pctlpts=90 97.5;
run;

```

The SEVERITYMODEL statement requests that an aggregate sample be generated by compounding only the gamma distribution and the frequency distribution. Specifying the SEED= value helps you get an identical sample each time you execute this step, provided that you use the same execution environment. In the single-machine mode of execution, the execution environment is the combination of the operating environment and the number of threads that are used for execution. In distributed computing mode, the execution environment is the combination of the operating environment, the number of worker nodes, and the number of threads that are used for execution on each worker node.

Upon completion, PROC CCDM creates the two output data tables that you specify in the OUT= options in the OUTPUT and OUTSUM statements. The data table mycas.AggregateLossSample contains 10,000 observations such that the value of the variable AggLoss in each observation represents one possible aggregate loss value that you can expect to see in one year. Together, the set of the 10,000 values of AggLoss represents one sample of compound distribution. PROC CCDM uses this sample to compute the empirical estimates of various summary statistics and percentiles of the compound distribution. The data table mycas.AggregateLossSummary contains the estimates of mean, standard deviation, skewness, and kurtosis that you specify in the OUTSUM statement. It also contains the estimates of the 1st, 5th, 90th, 95th, 97.5th, and 99.5th percentiles that you specify in the OUTSUM statement. The value-at-risk (VaR) is an aggregate loss value such that there is a very low probability that an observed aggregate loss value exceeds the VaR. One of the commonly used probability levels to define VaR is 0.005, which makes the 99.5th percentile an empirical estimate of VaR. Hence, the OUTSUM statement of this example stores the 99.5th percentile in a variable named VaR. VaR is one of the widely used measures of worst-case risk.

Some of the default output and some of the output that you have requested by specifying the PRINT= option are shown in [Figure 5.1](#).

Figure 5.1 Information, Summary Statistics, and Percentiles of the Poisson-Gamma Compound Distribution

The CCDM Procedure

Severity Model: Gamma
Count Model: Poisson

Compound Distribution Information	
Severity Model	Gamma Distribution
Count Model	Poisson Model in Item Store COUNTSTOREPOISSON

The CCDM Procedure

Severity Model: Gamma
Count Model: Poisson

Sample Summary Statistics			
Mean	4053.7	Median	3381.3
Standard Deviation	3462.3	Interquartile Range	4519.6
Variance	11987861.6	Minimum	0
Skewness	1.14689	Maximum	25106.0
Kurtosis	1.67854	Sample Size	10000

Figure 5.1 *continued*

Sample Percentiles	
Percentile	Value
1	0
5	0
25	1388.7
50	3381.3
75	5908.3
90	8856.8
95	10739.6
97.5	12540.0
99	14640.9
99.5	16324.1
Percentile Method = 5	

The “Sample Summary Statistics” table indicates that for the given parameter estimates of the Poisson frequency and gamma severity models, you can expect to see a mean aggregate loss of 4,053.7 and a median aggregate loss of 3,381.3 in a year. The “Sample Percentiles” table indicates that there is a 0.5% chance that the aggregate loss exceeds 16,324.1, which is the VaR estimate, and a 2.5% chance that the aggregate loss exceeds 12,540. These summary statistic and percentile estimates provide a quantitative picture of the compound distribution.

You can also visually analyze the compound distribution by examining the plots that PROC CCDM creates when you specify the PLOTS= option. The first plot in [Figure 5.2](#) shows the empirical distribution function (EDF), which is a nonparametric estimate of the cumulative distribution function (CDF). The second plot shows the histogram and the kernel density estimate, which are nonparametric estimates of the probability density function (PDF).

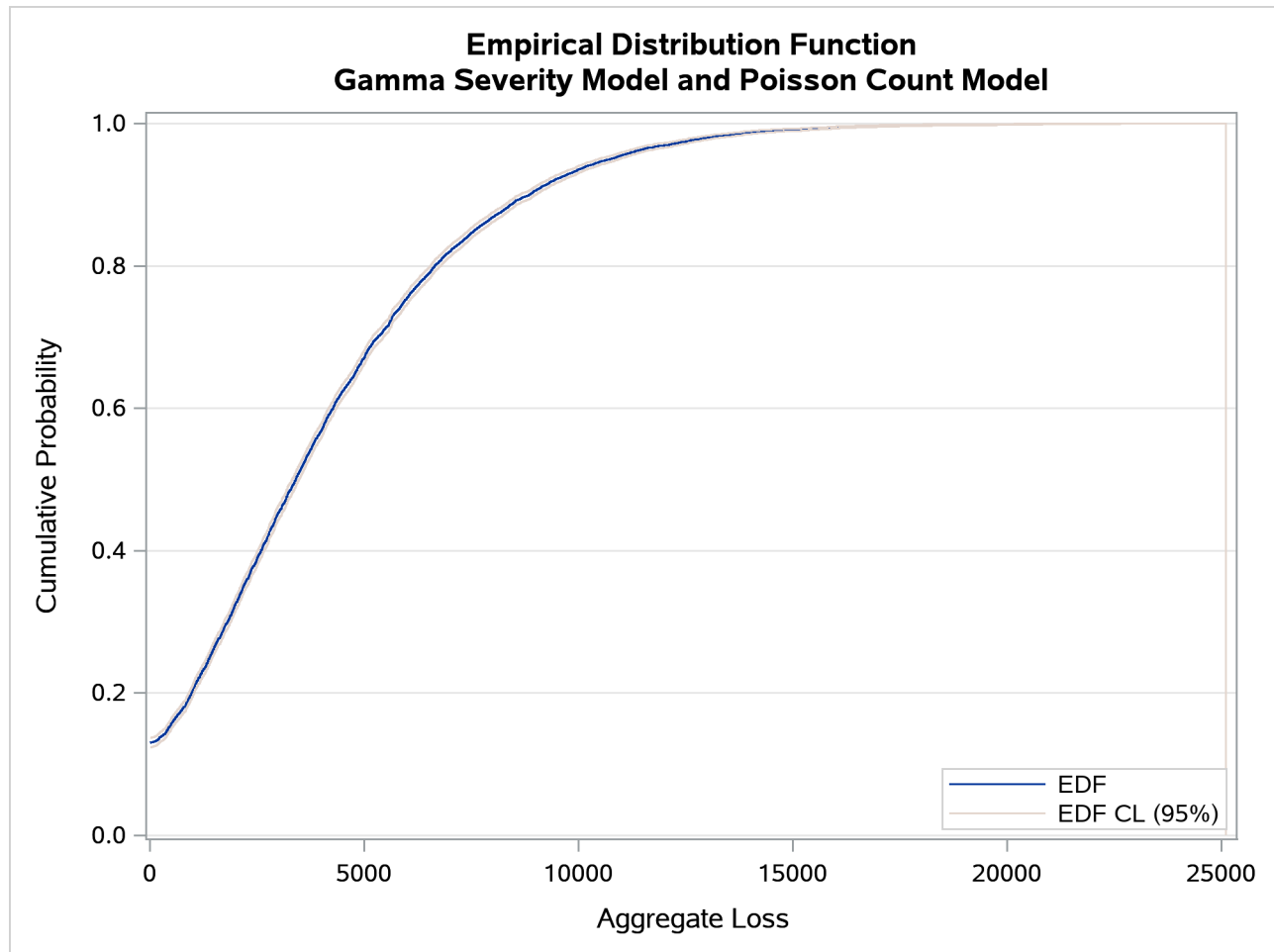
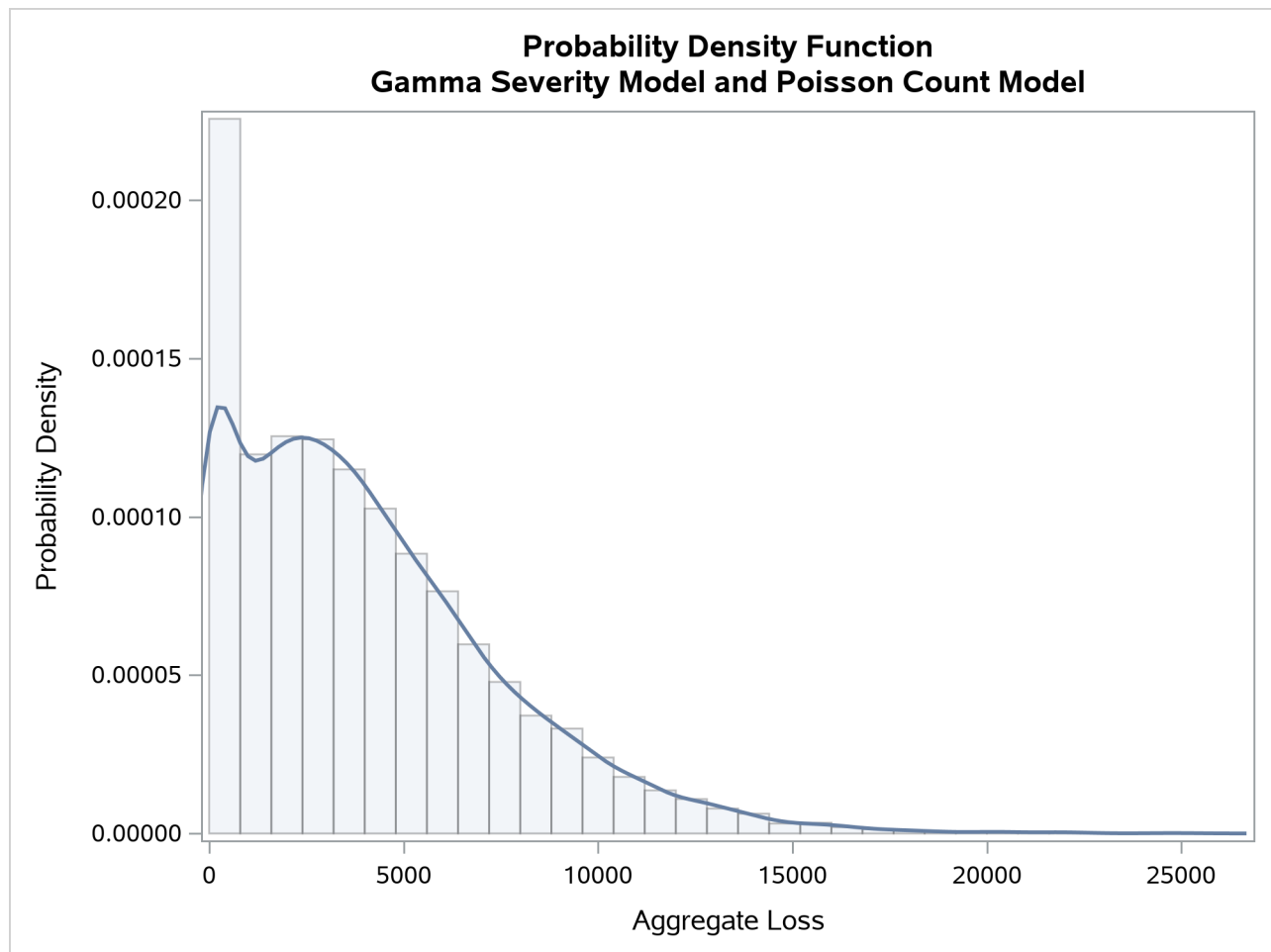
Figure 5.2 Nonparametric CDF and PDF Plots of the Poisson-Gamma Compound Distribution

Figure 5.2 *continued*

The plots confirm the right skew that is indicated by the estimate of skewness in Figure 5.1 and a relatively fat tail, which is indicated by comparing the maximum and the 99.5th percentiles in Figure 5.1.

Analyzing the Effect of Parameter Estimate Uncertainty on the Compound Distribution

Continuing with the previous example, note that you have fitted the frequency and severity models by using the historical data. Even if you choose the best-fitting models, the true underlying models are not known exactly. This fact is reflected in the uncertainty that is associated with the estimates of your model parameters. Any compound distribution estimate that you compute by using these uncertain parameter estimates is inherently uncertain. You can request that PROC CCDM perform parameter perturbation analysis; this analysis assesses the effect of the uncertainty in parameter estimates on the estimates of the compound distribution by simulating multiple samples, each of which uses parameters that are randomly perturbed from their mean estimates.

The following PROC CCDM step adds the NPerturbedSamples= option to the PROC CCDM statement to perform perturbation analysis and the PRINT=PERTURBSUMMARY option to display a summary of the perturbation analysis:

```
/* Perform parameter perturbation analysis of
   the Poisson-gamma compound distribution model */
proc ccdm countstore=mycas.countStorePoisson severityest=mycas.sevest
  seed=13579 nreplicates=10000 nperturbedsamples=30
  print (only)=(perturbsummary);
  severitymodel gamma;
  output out=mycas.aggregateLossSample samplevar=aggloss;
  outsum out=mycas.aggregateLossSummary mean stddev skewness kurtosis
    p01 p05 p95 p995=var pctlpts=90 97.5;
run;
```

The data table mycas.AggregateLossSummary contains the specified summary statistics and percentiles for all 30 perturbed samples. You can identify a perturbed sample by the value of the variable `_DRAWID_`. Figure 5.3 shows the first few observations of the data table mycas.AggregateLossSummary. For the first observation, the value of `_DRAWID_` is 0, which represents an unperturbed sample—that is, the aggregate sample that is simulated without perturbing the parameters from their means.

Figure 5.3 Summary Statistics and Percentiles of the Perturbed Samples

<code>_SEVERITYMODEL_</code>	<code>_COUNTMODEL_</code>	<code>_DRAWID_</code>	<code>_SAMPLEVAR_</code>	<code>N</code>	<code>MEAN</code>	<code>STDDEV</code>
Gamma	Poisson	0	aggloss	10000	4053.66	3462.35
Gamma	Poisson	1	aggloss	10000	4103.75	3424.54
Gamma	Poisson	2	aggloss	10000	4257.98	3595.69
Gamma	Poisson	3	aggloss	10000	4334.57	3598.11
Gamma	Poisson	4	aggloss	10000	3812.29	3258.42
Gamma	Poisson	5	aggloss	10000	3971.81	3399.79
Gamma	Poisson	6	aggloss	10000	4232.47	3566.42
Gamma	Poisson	7	aggloss	10000	4172.48	3516.92
Gamma	Poisson	8	aggloss	10000	3875.37	3326.50
Gamma	Poisson	9	aggloss	10000	4203.88	3518.37
Gamma	Poisson	10	aggloss	10000	3897.70	3400.52

<code>SKWNESS</code>	<code>KURTOSIS</code>	<code>P01</code>	<code>P05</code>	<code>P90</code>	<code>P95</code>	<code>P97_5</code>	<code>var</code>
1.14689	1.67854	0	0	8856.80	10739.64	12539.98	16324.10
1.05104	1.21420	0	0	8808.98	10584.92	12350.99	15762.00
1.13383	1.68019	0	0	9067.35	11087.18	13076.13	17018.82
1.07910	1.37329	0	0	9388.06	11262.39	12980.74	17026.75
1.12937	1.79096	0	0	8248.84	10001.45	11646.66	15176.26
1.19894	1.83912	0	0	8574.82	10576.42	12625.33	16327.78
1.16724	1.80881	0	0	9094.55	11021.82	12934.17	17096.81
1.17610	1.89699	0	0	8846.07	10927.64	12563.42	17200.04
1.15035	1.65788	0	0	8391.27	10211.29	11977.74	15988.06
1.11452	1.53488	0	0	9018.12	11008.27	12748.88	16458.14
1.20300	1.90903	0	0	8546.00	10419.82	12228.73	16171.06

The PRINT=PERTURBSUMMARY option in the preceding PROC CCDM step produces the “Sample Perturbation Analysis” and “Sample Percentile Perturbation Analysis” tables shown in Figure 5.4. The tables

show that you can expect a mean aggregate loss of about 4,059 and the standard error of the mean is 190.1. If you want to use the VaR estimate to determine the amount of reserves that you need to maintain to cover the worst-case loss, then you should consider not only the mean estimate of the 99.5th percentile, which is about 16,224.5, but also the standard error of 700.2 to account for the effect of uncertainty in your frequency and severity parameter estimates.

Figure 5.4 Summary of Perturbation Analysis of the Poisson-Gamma Compound Distribution

The CCDDM Procedure		
Severity Model: Gamma Count Model: Poisson		
Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	4059.0	190.13976
Standard Deviation	3439.6	124.94255
Variance	11846525	874364.8
Skewness	1.12920	0.05079
Kurtosis	1.58944	0.25511
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		
Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	0	0
5	0	0
25	1407.7	123.36686
50	3389.3	179.44672
75	5951.7	262.77659
90	8751.7	343.52016
95	10665.2	422.58551
97.5	12440.7	493.79317
99	14620.5	592.97468
99.5	16224.5	700.18732
Number of Perturbed Samples = 30		
Size of Each Sample = 10000		

Scenario Analysis

The distributions of loss frequency and loss severity often depend on exogenous variables (regressors). For example, the number of losses and the severity of each loss that an automobile insurance policyholder incurs might depend on the characteristics of both the policyholder and the vehicle. When you fit frequency and severity models, you need to account for the effects of such regressors on the probability distributions of the counts and severity. The CNTSELECT procedure enables you to model regression effects on the mean of the count distribution, and the SEVSELECT procedure enables you to model regression effects on the scale parameter of the severity distribution. When you use these models to estimate the compound distribution model of the aggregate loss, you need to specify a set of values for all the regressors, which represents the state of the world for which the simulation is conducted. This is referred to as the what-if or scenario analysis.

Suppose that you, as an automobile insurance company, have postulated that the distribution of the loss event frequency depends on five regressors (external factors): age of the policyholder, gender, type of car, miles driven annually, and policyholder's education level. Further, the distribution of the severity of each loss depends on three regressors: type of car, safety rating of the car, and annual household income of the policyholder (which can be thought of as a proxy for the luxury level of the car). Note that the frequency model regressors and severity model regressors can be different, as illustrated in this example.

Let these regressors be recorded, respectively, in the variables Age (scaled by a factor of 1/50), Gender (1: female, 2: male), CarType (1: sedan, 2: sport utility vehicle), AnnualMiles (scaled by a factor of 1/5,000), Education (1: high school graduate, 2: college graduate, 3: advanced degree holder), CarSafety (scaled to be between 0 and 1, the safest being 1), and Income (scaled by a factor of 1/100,000).

To illustrate, the following DATA steps simulate and record the historical data about the number of losses that various policyholders incur in a year in the variable NumLoss of the data table mycas.LossCounts, and the severity of each loss in the variable LossAmount of the data table mycas.Losses:

```
/* Simulate data for losses that several policyholders incur in a year */
data losses(keep=policyholderId age gender carType annualMiles
             education carSafety income noloss lossamount);
  call streaminit(12345);
  array cx{5} age gender carType annualMiles education;
  array cbeta{6} _TEMPORARY_ (1 -0.75 1 0.6 -1 -0.25);

  array sx{3} carType carSafety income;
  array sbeta{4} _TEMPORARY_ (3.5 1.5 -0.8 0.6);

  alpha = 1/3; theta = 1/alpha;
  Sigma = 1;
  do policyholderId=1 to 5000;
    /* simulate policyholder and vehicle attributes */
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then gender = 1; * female;
    else gender = 2; * male;

    if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
    else carType = 2; * SUV;

    annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;
```

```

educationLevel = rand('UNIFORM');
if (educationLevel < 0.5) then education = 1; *high school graduate;
else if (educationLevel < 0.85) then education = 2; *college graduate;
else education = 3; *advanced degree;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

/* simulate number of losses incurred by this policyholder */
cxbeta = cbeta(1);
do i=1 to dim(cx);
    cxbeta = cxbeta + cx(i) * cbeta(i+1);
end;
Mu = exp(cxbeta);
p = theta/(Mu+theta);
numloss = rand('NEGB',p,theta);

/* simulate severity of each loss */
if (numloss > 0) then do;
    noloss = 0;
    do iloss=1 to numloss;
        Mu = sbeta(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbeta(i+1);
        end;
        lossamount = exp(Mu) * rand('LOGNORMAL')**Sigma;
        output;
    end;
end;
else do;
    noloss = 1;
    lossamount = .;
    output;
end;
end;
run;

/* Aggregate number of annual loss events for each policyholder */
data losscounts(keep=age gender carType annualMiles education numloss);
set losses;
by policyholderId;
retain numloss 0;
if (noloss ne 1) then
    numloss = numloss + 1;
if (last.policyholderId) then do;
    output;
    numloss = 0;
end;
run;

/* Load data into the CAS server */
data mycas.losses;

```

```

    set losses;
run;
data mycas.losscounts;
    set losscounts;
run;

```

Note that the last two DATA steps in the previous program load the local SAS data sets Work.LossCounts and Work.Losses into the data tables mycas.LossCounts and mycas.Losses, respectively, in your CAS session that is associated with the mycas CAS engine libref.

The following PROC CNTSELECT step fits the count regression model and stores the fitted model information in the item store mycas.CountregModel:

```

/* Fit negative binomial frequency model for the number of losses */
proc cntselect data=mycas.losscounts store=mycas.countregmodel;
    model numloss = age gender carType annualMiles education / dist=negbin;
run;

```

You can examine the parameter estimates of the count model that are stored in the item store mycas.CountregModel by submitting the following statements:

```

/* Examine the parameter estimates for the model in the item store */
proc cntselect data=mycas.losscounts;
    viewstore / instore=mycas.countregmodel finalestimates;
run;

```

The “Parameter Estimates” table that is displayed by the SHOW statement is shown in [Figure 5.5](#).

Figure 5.5 Parameter Estimates of the Count Regression Model

The CNTSELECT Procedure

Model fit results restored from item store COUNTREGMODEL:

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.910479	0.090515	10.06	<.0001
age	1	-0.626803	0.058547	-10.71	<.0001
gender	1	1.025034	0.032099	31.93	<.0001
carType	1	0.615165	0.031153	19.75	<.0001
annualMiles	1	-1.010276	0.017512	-57.69	<.0001
education	1	-0.280246	0.021677	-12.93	<.0001
_Alpha	1	0.318403	0.020090	15.85	<.0001

The following PROC SEVSELECT step fits the severity scale regression models for all the common distributions that are predefined in the procedure:

```

/* Fit severity models for the magnitude of losses */
proc sevselect data=mycas.losses outest=mycas.sevregest print=all;
    loss lossamount;
    scalemodel carType carSafety income;
    dist _predef_;
    nloptions maxiter=100;
run;

```

The comparison of fit statistics of various scale regression models is shown in Figure 5.6. The scale regression model based on the lognormal distribution is deemed the best-fitting model according to the likelihood-based statistics, whereas the scale regression model based on the Pareto distribution is deemed the best-fitting model according to the statistics based on the empirical distribution function (EDF).

Figure 5.6 Severity Model Comparison

The SEVSELECT Procedure

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Burr	127231	127243	127243	127286	4.66462	75.26361	9.17877
Exp	128431	128439	128439	128467	3.61201	61.22673	4.17008
Gamma	128324	128334	128334	128370	4.44757	92.64608	8.25474
Igauss	127434	127444	127444	127480	3.65517	71.24503	5.95287
Logn	127062*	127072*	127072*	127107*	4.13167	71.45566	7.20527
Pareto	128166	128176	128176	128211	3.24411*	37.34514*	2.41164*
Gpd	128166	128176	128176	128211	3.24411	37.34527	2.41165
Weibull	128429	128439	128439	128475	3.65639	64.22930	4.54181

Asterisk (*) denotes the best model in the column.

Now, you are ready to analyze the distribution of the aggregate loss that can be expected from a specific policyholder. First, you need to encode and scale the policyholder's information into the appropriate regressor variables of a data table. This table is called the scenario data table. Note that you need to follow the same encoding and scaling method for the regressor variables in the scenario data table as you use for the regressor variables in the input data tables of the SEVSELECT and CNTSELECT procedures. The following DATA steps create the scenario data table mycas.SinglePolicy, which contains a scenario that consists of a 59-year-old male policyholder who has an advanced degree, earns 159,870, and drives a sedan that has a very high safety rating about 11,474 miles annually:

```
/* Generate the scenario data table for single policyholder */
data singlePolicy(keep=age gender carType annualMiles
                  education carSafety income);
    age = 59/50;           * age scaled by 50;
    gender = 2;            * male;
    carType = 1;           * sedan;
    annualMiles = 11474/5000; * miles scaled by 5,000;
    education = 3;         * advanced degree;
    carSafety = 0.99532;    * car safety in the range (0,1);
    income = 159870/100000; * annual income scaled by 100,000;
    output;
run;

/* Load data into the CAS server */
data mycas.singlePolicy;
    set singlePolicy;
run;
```

The following PROC CCDM step analyzes the compound distribution of the aggregate loss that the policyholder in the scenario data table mycas.SinglePolicy incurs in a year by using the frequency model from the item store mycas.CountregModel and the two best severity models, lognormal and Pareto, from the

data table mycas.SevRegEst:

```
/* Simulate the aggregate loss distribution for the scenario
   with single policyholder */
proc ccdm data=mycas.singlePolicy nreplicates=10000 seed=13579 print=all
      countstore=mycas.countregmodel severityest=mycas.sevregist;
      severitymodel logn pareto;
      outsum out=mycas.onepolicysum mean stddev skew kurtosis median
      pctlpts=97.5 to 99.5 by 1;
run;
```

The results from the preceding PROC CCDM step are shown in [Figure 5.7](#).

When you use a severity scale regression model, the “Compound Distribution Information” table displays the severity scale regression effects that PROC CCDM uses. PROC CCDM detects the severity regression effects automatically by examining the metadata that PROC SEVSELECT stores in the SEVERITYEST= data table and verifies the existence of the individual regressors in the DATA= data table.

Figure 5.7 Scenario Analysis Results for One Policyholder with Lognormal Severity Model

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Lognormal Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	206.68795	Median	0
Standard Deviation	412.86689	Interquartile Range	245.15277
Variance	170459.1	Minimum	0
Skewness	4.14952	Maximum	6395.5
Kurtosis	27.91648	Sample Size	10000

Figure 5.7 *continued*

Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	0
75	245.15277
95	979.49178
97.5	1355.4
98.5	1719.2
99	1993.0
99.5	2438.6
Percentile Method = 5	

The “Sample Summary Statistics” and “Sample Percentiles” tables in [Figure 5.7](#) show estimates of the aggregate loss distribution for the lognormal severity model. The average expected loss is about 206.7, and the worst-case loss, if approximated by the 97.5th percentile, is about 1,355.4. The percentiles table shows that the distribution is highly skewed to the right; this is also confirmed by the skewness estimate. The median estimate of 0 can be interpreted in two ways. One way is to conclude that the policyholder will not incur any loss in 50% of the years during which he or she is insured. The other way is to conclude that 50% of policyholders who have the characteristics of this policyholder will not incur any loss in a given year. However, there is a 2.5% chance that the policyholder will incur a loss that exceeds 1,355.4 in any given year and a 0.5% chance that the policyholder will incur a loss that exceeds 2,438.6 in any given year.

If the aggregate loss sample is simulated by using the Pareto severity model, then the results are as shown in [Figure 5.8](#). The average and worst-case losses are 205.3 and 1,359.7, respectively. These estimates are very close to the values that the lognormal severity model predicts.

Figure 5.8 Scenario Analysis Results for One Policyholder with Pareto Severity Model

The CCDM Procedure

Severity Model: Pareto
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Pareto Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

Figure 5.8 *continued***The CCDM Procedure**

Severity Model: Pareto
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	205.32253	Median	0
Standard Deviation	390.38856	Interquartile Range	254.28294
Variance	152403.2	Minimum	0
Skewness	3.18983	Maximum	4625.0
Kurtosis	14.22054	Sample Size	10000

Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	0
75	254.28294
95	999.32562
97.5	1359.7
98.5	1658.4
99	1855.1
99.5	2215.6
Percentile Method = 5	

The scenario that you just analyzed contains only one policyholder. You can expand the scenario to include multiple policyholders. The following DATA step simulates the data table `mycas.GroupOfPolicies`, which records information about five different policyholders, as shown in [Figure 5.9](#):

```
/* Generate the scenario data table for multiple policyholders */
data groupOfPolicies(keep=policyholderId age gender carType annualMiles
                      education carSafety income);
  call streaminit(67897);

  do policyholderId=1 to 5;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then gender = 1; * female;
    else gender = 2; * male;

    if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
    else carType = 2; * SUV;

    annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

    educationLevel = rand('UNIFORM');
    if (educationLevel < 0.5) then education = 1; *high school graduate;
    else if (educationLevel < 0.85) then education = 2; *college graduate;
```

```

else education = 3; *advanced degree;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

output;
end;
run;

/* Load data into the CAS server */
data mycas.groupOfPolicies;
set groupOfPolicies;
run;

```

Figure 5.9 Scenario Analysis Data for Multiple Policyholders

policyholderId	age	gender	carType	annualMiles	education	carSafety	income
1	1.18	2	1	2.2948	3	0.99532	1.59870
2	0.66	2	1	2.6718	2	0.86412	0.84459
3	0.64	2	2	1.9528	1	0.86478	0.50177
4	0.46	1	2	2.6402	2	0.27062	1.18870
5	0.62	1	1	1.7294	1	0.32830	0.37694

The following PROC CCDM step conducts a scenario analysis for the aggregate loss that is incurred by all five policyholders in the data table mycas.GroupOfPolicies together in one year:

```

/* Simulate the aggregate loss distribution for the scenario
of multiple policyholders */
proc ccdm data=mycas.groupOfPolicies nreplicates=10000 seed=13579 print=all
countstore=mycas.countregmodel severityest=mycas.sevregest
plots=(conditionaldensity(rightq=0.95)) nperturbedSamples=50;
severitymodel logn pareto;
outsum out=mycas.multipolicysum mean stddev skew kurtosis median
pctlpts=97.5 to 99.5 by 1;
run;

```

The preceding PROC CCDM step conducts perturbation analysis by simulating 50 perturbed samples. The perturbation summary results for the lognormal severity model are shown in [Figure 5.10](#), and the results for the Pareto severity model are shown in [Figure 5.11](#). If the severity of each loss follows the fitted lognormal distribution, then you can expect that the group of policyholders together incurs an average loss of $5,319 \pm 175$ and a worst-case loss of $15,841 \pm 501$ when you define the worst-case loss as the 97.5th percentile.

Figure 5.10 Perturbation Analysis of Losses from Multiple Policyholders with Lognormal Severity Model

The CCDM Procedure		
Severity Model: Logn Count Model: NegBin(p=2)		
Compound Distribution Information		
Severity Model	Lognormal Distribution	
Scale Model Effects	carSafety carType income	
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL	
Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	5319.4	175.09159
Standard Deviation	4190.8	142.93553
Variance	17583336	1197596.0
Skewness	2.09204	0.29198
Kurtosis	10.24691	5.01699
Number of Perturbed Samples = 50 Size of Each Sample = 10000		
Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	198.85858	25.46560
5	748.40992	49.07759
25	2380.2	103.20610
50	4326.2	160.64004
75	7137.5	238.65458
95	13179.6	377.35981
95	13179.6	377.35981
97.5	15841.4	500.94984
98.5	17871.7	544.29623
99	19579.2	639.20323
99.5	22728.4	917.34826
Number of Perturbed Samples = 50 Size of Each Sample = 10000		

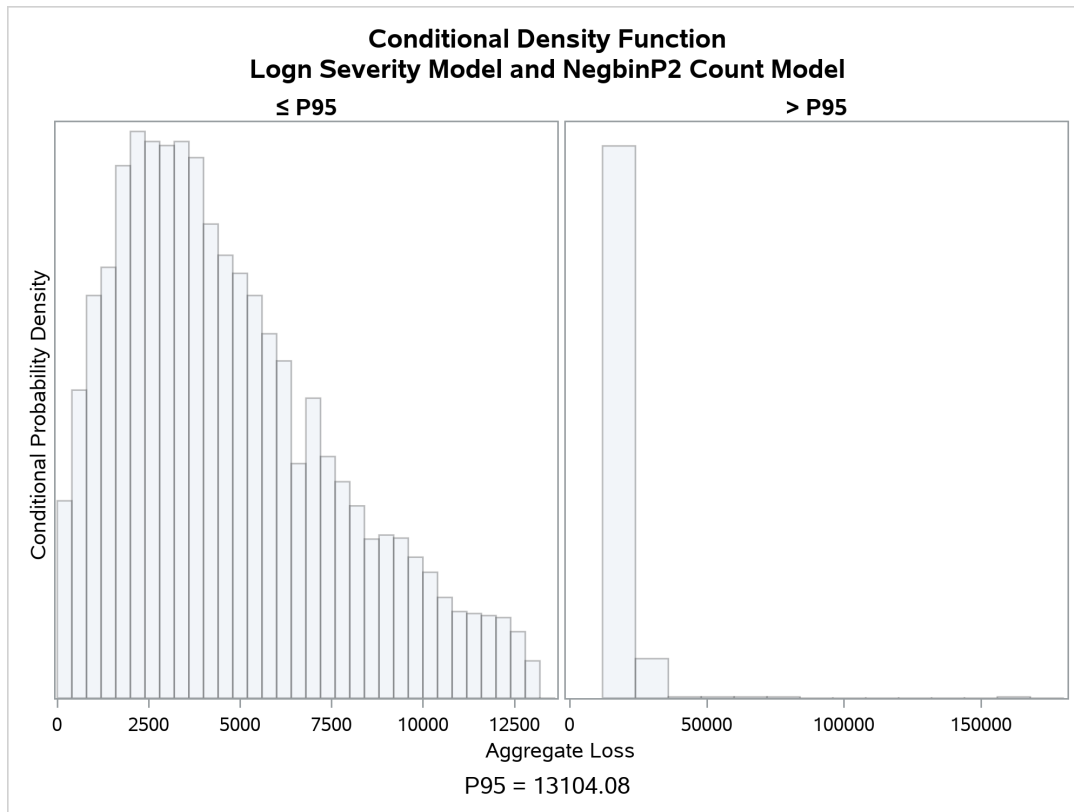
If the severity of each loss follows the fitted Pareto distribution, then you can expect an average loss of $5,251 \pm 168$ and a worst-case loss of $15,091 \pm 458$.

If you decide to use the 99.5th percentile to define the worst-case loss, then the worst-case loss is $22,728 \pm 917$ for the lognormal severity model and $20,497 \pm 747$ for the Pareto severity model. The numbers for lognormal and Pareto are well within one or two standard errors of each other, which indicates that the aggregate loss distribution is less sensitive to the choice of these two severity distributions in this particular example. You can use the results from either of them.

Figure 5.11 Perturbation Analysis of Losses from Multiple Policyholders with Pareto Severity Model

The CCDM Procedure		
Severity Model: Pareto Count Model: NegBin(p=2)		
Compound Distribution Information		
Severity Model	Pareto Distribution	
Scale Model Effects	carSafety carType income	
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL	
Sample Perturbation Analysis		
Statistic	Estimate	Standard Error
Mean	5251.1	167.69834
Standard Deviation	3921.6	118.27373
Variance	15392672	921516.4
Skewness	1.50020	0.09646
Kurtosis	3.82073	0.94306
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		
Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	160.42723	25.50486
5	704.45468	46.67739
25	2382.6	101.04928
50	4369.0	155.02402
75	7161.0	229.22691
95	12773.4	364.41086
95	12773.4	364.41086
97.5	15091.8	458.00124
98.5	16779.3	525.18704
99	18119.6	599.56740
99.5	20496.9	747.39377
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

The `PLOTS=CONDITIONALDENSITY` option that is used in the preceding `PROC CCDM` step creates the conditional density plots for the body and right-tail regions of the density function of the aggregate loss. The plots for the aggregate loss sample that is generated by using the lognormal severity model are shown in [Figure 5.12](#). The plot on the left side is the plot of $\Pr(Y|Y \leq 13,104)$, where the limit 13,104 is the 95th percentile as specified by the `RIGHTQ=0.95` suboption of the `PLOTS=CONDITIONALDENSITY` option. The plot on the right side is the plot of $\Pr(Y|Y > 13,104)$, which helps you visualize the right-tail region of the density function. You can also request the plot of the left tail by specifying the `LEFTQ=` suboption if you want to explore the details of the left tail region. The conditional density plots are always produced by using the unperturbed sample.

Figure 5.12 Conditional Density Plots for the Aggregate Loss of Multiple Policyholders

Syntax: CCDM Procedure

The following statements are available in the CCDM procedure:

```
PROC CCDM options ;
  BY variable-list ;
  SEVERITYMODEL severity-model-list < / definition-option > ;
  EXTERNALCOUNTS COUNT=frequency-variable < ID=replication-id-variable > ;
  COUNTMODEL distribution-name(parameter-value < (value-option) > < , > ... parameter-value
    < (value-option) > ) ;
  SIMULATEDSYMBOL symbol ~ distribution-name(parameter-value-list) < ... symbol ~ distribution-
    name(parameter-value-list) > ;
  OUTPUT OUT=CAS-libref.data-table < variable-name-options > < / out-option > ;
  OUTSUM OUT=CAS-libref.data-table statistic-keyword=variable-name < ... statistic-
    keyword=variable-name > > < outsum-options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
Programming statements ;
```

Functional Summary

Table 5.1 summarizes the statements and options available in the CCDM procedure.

Table 5.1 PROC CCDM Functional Summary

Description	Statement	Option
Statements		
Specifies the names of severity distribution models	SEVERITYMODEL	
Specifies externally simulated count data	EXTERNALCOUNTS	
Specifies a count distribution that has fixed parameter values	COUNTMODEL	
Specifies simulated symbols and their probability distributions	SIMULATEDSYMBOL	
Specifies where and how the full simulated samples are written	OUTPUT	
Specifies where and how the summary statistics of simulated samples are written	OUTSUM	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	
Specifies programming statements that define an objective function	Programming statements	
Data Table Options		
Specifies the input data table	PROC CCDM	DATA=
Specifies the output data table for the individual severity draws	PROC CCDM	OUTDRAW=
Specifies the output data table for the full simulated samples	OUTPUT	OUT=
Specifies the output data table for the summary statistics	OUTSUM	OUT=
Model Input Options		
Specifies the variable that contains externally simulated counts	EXTERNALCOUNTS	COUNT=
Specifies the item store that contains the frequency (count) model	PROC CCDM	COUNTSTORE=
Specifies the replicate identifier variable for external counts	EXTERNALCOUNTS	ID=
Specifies the input data table for parameter estimates of the severity models	PROC CCDM	SEVERITYEST=
Specifies the item store for parameter estimates of the severity models	PROC CCDM	SEVERITYSTORE=

Table 5.1 *continued*

Description	Statement	Option
Simulation Options		
Specifies the adjusted severity symbols in the programming statements	PROC CCDM	ADJUSTEDSEVERITY=
Specifies that the covariance estimates be ignored for perturbation analysis	PROC CCDM	IGNORECOV
Specifies the left-truncation threshold for truncating a severity distribution	PROC CCDM	LEFTTRUNCATION=
Specifies the upper limit on the count for each sample point	PROC CCDM	MAXCOUNTDRAW=
Specifies the number of parameter-perturbed samples to be simulated	PROC CCDM	NPERTURBEDSAMPLES=
Specifies a number that controls the size of the simulated sample	PROC CCDM	NREPLICATES=
Specifies how the perturbed samples are generated	PROC CCDM	PERTURBMODE=
Specifies the right-truncation limit for truncating a severity distribution	PROC CCDM	RIGHTTRUNCATION=
Specifies a seed for the internal pseudorandom number generator	PROC CCDM	SEED=
Specifies the type of aggregate loss you want to simulate	PROC CCDM	SIMULATIONMODE=
Output Preparation Options		
Specifies the variables for the aggregate adjusted loss samples	OUTPUT	ADJSAMPLEVAR=
Specifies that a backward compatible output style be used for the case of a single adjusted severity symbol	PROC CCDM	ONEADJSEVCOMPAT
Specifies the names of the variables for percentiles	OUTSUM	PCTLNAME=
Specifies the decimal precision to form default percentile variable names	OUTSUM	PCTLNDEC=
Specifies percentiles to compute and report	OUTSUM	PCTLPTS=
Specifies the method to compute the percentiles	PROC CCDM	PCTLDEF=
Specifies that all perturbed samples be written to the OUT= data table	OUTPUT	PERTURBOUT
Specifies the variable for the aggregate loss sample	OUTPUT	SAMPLEVAR=
Specifies that zero-valued aggregate losses be removed from the aggregate loss sample	PROC CCDM	TRUNCATEZEROS
Specifies the denominator for computing second- and higher-order moments	PROC CCDM	VARDEF=

Table 5.1 continued

Description	Statement	Option
Displayed Output Options		
Suppresses all displayed output	PROC CCDM	NOPRINT
Specifies which graphical output to create	PROC CCDM	PLOTS=
Specifies which displayed output to produce	PROC CCDM	PRINT=

PROC CCDM Statement

PROC CCDM options ;

The PROC CCDM statement invokes the procedure. You can specify the following *options*, which are listed in alphabetical order.

ADJUSTEDSEVERITY=*symbol-name*

ADJSEV=*symbol-name*

ADJUSTEDSEVERITY=(*symbol-name* <... *symbol-name*>)

ADJSEV=(*symbol-name* <... *symbol-name*>)

names one or more symbols that represent an adjusted severity value in the SAS programming statements that you specify. If you specify more than one *symbol-name*, then separate them with spaces and enclose them in parentheses. Each *symbol-name* is a SAS name that conforms to the naming conventions of a SAS variable. For more information, see the section “[Programming Statements](#)” on page 138.

COUNTSTORE=*CAS-libref.data-table*

names the input data table that contains all the information about the frequency (count) model in the form of an item store. The CNTSELECT procedure generates this item store data table when you use the STORE= option.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the DATA= option.

The exogenous variables in the frequency model, if any, are deduced from this item store. The DATA= data table must contain all those variables.

If you specify a BY statement in the PROC CNTSELECT step that creates the COUNTSTORE= item store, then you must specify an identical BY statement in the PROC CCDM step.

You must specify this option if you do not specify the COUNTMODEL or EXTERNALCOUNTS statement. This option is ignored if you specify the EXTERNALCOUNTS statement, because PROC CCDM does not need to simulate frequency counts internally when you specify externally simulated counts.

DATA=CAS-libref.data-table

names the input data table for PROC CCDM to use. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 96.

data-table specifies the name of the input data table.

The DATA= data table specifies information about the scenario for which you want to estimate the aggregate loss distribution. It is expected to contain the values of regression variables in the frequency or severity model and severity adjustment variables that you use in the programming statements.

IGNORECOV

IGNOREPARMCOVARIANCE

specifies that the covariance estimates be ignored for perturbation analysis. If you specify this option, then even if the covariance estimates of severity or frequency model parameters are valid, they are not used for perturbation analysis, and only standard errors are used. This enables you to force independence among parameters. However, perturbation analysis based on covariance estimates is more accurate than perturbation analysis based on the standard errors alone.

For more information about parameter perturbation analysis, see the section “[Parameter Perturbation Analysis](#)” on page 161.

LEFTTRUNCATION=*number*

LTRUNC=*number*

specifies the left-truncation threshold for truncating a severity distribution.

Specifying this option, and optionally the **RIGHTTRUNCATION=** option, enables you to simulate from a truncated severity distribution. For more information, see the section “[Making a Random Draw from a Truncated Severity Distribution](#)” on page 147.

MAXCOUNTDRAW=*number*

MAXCOUNT=*number*

specifies an upper limit on the number of loss events (count) that is used for simulating one aggregate loss sample point. If *number* is equal to $N_{\max}$, then any count greater than $N_{\max}$ is assumed to be equal to $N_{\max}$, and only $N_{\max}$ severity draws are made to compute one point in the aggregate loss sample.

If you specify this option and also specify the COUNTSTORE= item store or the COUNTMODEL statement, then the limit is applied to each count that PROC CCDM randomly draws from the count distribution. Any count draw larger than *number* is replaced by *number*.

If you specify this option and also specify the EXTERNALCOUNTS statement, then the limit is applied to each observation in the DATA= data table, and any value of the COUNT= variable larger than *number* is replaced by *number*.

If you do not specify this option, then a default value of 1,000 is used.

If you specify a *number* significantly larger than 1,000, then PROC CCDM might take a very long time to complete the simulation, especially when some counts are closer to the limit.

NOPRINT

turns off all displayed output. If you specify this option, then PROC CCDM ignores any value that you specify for the PRINT= option.

NPERTURBEDSAMPLES=number**NPERTURB=number**

performs parameter perturbation analysis, where *number* determines how many perturbed parameter sets are generated. A separate full sample is simulated for each set of perturbed parameter values. The summary statistics and percentiles are computed for each such perturbed sample, and their values are aggregated across the samples to compute the mean and standard deviation of each summary statistic and percentile.

The parameter perturbation analysis makes random draws of parameter values from a multivariate normal distribution if the covariance estimates of the parameters are available. For the multivariate normal distribution of severity model parameters, PROC CCDM attempts to read the covariance estimates from the SEVERITYEST= data table or the SEVERITYSTORE= item store. For the multivariate normal distribution of count model parameters, PROC CCDM attempts to read the covariance estimates from the item store that is specified in the COUNTSTORE= option. If covariance estimates are not available or valid, or if you specify the IGNORECOV option, then for each parameter, PROC CCDM makes a random draw from the univariate normal distribution whose mean and standard deviation are equal to that parameter's point estimate and standard error, respectively. The standard errors for severity model parameters are read from either the SEVERITYEST= data table or the SEVERITYSTORE= item store. The standard errors for count model parameters are either read from the COUNTSTORE= item store or you must specify them in the COUNTMODEL statement. If neither covariance nor standard error estimates are available for any severity or frequency model parameter, then perturbation analysis is not conducted.

If you specify the PRINT=ALL or PRINT=PERTURBSUMMARY option, then a summary of the perturbation analysis is printed for the core summary statistics and the percentiles of the aggregate loss distribution. If you specify the OUTSUM statement, then the requested summary statistics are written to the OUTSUM= data table for each perturbed sample. You can also optionally request that each perturbed sample be written in its entirety to the OUT= data table by specifying the PERTURBOUT option in the OUTPUT statement.

For more information about parameter perturbation analysis, see the section “[Parameter Perturbation Analysis](#)” on page 161.

NREPLICATES=number**NREP=number**

specifies a *number* that controls the size of the compound distribution sample that PROC CCDM simulates. The *number* is interpreted differently based on whether you specify the EXTERNALCOUNTS statement.

If you do not specify the EXTERNALCOUNTS statement, then the sample size is equal to *number*. If you do not specify this option, then a default value of 100,000 is used.

If you specify the EXTERNALCOUNTS statement, then the number of replicates that you specify in the DATA= data table is multiplied by *number* to get the total size of the compound distribution sample. If you do not specify this option, then a default value of 1 is used.

ONEADJSEVCOMPAT

specifies that the output of the case of single adjusted severity symbol be prepared by using the same method as used by the versions of the CCDM procedure prior to SAS Econometrics 8.5. In particular, specifying this option causes the following output-related changes, where *sym* denotes the name of the single adjusted severity symbol that you specify in the [ADJUSTEDSEVERITY=](#) option:

- If you specify the OUTPUT statement, then the default name of the adjusted severity variable in the OUTSAMPLE= table is `_AGGADJSEV_` instead of *sym*.
- If you specify the OUTSUM statement, then the default value in the `_SAMPLEVAR_` column of the OUTSUM= table is `_AGGADJSEV_` instead of *sym* for the rows that correspond to the aggregate adjusted severity.
- The ODS path for the adjusted severity group's output table or plot uses `AggregateAdjustedLoss` instead of *sym*.
- The label for the adjusted severity group in the table of contents of a SAS client is "Aggregate Adjusted Loss" instead of "Aggregate Adjusted Loss *sym*."
- The title of the SummaryStatistics ODS table is "Adjusted Sample Summary Statistics" instead of "Summary Statistics for *sym*."
- The title of the PerturbedSummary ODS table is "Adjusted Sample Perturbation Analysis" instead of "Sample Perturbation Analysis for *sym*."
- The title of the Percentiles ODS table is "Adjusted Sample Percentiles" instead of "Percentiles for *sym*."
- The title of the PerturbedPctlSummary ODS table is "Adjusted Sample Percentile Perturbation Analysis" instead of "Percentile Perturbation Analysis for *sym*."
- The titles of plots use just "Adjusted Sample" instead of "Adjusted Sample *sym*."
- The X-axis variable label in plots is just "Aggregate Adjusted Loss" instead of "Aggregate Adjusted Loss *sym*."

When you do not specify this option, the style of the output for the case of a single adjusted severity symbol is no different from the style of the output for the case of more than one adjusted severity symbol.

OUTDRAW=CAS-libref.data-table

specifies the output data table to contain the individual severity random draws. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section "Using CAS Sessions and CAS Engine Librefs" on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

For more information about the variables in the OUTDRAW= data table, see the section "[OUTDRAW= Data Table](#)" on page 169.

PCTLDEF=percentile-method

specifies the method of computing the percentiles of the compound distribution. The *percentile-method* can be 1, 2, 3, 4, or 5. By default, PCTLDEF=5. For more information, see the description of the PCTLDEF= option in the UNIVARIATE procedure in the [Base SAS Procedures Guide: Statistical Procedures](#).

PERTURBMODE=mode

PERTURBMETHOD=mode

specifies a method to generate perturbed samples.

This option has no effect if you do not specify the **NPETURBEDSAMPLES=** option or if you specify the **EXTERNALCOUNTS** statement.

Let P and W denote the values of **NPETURBEDSAMPLES=** option and number of worker nodes in the CAS session that the procedure is running in, respectively. Also, let N denote the value of the **NREPLICATES=** option.

You can specify one of the following *modes*:

AUTO | SYNC | 1

generates perturbed samples in either distributed or partitioned manner depending on the values of P and W as follows:

- If $P < W$, then each perturbed sample is distributed across all W workers, such that each worker generates a subsample of approximately N/W points of each perturbed aggregate loss sample. If you specify the **PERTURBOUT** option in the **OUTPUT** statement, then observations that have the same value of the **_DRAWID_** variable are distributed across all worker nodes. The controller node needs to gather the subsamples from all the worker nodes in order to compute the percentiles, and perturbed samples are generated sequentially for each severity model within each BY group.
- If $P \geq W$, then perturbed samples are generated in a partitioned manner such that each worker node generates approximately P/W perturbed samples, each of size N . If you specify the **PERTURBOUT** option in the **OUTPUT** statement, then observations that have the same value of the **_DRAWID_** variable are available on only one of the worker nodes. There is no need for communicating any sample data to the controller node, because each worker node has enough data to compute the percentiles of each perturbed sample. The controller node gathers only the summary statistics and percentiles of perturbed samples from all worker nodes. Further, each worker node can simulate its assigned P/W perturbed samples independently of other worker nodes. Together, these two aspects make the perturbation analysis more efficient.

DIST | SYNCDIST | 2

always generates perturbed samples in distributed manner irrespective of the values of P and W . This option's behavior is equivalent to the behavior of the **AUTO** option for the $P < W$ case.

It is recommended that you specify the **AUTO** method. By default, **PERTURBMETHOD=DIST** to ensure that the current release of PROC CCDM produces, by default, the same perturbation results as releases prior to SAS Econometrics 8.4.

PLOTS <(global-plot-options)> =plot-request-option

PLOTS <(global-plot-options)> =(plot-request-option ... plot-request-option)

specifies the desired graphical output.

By default, the CCDM procedure produces no graphical output.

When you request plots by using this option, PROC CCDM creates a plot data table on the server. This table contains an estimate of the empirical distribution function (EDF) for a carefully selected

subsample of the simulated aggregate loss sample. PROC CCDM brings the plot data table back to the client machine where your SAS session is running and uses it to create all the plots that you specify.

You can specify the following *global-plot-options*:

ONLY

turns off the default graphical output and creates only the requested plots.

NOBS=number

specifies the approximate number of observations in the plot data table. The larger the *number*, the more accurate are the plots, but also the larger the size of the data that need to be brought back to the client machine. A larger plot data table not only requires more time and resources to prepare but can also cost you additional money if the data movement on the client-server connection is metered.

If you specify a nonzero *number*, then it must be greater than 10. If you do not specify this option or if you specify NOBS=0, then a default value of 5,000 is used.

NADJSEVTOPLOT=k

NADJSEV=k

specifies the number of aggregate adjusted loss samples for which to create the requested plots. PROC CCDM creates plots for the first *k* adjusted severity symbols that you specify in the [ADJUSTEDSEVERITY=](#) option.

This option is ignored if you do not specify the [ADJUSTEDSEVERITY=](#) option and the programming statements to adjust the simulated severity values.

If you specify *m* symbols in the [ADJUSTEDSEVERITY=](#) option and if $k > m$, then PROC CCDM creates the requested plots for all adjusted severity symbols.

Note that PROC CCDM prepares a separate plot data table for each of the *k* aggregate adjusted loss samples, so the amount of data that PROC CCDM needs to bring back to the client machine increases linearly with the value of *k*.

By default, PROC CCDM does not create plots for aggregate adjusted loss samples—that is, NADJSEVTOPLOT=0 by default.

You control which plots to create for each aggregate loss sample by specifying the *plot-request-options*. If you specify more than one *plot-request-option*, then separate them with spaces and enclose them in parentheses. You can specify the following *plot-request-options*:

ALL

displays all the graphical output.

CONDITIONALDENSITY (*conditional-density-plot-options*)

CONDPDF (*conditional-density-plot-options*)

creates a group of plots of the conditional density functions estimates. The group contains at most three plots, each conditional on the value of the aggregate loss being in one of the three regions that are defined by the quantiles that you specify in the following *conditional-density-plot-options*:

LEFTQ=number

specifies the quantile in the range (0,1) that marks the end of the left-tail region. If you specify a value of l for *number*, then the left-tail region is defined as the set of values that are less than or equal to q_l , where q_l is the l th quantile. For the left-tail region, nonparametric estimates of the conditional probability density function $f_S^l(s) = \Pr[S = s | S \leq q_l]$ are plotted. The value of q_l is estimated by the 100/ l th percentile of the simulated compound distribution sample.

If you do not specify this option or you specify a missing value for this option, then the left-tail region is not plotted.

RIGHTQ=number

specifies the quantile in the range (0,1) that marks the beginning of the right-tail region. If you specify a value of r for *number*, then the right-tail region is defined as the set of values that are greater than q_r , where q_r is the r th quantile. For the right-tail region, nonparametric estimates of the conditional probability density function $f_S^r(s) = \Pr[S = s | S > q_r]$ are plotted. The value of q_r is estimated by the 100/ r th percentile of the simulated compound distribution sample.

If you do not specify this option or you specify a missing value for this option, then the right-tail region is not plotted.

You must specify a nonmissing value for at least one of the preceding two options. For the region between the LEFTQ= and RIGHTQ= quantiles, which is referred to as the central or body region, nonparametric estimates of the conditional probability density function $f_S^c(s) = \Pr[S = s | q_l < S \leq q_r]$ are plotted. If you do not specify a LEFTQ= value, then q_l is assumed to be 0. If you do not specify a RIGHTQ= value, then q_r is assumed to be ∞ .

DENSITY

creates a plot of the nonparametric estimates of the probability density function (in particular, histogram and kernel density estimates) of the compound distribution.

EDF <(edf-plot-option)>

creates a plot of the nonparametric estimates of the cumulative distribution function of the compound distribution.

You can request that the confidence interval be plotted by specifying the following *edf-plot-option*:

ALPHA=number

specifies the confidence level in the (0,1) range that is used for computing the confidence intervals for the EDF estimates. If you specify a value of α for *number*, then the upper and lower confidence limits for the confidence level of $100(1 - \alpha)$ are plotted.

NONE

displays none of the graphical output. If you specify this option, then it overrides all other plot request options. The default graphical output is also suppressed.

PRINT <(*global-display-option*)> =*display-option*

PRINT <(*global-display-option*)> =(*display-option* ... *display-option*)

specifies the desired displayed output. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses.

You can specify the following *global-display-option*:

ONLY

turns off the default displayed output and displays only the requested output.

You can specify the following *display-options*:

ALL

displays all the output.

NONE

displays no output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

PERCENTILES

displays the percentiles of the compound distribution sample. This includes all the predefined percentiles and percentiles that you request in the OUTSUM statement.

PERTURBSUMMARY

displays the mean and standard deviation of the summary statistics and percentiles that are taken across all *P* samples that the procedure produces by perturbing the model parameters, where *P* is the value of the NPERTURBEDSAMPLES= option that you specify in the PROC CCDM statement.

SUMMARYSTATISTICS | SUMSTAT

displays the summary statistics of the compound distribution sample.

If you do not specify the PRINT= option or the ONLY *global-display-option*, then the procedure behaves as if you have specified PRINT=(SUMMARYSTATISTICS).

RIGHTTRUNCATION=*number*

RTRUNC=*number*

specifies the right-truncation limit for truncating a severity distribution.

Specifying this option, and optionally the [LEFTTRUNCATION](#)= option, enables you to simulate from a truncated severity distribution. For more information, see the section “[Making a Random Draw from a Truncated Severity Distribution](#)” on page 147.

SEED=*number*

specifies the integer to use as the seed in generating the pseudorandom numbers that are used for simulating severity and frequency values.

If you do not specify the seed, or if *number* is negative or 0, then PROC CCDM uses the time of day from the computer’s clock as the seed.

SEVERITYEST=CAS-libref.data-table

names the input data table that contains the parameter estimates for the severity models. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the *DATA=* option. This data table must be an *OUTEST=* data table that the *SEVSELECT* procedure produces.

The names of the regression variables in the scale regression models, if any, are deduced from this data table. If the severity models are scale regression models, then the *DATA=* data table must contain all the regressors in the scale regression models.

To ensure that PROC CCDM correctly matches the values of regressors and the values of regression parameter estimates, you need to ensure that names of the regressors in the *DATA=* data table match the names of the regressors that you specify in the *SCALEMODEL* statement of the PROC *SEVSELECT* step that fits the severity models.

If you specify a *BY* statement in the PROC *SEVSELECT* step that creates the *SEVERITYEST=* data table, then you must specify an identical *BY* statement in the PROC CCDM step.

SEVERITYSTORE=CAS-libref.data-table**SEVSTORE=CAS-libref.data-table**

specifies the input data table that contains the context and estimates of the severity models in the form of an item store. Specifying the *OUTSTORE=* option in a PROC *SEVSELECT* step creates this item store data table.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the *DATA=* option.

If your severity models contain classification or interaction effects, then you need to use this option instead of the *SEVERITYEST=* option to specify the estimates of the severity models. If you specify this option, you cannot specify the *SEVERITYEST=* option.

If you specify a *BY* statement in the PROC *SEVSELECT* step that creates the *SEVERITYSTORE=* item store, then you must specify an identical *BY* statement in the PROC CCDM step.

SIMULATIONMODE=mode

specifies the mode of simulating aggregate loss. You can specify one of the following *modes*:

COLLECTIVERISK**CR**

specifies the collective risk mode. Aggregate loss is defined as $S = \sum_{i=1}^N X_i$, where X_i denotes a continuous random variable that represents the severity of one loss event and N denotes the discrete random variable that represents the number of loss events you expect to see in a particular time period. The severity random variables $\{X_i\}$ that are associated with all loss events are assumed to be iid. For each random frequency draw N , PROC CCDM makes N random draws from the severity distribution and adds those severity draws to compute one sample point in the aggregate loss sample.

If you specify this mode (or choose it by default) and if you specify the frequency distribution by using the *COUNTMODEL* statement, then the count distribution must be a discrete distribution.

CUSTOM**CSTM**

specifies the custom mode. In this mode, you use programming statements to combine the severity and frequency values according to your business logic in order to compute the aggregate loss. If you specify this mode, then you must also use the `ADJUSTEDSEVERITY=` option to specify an aggregate loss symbol that your programming statements compute. For each random frequency draw N , PROC CCDM makes one random draw, X , from the severity distribution and sets the keyword symbols `_FREQ_` and `_SEV_` to N and X , respectively. It then executes your programming statements to compute one sample point in the aggregate loss sample.

This mode is especially useful if you also use the `SIMULATEDSYMBOL` statement to specify some stochastic variables that you have externally estimated to follow certain parametric probability distributions. For example, the following statements simulate a sample of the total paid loss `paidLoss` as $P = NEX + C$, where N denotes the average frequency of loss events per exposure and follows a lognormal distribution with $\mu = 3$ and $\sigma = 0.5$, E denotes the estimate of the total exposure in a particular time period and follows a normal distribution $\mathcal{N}(1000, 75)$, X denotes the average severity of a loss event, and C denotes the random change in losses from one time period to next and follows a normal distribution $\mathcal{N}(500, 10)$:

```
proc ccdm severitystore=mycas.sevstore simulationmode=custom
    adjustedseverity=paidLoss;
    countmodel lognormal(3,0.5);
    simulatedsymbol E ~ normal(1000, 75) C ~ normal(500, 10);
    paidLoss = _freq_ * E * _sev_ + C;
run;
```

PUREPREMIUM**PP**

specifies the pure premium mode. Aggregate loss is defined as $S = NX$, where N denotes the average frequency of loss events you expect to see in a particular time period and X denotes a continuous random variable that represents the average severity of the loss events in that time period. For each random frequency draw N , PROC CCDM makes one random draw, X , from the severity distribution and multiplies N and X to compute one sample point in the aggregate loss sample.

By default, `SIMULATIONMODE=COLLECTIVERISK`.

TRUNCATEZEROS

truncates (removes) zero-valued aggregate loss values from the aggregate loss sample.

The value of the aggregate loss is 0 when the sum of randomly drawn counts for all observations is 0 for the internally simulated counts or when the total count for a replication is 0 in the external counts data table.

If you omit this option, then PROC CCDM keeps the zero-valued aggregate losses in the output sample.

VARDEF=divisor

specifies the divisor to use in the calculation of variance, standard deviation, kurtosis, and skewness of the compound distribution sample. If the sample size is N , then you can specify one of the following values for the *divisor*:

DF

sets the divisor for variance to $N - 1$. This also changes the definitions of skewness and kurtosis.

N

sets the divisor to N .

By default, VARDEF=DF. For more information, see the section “Descriptive Statistics” on page 166.

BY Statement

BY *variable-list* ;

You can use the BY statement in the CCDM procedure to process the input data table in groups of observations defined by the BY variables.

If you specify the BY statement, then unlike procedures that do not use a CAS server, PROC CCDM does not need the input data table to be sorted in the order of the BY variables. As a consequence, the NOTSORTED and DESCENDING options have no effect and PROC CCDM ignores them.

COUNTMODEL Statement

COUNTMODEL *distribution-name*(*parameter-value* < (*value-option*)>< , > ... *parameter-value* < (*value-option*)>) ;

The COUNTMODEL statement enables you to specify a parametric count model that has fixed parameter values. You must specify this statement if you do not specify the COUNTSTORE= option or the EXTERNALCOUNTS statement. On the other hand, PROC CCDM ignores this statement if you specify either the COUNTSTORE= option or the EXTERNALCOUNTS statement.

You must specify the following:

distribution-name

identifies your desired distribution. For a list of supported distributions and their acceptable names, see Table 5.2 or Table 5.3 in the section “Fixed Parameter-Value Distributions” on page 162. Each distribution accepts a fixed number of parameters in a specific order.

parameter-values

specifies the values of the distribution parameters. You must specify as many *parameter-values* as the expected number of parameters for the distribution that corresponds to *distribution-name*. PROC CCDM associates your specified values to parameters in the order the parameters are listed from top to bottom in the Parameters column of Table 5.2 or Table 5.3. The Parameters column also specifies the acceptable bounds, if any, for each parameter. Your specified *parameter-value* must be within those bounds.

You can optionally specify a standard error for each parameter by enclosing the following *value-option* in parentheses immediately after a parameter's value:

STDERROR=*number*

SE=*number*

number

specifies the standard error of the parameter. If you specify a value v and standard error s for a parameter, then PROC CCDDM assumes that the parameter follows a normal distribution $\mathcal{N}(v, s)$.

You can separate the parameter value specifications by using either a comma or a space.

The following example COUNTMODEL statement specifies that the counts follow the standard negative binomial distribution with parameters $p = 0.7$ and $n = 5$:

```
countmodel negbin(0.7 5);
```

If you request perturbation analysis by specifying a positive value for the **NPURTURBEDSAMPLES=** option and the parameters of your count distribution are stochastic, then it is recommended that you specify the standard error for each parameter. The following COUNTMODEL statement specifies that the counts follow the Weibull distribution with parameters $\theta = 50$ and $\tau = 0.75$ such that the standard errors for the θ and τ parameters are 3.5 and 0.25, respectively:

```
countmodel weibull(50(se=3.5), 0.75(0.25));
```

With this specification, when PROC CCDDM conducts the perturbation analysis, it sets the perturbed values of θ and τ to randomly drawn quantiles from the normal distributions $\mathcal{N}(50, 3.5)$ and $\mathcal{N}(0.75, 0.25)$, respectively.

PROC CCDDM permits a continuous count distribution (such as the Weibull distribution in the preceding example) only if you use the **SIMULATIONMODE=** option to specify a simulation mode other than the collective risk mode. For example, in the pure premium simulation mode (**SIMULATIONMODE=PP**), the count might represent an average claim frequency that an insurance company observes for a type of policy in a particular time period. Such an average frequency can have fractional values that are best modeled by a continuous distribution.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 172. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the CAS output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 172. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

EXTERNALCOUNTS Statement

EXTERNALCOUNTS **COUNT**=*frequency-variable* <**ID**=*replication-id-variable*> ;

The EXTERNALCOUNTS statement enables you to specify externally simulated frequency counts. By default, PROC CCDM internally simulates the number of loss events by using the frequency model from the COUNTSTORE= item store or the COUNTMODEL statement. However, if you specify the EXTERNALCOUNTS statement, then PROC CCDM uses the counts that you specify in the DATA= data table and simulates only the severity values internally.

If you specify more than one EXTERNALCOUNTS statement, only the first one is used.

You must specify the following option in the EXTERNALCOUNTS statement:

COUNT=*count-variable*

specifies the variable that contains the simulated counts. This variable must be present in the DATA= data table.

You can also specify the following option in the EXTERNALCOUNTS statement:

ID=*replication-id-variable*

specifies the variable that contains the replicate identifier. This variable must be present in the DATA= data table.

When you specify this option, you must partition the observations of the DATA= data table by the BY variables, if any, and the ID= variable. One way to partition the table is to run the following DATA step when you upload a local SAS data set to a CAS table on the CAS server, where *BY-variable-list* is the list of BY variables that is identical to the list in the BY statement:

```
data mycas.counts (partition=(<BY-variable-list> <replication-id-variable>));
  set inlib.counts;
run;
```

If the DATA= data table is already loaded into the CAS server and not partitioned by the required set of variables, then one way to partition the table is to use PROC CAS to run the **partition** action.

If you do not specify the BY statement, then the DATA= data table must be partitioned by just the ID= variable.

If you do not specify the ID= option, then PROC CCDM assumes that each observation represents one replication. In other words, the observation number serves as the default replication identifier.

The simulation process of using the external counts to generate the compound distribution sample is described in the section “[Simulation with External Counts](#)” on page 143.

OUTPUT Statement

OUTPUT **OUT**=*CAS-libref.data-table* < *variable-name-option* > < / *out-option* > ;

The OUTPUT statement enables you to specify the data table to output the generated compound distribution sample.

If you specify more than one OUTPUT statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*CAS-libref.data-table*

OUTSAMPLE=*CAS-libref.data-table*

specifies the output data table to contain the simulated compound distribution sample. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the DATA= option. If you specify programming statements to adjust individual severity values, then this data table contains both unadjusted and adjusted samples.

You can specify the following *variable-name-options* to control the names of the variables that are created in the OUT= data table:

ADJSAMPLEVAR=*variable-name*

ADJSAMPLEVAR=(*variable-name* < ... *variable-name* >)

specifies the names of the variables to contain the adjusted compound distribution sample in the OUT= data table. If you specify more than one *variable-name*, then separate them with spaces and enclose them in parentheses.

This option is ignored if you do not specify the [ADJUSTEDSEVERITY=](#) option and the programming statements to adjust the simulated severity values.

If you specify k names in the ADJSAMPLEVAR= option and if you specify m symbols in the ADJUSTEDSEVERITY= option such that $m > k$, then the k names are used for the first k symbols. The variable names for the remaining $m - k$ symbols are the same as the corresponding symbol names. If $m < k$, then the extra $k - m$ names are ignored.

If you do not specify the ADJSAMPLEVAR= option, if you specify only one adjusted severity symbol in the ADJUSTEDSEVERITY= option, and if you specify the [ONEADJSEVCOMPAT](#) option, then “_AGGADJSEV_” is used by default.

SAMPLEVAR=*variable-name*

specifies the name of the variable to contain the simulated sample in the OUT= data table. If you do not specify this option, then “_AGGSEV_” is used by default.

Further, you can request that the perturbed samples be written to the OUT= data table by specifying the following *out-option*:

PERTURBOUT

writes all the perturbed samples to the OUT= data table. Each perturbed sample is identified by the value of the _DRAWID_ variable in the OUT= data table. A value of 0 indicates an unperturbed sample.

Separate compound distribution samples are generated for each combination of specified severity and frequency models. The _SEVERITYMODEL_ and _COUNTMODEL_ columns in the OUT= data table identify the severity and frequency models, respectively, that are used to generate the sample in the SAMPLEVAR= variable and ADJSAMPLEVAR= variables.

For more information about the variables in the OUTSAMPLE= data table, see the section “[OUTSAMPLE= Data Table](#)” on page 170.

OUTSUM Statement

OUTSUM OUT=*CAS-libref.data-table statistic-keyword* <=*variable-name*> <... *statistic-keyword* <=*variable-name*>> <*outsum-options*> ;

The OUTSUM statement enables you to specify the data table in which PROC CCDM writes the summary statistics of the compound distribution samples.

If you specify more than one OUTSUM statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*CAS-libref.data-table*

OUTSUM=*CAS-libref.data-table*

specifies the output data table that contains the summary statistics of each of the simulated compound distribution samples. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 96. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the DATA= option.

You can control the summary statistics that appear in this data table by specifying different *statistic-keywords* and *outsum-options*.

You can request that one or more predefined statistics of the compound distribution sample be written to the OUTSUM= data table. For each specification of the form *statistic-keyword* <=*variable-name*>, the statistic that is specified by the *statistic-keyword* is written to a variable named *variable-name*. If you do not specify the *variable-name*, then the statistic is written to a variable named *statistic-keyword*. You can specify the following *statistic-keywords*:

KURTOSIS

KURT

specifies the kurtosis of the compound distribution sample.

MEAN

specifies the mean of the compound distribution sample.

P01

specifies the 1st percentile of the compound distribution sample.

P05

specifies the 5th percentile of the compound distribution sample.

P10

specifies the 10th percentile of the compound distribution sample.

P25

Q1

specifies the lower or 1st quartile (the 25th percentile) of the compound distribution sample.

P50

MEDIAN

Q2

specifies the median (the 50th percentile) of the compound distribution sample.

P75

Q3

specifies the upper or 3rd quartile (the 75th percentile) of the compound distribution sample.

P90

specifies the 90th percentile of the compound distribution sample.

P95

specifies the 95th percentile of the compound distribution sample.

P99

specifies the 99th percentile of the compound distribution sample.

P99_5

P995

specifies the 99.5th percentile of the compound distribution sample.

QRANGE

specifies the interquartile range ($Q3 - Q1$) of the compound distribution sample.

SKEWNESS

SKEW

specifies the skewness of the compound distribution sample.

STDDEV**STD**

specifies the standard deviation of the compound distribution sample.

All percentiles are computed by using the method that you specify in the **PCTLDEF=** option in the PROC CCDDM statement. You can also request additional percentiles to be reported in the OUTSUM= data table by specifying the following *outsum-options*:

PCTLPTS=percentile-list

specifies one or more percentiles that you want to be computed and written to the OUTSUM= data table. This option is useful if you need to request percentiles that are not available in the preceding list of *statistic-keyword* values. Each percentile value must belong to the (0,100) open interval. The *percentile-list* is a comma-separated list of numbers. You can also use a list notation of the form “< number1 > to < number2 > by < increment >”. For example, the following two options are equivalent:

```
pctlpts=10, 20, 99.6, 99.7, 99.8, 99.9
pctlpts=10, 20, 99.6 to 99.9 by 0.1
```

You can specify the name of the variable for a particular percentile value in the PCTLNAME= option.

PCTLNAME=percentile-variable-name-list

specifies the names of the variables that contain the estimates of the percentiles that you request by using the PCTLPTS= option.

If you specify the PCTLNAME= option, you must specify it as the last option in the OUTSUM statement.

If you do not specify the PCTLNAME= option, then each percentile value t in the list of values in the PCTLPTS= option is written to the variable named “Pt,” where the decimal point in t , if any, is replaced by an underscore.

The *percentile-variable-name-list* is a space-separated list of names. You can also use a shortcut notation of <prefix> m –<prefix> n for two integers m and n ($m < n$) to generate the following list of names: <prefix> m , <prefix> $m + 1$, ..., and <prefix> n . For example, the following two options are equivalent:

```
pctlname=p1 p2 pc5 pc6 pc7 pc8 pc9 pc10
pctlname=p1 p2 pc5-pc10
```

The name in j th position of the expanded name list of the PCTLNAME= option is used to create a variable for a percentile value in the j th position of the expanded value list of the PCTLPTS= option. If you specify k_n names in the PCTLNAME= option and k_v percentile values in the PCTLPTS= option, and if $k_n < k_v$, then the first k_n percentiles are written to the variables that you specify. The remaining $k_v - k_n$ percentiles are written to the variables that have the name of the form P_t , where t is the text representation of the percentile value that is formed by retaining at most PCTLNDEC= digits after the decimal point and replacing the decimal point with an underscore (‘_’). For example, assume that you specify the following options:

```
pctlpts=10, 20, 99.3 to 99.5 by 0.1, 99.995
pctlname=pten ptwenty ninenine3-ninenine5
```

Then PROC CCDDM writes the 10th and 20th percentiles to the variables `pten` and `ptwenty`, respectively; the 99.3rd through 99.5th percentiles to the variables `ninenine3`, `ninenine4`, and `ninenine5`, respectively; and the remaining 99.995th percentile to the variable `P99_995`.

If a percentile value in the `PCTLPTS=` option matches a percentile value that is implied by one of the predefined percentile statistics and you specify the corresponding *statistic-keyword*, then the variable name that is implied by the *statistic-keyword*≤*variable-name* specification takes precedence over the name that you specify in the `PCTLNAME=` option. For example, assume that you specify the predefined percentile statistic of `P95`, as in the following `OUTSUM` statement:

```
outsum out=mypctls p95=ninetyfifth
      pctlpts=95 to 99 by 1 pctlname=pct95-pct99;
```

Then the 95th percentile is written to the variable `ninetyfifth` instead of the variable `pct95` that the `PCTLNAME=` option implies.

PCTLNDEC=integer-value

specifies the maximum number of decimal places to use while creating the names of the variables for the percentile values in the `PCTLPTS=` option. By default, `PCTLNDEC=3`. For example, for a percentile value of 99.9995, PROC CCDDM creates a variable named `P99_999` by default, but if you specify `PCTLNDEC=4`, then the variable is named `P99_9995`.

The `PCTLNDEC=` option is used only for percentile values for which you do not specify a name in the `PCTLNAME=` option.

Note that all variable names in the `OUTSUM=` data table have a limit of 32 characters. If a name exceeds that limit, then it is truncated to the first 32 characters. For more information about the variables in the `OUTSUM=` data table, see the section “[OUTSUM= Data Table](#)” on page 171.

SEVERITYMODEL Statement

```
SEVERITYMODEL severity-model-list < / definition-option > ;
```

The `SEVERITYMODEL` statement specifies one or more severity distribution models that you want to use in simulating a compound distribution sample. The *severity-model-list* is a space-separated list of names of severity models that you would use with PROC SEVSELECT’s `DIST` statement. The `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store must contain all the severity models in the list. If you specify the `SEVERITYEST=` data table and you specify a name that does not appear in the `_MODEL_` column of the `SEVERITYEST=` data table, then that name is ignored. Similarly, if you specify the `SEVERITYSTORE=` item store and a severity model by that name does not appear in the item store, then that name is ignored.

You can specify the following *definition-option* in the `SEVERITYMODEL` statement:

DEFINITIONSTABLE="CAS-table-name" <(CASLIB="caslib")>

SEVDEF="CAS-table-name" <(CASLIB="caslib")>

specifies a data table on the CAS server that contains the severity distribution function definitions. If you do not specify the CASLIB= option, then PROC CCDM assumes that the table is in the caslib that is active in the current CAS session. To specify a different caslib, use the CASLIB= option.

If you do not specify the DEFINITIONSTABLE= option, then PROC CCDM loads the severity function definitions from the library Sashelp.Svrtldist and the libraries that you specify in the CMPLIB= SAS system option and creates a temporary CAS table named cdm_svrtldist in the current CAS session. This process requires some time and effort, which you can avoid by specifying the DEFINITIONSTABLE= option. You can use the CAS table that the SEVSELECT procedure creates when it fits the severity models. For more information, see the OUTFUNCDEF= option in the DIST statement of the SEVSELECT procedure.

If you specify more than one SEVERITYMODEL statement, only the first one is used.

If you do not specify a SEVERITYMODEL statement, then this is equivalent to specifying all the severity models that appear in the SEVERITYEST= data table or the SEVERITYSTORE= item store.

A compound distribution sample is generated for each severity model by compounding that severity model with the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement, or the external frequency model that is encoded by the COUNT= variable that you specify in the EXTERNALCOUNTS statement.

SIMULATEDSYMBOL Statement

SIMULATEDSYMBOL *symbol* ~ *distribution-name*(*parameter-value-list*) <... *symbol* ~ *distribution-name*(*parameter-value-list*)> ;

The SIMULATEDSYMBOL statement specifies one or more symbols and their associated probability distributions that you want PROC CCDM to simulate. You must use these symbols in the programming statements that define one or more of the ADJUSTEDSEVERITY= symbols. Hence, this statement is ignored if you do not specify the ADJUSTEDSEVERITY= option.

Each simulated *symbol* is a random variable that follows a specific parametric probability distribution, which you specify by using the *distribution-name* followed by a *parameter-value-list* that is enclosed in parentheses. The list of supported distributions is shown in Table 5.2 and Table 5.3 in the section “Fixed Parameter-Value Distributions” on page 162. The *distribution-name* is one of the names that appear in the Names column of those tables, and *parameter-value-list* is a comma- or space-separated list of numbers that represent the values of the parameters in the order in which the parameters appear top to bottom in the Parameters column for the specified *distribution-name* in those tables. The SIMULATEDSYMBOL statement can contain multiple such symbol specifications separated by a space.

The simulated symbols are useful in applications where your adjusted severity depends on some random variables that you have estimated externally. For example, if you are simulating the pure premium by specifying the SIMULATIONMODE=PP option, and if your frequency is modeled as the average number of claims per exposure, then to estimate the loss that you expect to pay for multiple exposures, you need to multiply the pure premium by the exposure. Typically, exposure is a random variable that you estimate externally by using a regression or time series forecasting technique. The SIMULATEDSYMBOL statement enables you to easily incorporate such stochastic variables in the simulation process. For example, if exposure

denotes the exposure and if you have estimated it to follow a normal distribution $\mathcal{N}(\mu = 100, \sigma = 7.5)$, then you can specify the following statements to instruct PROC CCDM to estimate the distribution of the paid loss (paidLoss) that accounts for the randomness of exposure:

```
proc ccdm severitystore=mycas.sevstore countstore=mycas.cntstore
    simulationmode=pp adjustedseverity=paidLoss;
    simulatedsymbol exposure ~ normal(100,7.5);
    paidLoss = _sev_ * exposure;
run;
```

Programming Statements

In PROC CCDM, you can use a series of programming statements that use variables in the DATA= data table to define multiple adjustments to an individual severity value. Each adjusted severity value is aggregated to form a separate adjusted compound distribution sample.

The programming statements are executed for each simulated individual severity value. The observation of the input data table that is used to evaluate the programming statements is determined by the simulation process that is described in the section “[Simulation Processes for Collective Risk Simulation Mode](#)” on page 139.

For more information, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 151.

Details: CCDM Procedure

Specifying Scenario Data in the DATA= Data Table

A scenario represents a state of the world for which you want to estimate the distribution of aggregate losses. The state consists of one or more entities that generate the loss events. For example, an entity might be an individual who has an insurance policy or an organization that has a workers’ compensation policy. Each entity has some characteristics of its own and some external factors that affect the frequency with which it generates the losses and the severity of each loss. For example, characteristics of an individual automobile insurance policyholder can include various demographics of the individual and various features of the automobile. Characteristics of an organization that has a workers’ compensation policy can be the number of employees, revenue, ratio of temporary to permanent employees, and so on. The organization can also be affected by external macroeconomic factors such as GDP and unemployment rate of the country where it operates and factors that affect its industry. You need to quantify and specify all these characteristics as external factors (regressors) when you fit severity and frequency models.

You should specify all the information about a scenario in the DATA= data table that you specify in the PROC CCDM statement. Each observation in the DATA= data table encodes the characteristics of an entity. For proper simulation of severities, you must specify in the DATA= data table all the characteristics that you use as regressors in the severity scale regression models. When you use the COUNTSTORE= option to specify the frequency model, you must specify in the DATA= data table all the characteristics that you use as regressors in the frequency model in order to properly simulate the counts. All the regressors are expected to

have nonmissing values. If any of the regressors have a missing value in an observation, then that observation is ignored.

The information in the DATA= data table is interpreted as follows, based on whether you specify the EXTERNALCOUNTS statement:

- If you do not specify the EXTERNALCOUNTS statement, then all the observations in the data table form a scenario. The observations are used together to compute one random draw from the compound distribution. The total number of draws is equal to the value that you specify in the NREPLICATES= option. The simulation process is described in the section “[Simulation with Regressors and No External Counts](#)” on page 140 and illustrated using an example in the section “[Illustration of Aggregate Loss Simulation Process](#)” on page 141.
- If you specify the EXTERNALCOUNTS statement, then the DATA= data table is expected to contain multiple replications (draws) of the frequency counts that you simulate externally for a scenario. The DATA= data table must contain the COUNT= variable that you specify in the EXTERNALCOUNTS statement. The replications are identified by the observation number or the ID= variable that you specify in the EXTERNALCOUNTS statement. For each observation in a given replication, the COUNT= variable is expected to contain the count of losses that are generated by the entity associated with that observation. All the observations of a given replication are used together to compute one random draw from the compound distribution. The size of the compound distribution sample is equal to the number of distinct replications that you specify in the DATA= data table, multiplied by the value that you specify in the NREPLICATES= option. The simulation process is described in the section “[Simulation with External Counts](#)” on page 143 and illustrated using an example in the section “[Illustration of the Simulation Process with External Counts](#)” on page 143.

In both cases, an observation can also contain severity adjustment variables that you can use to adjust the severity of the losses generated by that entity, based on some policy rules. For more information about simulating the adjusted compound distribution sample, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 151.

If you specify severity and frequency models that contain no regression effects, and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you do not need to specify the DATA= data table. This case corresponds to a fixed scenario that is represented entirely by the distribution parameters of the models.

Simulation Processes for Collective Risk Simulation Mode

This section describes the simulation processes for the collective risk mode, which is the simulation mode that PROC CCDM uses when you do not specify the SIMULATIONMODE= option or when you specify SIMULATIONMODE=CR. For other simulation modes, see the section “[Simulation Processes for Pure Premium and Custom Simulation Modes](#)” on page 147.

PROC CCDM selects a simulation process based on whether you specify external counts or you request that PROC CCDM simulate the counts and on whether or not the severity or frequency models contain regression effects. The following sections describe the processes for the different possibilities.

Simulation with No Regressors and No External Counts

If you specify severity and frequency models that contain no regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution, *dist*. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. Use the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement to draw a value N from the count distribution, where N denotes the number of loss events that are expected to occur in the time period that is being simulated. Adjust N to conform to the upper limit by setting it equal to $\min(N, N_{\max})$, where $N_{\max}$ is the value of the MAXCOUNTDRAW= option.
2. Use the parameter estimates of the severity distribution *dist* (which are read from the SEVERITYEST= data table or the SEVERITYSTORE= item store that you specify) to draw N severity values, X_j ($j = 1, \dots, N$). (For more information about how PROC CCDM draws an individual severity value, see the section “Making a Random Draw from a Severity Distribution” on page 146.)
3. Add the N severity values that are drawn in step 2 to compute one sample point, S , of the compound distribution sample as

$$S = \sum_{j=1}^N X_j$$

Although it is more common to fit a frequency model that contains regressors, the CNTSELECT procedure enables you to fit a frequency model that does not contain regressors. If you do not specify any regressors in the MODEL statement of PROC CNTSELECT, then it fits a model that contains only an intercept.

Simulation with Regressors and No External Counts

If the severity or frequency models contain regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you must specify a DATA= data table to provide values of the regression variables, which together represent a scenario for which you want to simulate the CDM. In this case, PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution.

Note that you are performing scenario analysis when regression effects are present. Let K denote the number of observations that form the scenario. This is the number of observations either in the current BY group or in the entire DATA= data table if you do not specify the BY statement. If $K > 1$, then you are modeling the scenario for a group of entities. If $K = 1$, then you are modeling the scenario for one entity.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. For each observation k ($k = 1, \dots, K$), draw a count N_k from the frequency model, where N_k denotes the number of loss events that are expected to be generated by entity k in the time period that is being simulated. If you specify the COUNTSTORE= item store, then draw N_k from the frequency model whose parameters are determined by the frequency regressors in observation k . If you specify the COUNTMODEL statement, the frequency model does not depend on the regressors, so draw N_k by using the information in the COUNTMODEL statement. Adjust N_k to conform to the upper limit by setting it equal to $\min(N_k, N_{\max})$, where $N_{\max}$ is the value of the MAXCOUNTDRAW= option.
2. Add counts from all observations to compute $N = \sum_{k=1}^K N_k$, which is the total number of loss events that are expected to occur in the time period that is being simulated.
3. Make N random draws from the severity distribution and add them to generate one sample point of the compound distribution sample. If you specify a scale regression model for the severity distribution, then use the values of severity regressors in the k th observation to compute the scale parameter of the severity distribution and make N_k severity draws from that distribution. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.)

If you specify the BY statement, then a separate sample of size M is created for each BY group in the DATA= data table.

Illustration of Aggregate Loss Simulation Process

As an illustration of the simulation process, consider a very simple example of analyzing the distribution of an aggregate loss that is incurred by a set of policyholders of an automobile insurance company in a period of one year. It is postulated that the frequency and severity distributions depend on three variables: Age, Gender (1: female, 2: male), and CarType (1: sedan, 2: sport utility vehicle). So these variables are used as regressors while you fit the count model and severity scale regression model by using the CNTSELECT and SEVSELECT procedures, respectively. Now, consider that you want to use the fitted frequency and severity models to estimate the distribution of the aggregate loss that is incurred by a set of five policyholders. The characteristics of the five policyholders are encoded in a data table named mycas.Scenario that has the following contents:

Obs	age	gender	carType
1	30	2	1
2	25	1	2
3	45	2	2
4	33	1	1
5	50	1	1

The column Obs contains the observation number. It is shown only for the purpose of illustration. It does not need to be present in the data table. The following PROC CCDM step simulates the scenario in the data table mycas.Scenario:

```
proc ccdm data=mycas.scenario
    severityest=<severity parameter estimates data table>
    countstore=<count model store> nreplicates=<sample size>;
    severitymodel <severity distribution name(s)>;
run;
```

The following process generates a sample from the aggregate loss distribution for the scenario in the data table mycas.Scenario:

1. Use the values Age=30, Gender=2, and CarType=1 in the first observation to draw a count from the count distribution. Let that count be 2. Repeat the process for the remaining four observations. Let the counts be as shown in the Count column in the following table:

Obs	age	gender	carType	count
1	30	2	1	2
2	25	1	2	1
3	45	2	2	2
4	33	1	1	3
5	50	1	1	0

The Count column is shown for illustration only; it is not added as a variable to the DATA= data table.

2. The simulated counts from all the observations are added together to get a value of $N = 8$. This means that for this particular sample point, you expect a total of eight loss events in a year from these five policyholders.
3. For the first observation, the scale parameter of the severity distribution is computed by using the values Age=30, Gender=2, and CarType=1. That value of the scale parameter is used together with estimates of the other parameters from the SEVERITYEST= data table to make two draws from the severity distribution. Each draw simulates the magnitude of the loss that is expected from the first policyholder. The process is repeated for the remaining four policyholders. The fifth policyholder does not generate any loss event for this particular sample point, so no severity draws are made by using the fifth observation. Let the severity draws, rounded to integers for convenience, be as shown in the _SEV_ column in the following table:

Obs	age	gender	carType	count	_sev_		
1	30	2	1	2	350	2100	
2	25	1	2	1	4500		
3	45	2	2	2	700	4300	
4	33	1	1	3	600	1500	950
5	50	1	1	0			

The _SEV_ column is shown for illustration only; it is not added as a variable to the DATA= data table.

PROC CCDM adds the severity values of the eight draws to compute an aggregate loss value of 15,000. After recording this amount in the sample, the process returns to step 1 to compute the next point in the aggregate loss sample. For example, in the second iteration, the count distribution of each policyholder might generate one loss event, for a total of five loss events, and the five severity draws from the severity distributions that govern each policyholder might add up to 5,000. Then, the value of 5,000 is recorded as the second point in the aggregate loss sample. The process continues until M aggregate loss sample points are simulated, where the M is the value that you specify in the NREPLICATES= option.

Simulation with External Counts

If you specify externally simulated counts by using the `EXTERNALCOUNTS` statement, then each replication in the input data table represents the loss events that are generated by an entity. An entity can be an individual or organization for which you want to estimate the compound distribution. If an entity has any characteristics that are used as external factors (regressors) in developing the severity scale regression model, then you must specify the values of those factors in the `DATA=` data table. If you specify the `ID=` variable, then multiple observations for the same replication ID represent different entities in a group for which you are simulating the CDM.

This section describes the simulation process that `PROC CCDM` uses in the presence of externally simulated counts.

The process is described for one severity distribution. If you specify multiple severity distributions in the `SEVERITYMODEL` statement, then the process is repeated for each specified distribution.

Let there be M distinct replications in the current `BY` group of the `DATA=` data table or in the entire `DATA=` data table if you do not specify the `BY` statement. A replication is identified by either the observation number or the value of the `ID=` variable that you specify in the `EXTERNALCOUNTS` statement.

For each of the M values of the replication identifier, the following steps are executed R times, where R is the value of the `NREPLICATES=` option:

1. If there are K ($K \geq 1$) observations for the current value of the replication identifier, then for each observation k ($k = 1, \dots, K$), set N_k equal to the value of the `COUNT=` variable in that observation. Adjust N_k to conform to the upper limit by setting it equal to $\min(N_k, N_{\max})$, where $N_{\max}$ is the value of the `MAXCOUNTDRAW=` option.
2. Compute the total number of losses N across all observations of the replication as $N = \sum_{k=1}^K N_k$.
3. Make N random draws from the severity distribution and add them to generate one sample point of the compound distribution sample. If you specify a scale regression model for the severity distribution, then use the values of severity regressors in the k th observation to compute the scale parameter of the severity distribution and make N_k severity draws from that distribution. (For more information about how `PROC CCDM` makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.)

This process generates a compound distribution sample of size $M \times R$. If you specify the `BY` statement, then a separate sample of size $M \times R$ is created for each `BY` group in the `DATA=` data table.

Illustration of the Simulation Process with External Counts

To illustrate the simulation process, consider the following simple example. In this example, your severity model does not contain any regressors. An example that uses a severity scale regression model is illustrated later. Assume that you have made 10 random draws from an external count model and recorded them in the variable `ExtCount` of a data table named `mycas.Counts1`, as follows:

Obs	extCount
1	3
2	2
3	0
4	1

5	3
6	4
7	1
8	2
9	0
10	5

Because the data table does not contain an ID= variable, the observation number that is shown in the Obs column acts as the replicate identifier. The following PROC CCDM step simulates an aggregate loss sample by using the data table mycas.Counts1:

```
proc ccdm data=mycas.counts1 nreplicates=5
    severityest=<severity parameter estimates data table>;
    severitymodel <severity distribution name(s)>;
    externalcounts count=extCount;
run;
```

The simulation process works as follows:

1. For the first replication, which is associated with the first observation, three severity values are drawn from the severity distribution by using the parameter estimates that you specify in the SEVERITYEST= data table. If the severity values are 150, 500, and 320, then their sum of 970 is recorded as the first point of the aggregate loss sample. Because the value of the NREPLICATES= option is 5, this process of drawing three severity values and adding them together to form a point of the aggregate loss sample is repeated four more times to generate a total of five sample points that correspond to the first observation.
2. For the second replication, two severity values are drawn from the severity distribution. If the severity values are 450 and 100, then their sum of 550 is recorded as a point of the aggregate loss sample. This process of drawing two severity values and adding them together to form a point of the aggregate loss sample is repeated four more times to generate a total of five sample points that correspond to the second observation.
3. The process continues until all the replications, which are observations in this case, are exhausted.

The process results in an aggregate loss sample of size 50, which is equal to the number of replications in the data table (10) multiplied by the value of the NREPLICATES= option (5).

Now, consider an example in which the severity models in the SEVERITYEST= data table are scale regression models. In this case, the severity distribution that is used for drawing the severity value is determined by the values of regressors in the observation that is being processed. Consider that you want to simulate the aggregate loss that one policyholder incurs and you have recorded, in the variable ExtCount, the results of 10 random draws from an external count model. The DATA= data table has the following contents:

Obs	age	gender	carType	extCount
1	30	2	1	5
2	30	2	1	2
3	30	2	1	0
4	30	2	1	1
5	30	2	1	3
6	30	2	1	4
7	30	2	1	1

8	30	2	1	2
9	30	2	1	0
10	30	2	1	5

The simulation process in this case is the same as the process in the previous case of no regressors, except that the severity distribution that PROC CCDM uses for drawing the severity values has a scale parameter that is determined by the values of the regressors Age, Gender, and CarType in the observation that is being processed. In this particular example, all observations have the same value for all regressors, indicating that you are modeling a scenario in which the characteristics of the policyholder do not change during the time for which you have simulated the number of events. You can also model a scenario in which the characteristics of the policyholder change by recording those changes in the values of the appropriate regressors.

Extending this example further, consider that you want to analyze the distribution of the aggregate loss that is incurred by a group of policyholders, as in the example in the section “[Illustration of Aggregate Loss Simulation Process](#)” on page 141. Let the data table mycas.Counts2 record multiple replications of the number of losses that each policyholder might generate. The contents of the data table mycas.Counts2 are as follows:

Obs	replicateId	age	gender	carType	extCount
1	1	30	2	1	2
2	1	25	1	2	1
3	1	45	2	2	3
4	1	33	1	1	5
5	1	50	1	1	1
6	2	30	2	1	3
7	2	25	1	2	2
8	1	45	2	2	0
9	2	33	1	1	4
10	2	50	1	1	1

The variable ReplicateId records the identifier for the replication. Each replication contains multiple observations, such that each observation represents one of the policyholders that you are analyzing. For simplicity, only the first two replications are shown here.

The following PROC CCDM step simulates an aggregate loss sample by using the data table mycas.Counts2:

```
proc ccdm data=mycas.counts2 nreplicates=3
    severityest=<severity parameter estimates data table>;
    severitymodel <severity distribution name(s)>;
    externalcounts count=extCount id=replicateId;
    output out=aggloss samplevar=totalLoss;
run;
```

When you specify an ID= variable in the EXTERNALCOUNTS statement, the DATA= data table must be partitioned by the ID= variable as noted in the description of the ID= option.

The simulation process works as follows:

1. First, the five observations of the first replication (ReplicateId=1) are analyzed. For the first observation (Obs=1), the scale parameter of the severity distribution is computed by using the values Age=30, Gender=2, and CarType=1. That value of the scale parameter is used together with estimates of the other parameters from the SEVERITYEST= data table to make two draws from the severity distribution. Next, the regressor values of the second observation are used to compute the scale parameter of the

severity distribution, which is used to make one severity draw. The process continues such that the regressor values in the third, fourth, and fifth observations are used to select which severity distribution to make three, five, and one draws from, respectively. Let the severity values that are drawn from the observations of this replication be as shown in the `_SEV_` column in the following table, where the `_SEV_` column is shown for illustration only; it is not added as a variable to the `DATA=` data table.

Obs	replicateId	age	gender	carType	extCount	_sev_
1	1	30	2	1	2	700 500
2	1	25	1	2	1	5000
3	1	45	2	2	3	900 1400 300
4	1	33	1	1	5	350 2000 150 800 600
5	1	50	1	1	1	250

The values of all 12 severity draws are added together to compute and record the value of 12,950 as the first point of the aggregate loss sample. Because you specify `NREPLICATES=3` in the `PROC CCDM` step, this process of making 12 severity draws from the respective observations is repeated two more times to generate a total of three sample points for the first replication.

2. The five observations of the second replication (`ReplicateId=2`) are analyzed next to draw three, two, four, and one severity values from the severity distributions, with scale parameters that are determined by the regressor values in the sixth, seventh, ninth, and tenth observations, respectively. The 10 severity values are added together to form a point of the aggregate loss sample. This process of making 10 severity draws from the respective observations is repeated two more times to generate a total of three sample points for the second replication.

If your data table `mycas.Counts2` contains 10,000 distinct values of `ReplicateId`, then 30,000 observations are written to the data table `mycas.AgglLoss` that you specify in the `OUTPUT` statement of the preceding `PROC CCDM` step. Because you specify `SAMPLEVAR=TotalLoss` in the `OUTPUT` statement, the aggregate loss sample is available in the `TotalLoss` column of the data table `mycas.AgglLoss`.

Making a Random Draw from a Severity Distribution

`PROC CCDM` uses a simple process to make a random draw from the severity distribution. It first makes a draw from the uniform distribution. If this value is denoted by p , then the corresponding severity draw is computed as $x = F^{-1}(p)$, where F denotes the cumulative distribution function (CDF) of the severity distribution.

For a severity distribution *dist*, `PROC CCDM` computes the inverse of the CDF (that is, F^{-1}) by evaluating the `dist_QUANTILE` function. If `PROC CCDM` does not find the definition of the `dist_QUANTILE` function in the `DEFINITIONSTABLE=` table or in the libraries that you specify in the `CMPLIB=` system option, it uses a numerical inversion algorithm to invert the CDF function at p . This algorithm yields an approximate inverse and can be slow because it makes multiple calls to the `dist_CDF` or `dist_LOGCDF` function of the distribution. So it is recommended that you define the `dist_QUANTILE` function whenever possible for any custom severity distribution that you define for use with `PROC CCDM`. For more information about defining custom severity distributions, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895 in Chapter 15, “[The SEVSELECT Procedure](#).” The `dist_QUANTILE` function is already defined for all severity distributions in the library `Sashelp.Svrtldist`.

To evaluate the `dist_QUANTILE` or `dist_CDF` function, `PROC CCDM` uses the distribution parameters that it reads from the `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store. If the severity model

contains regression effects, then PROC CCDDM computes the scale parameter of the severity distribution by using the base value of the scale parameter and the values of the regression effects.

Making a Random Draw from a Truncated Severity Distribution

PROC CCDDM assumes that the severity distribution that you specify in the `SEVERITYEST=` data table or the `SEVERITYSTORE=` item store is an untruncated distribution—that is, a distribution that has the full support. For severity distributions that have a scale parameter, full support is $[0, \infty)$, and for severity distributions that have a log-scale parameter, full support is $(-\infty, \infty)$. If you want PROC CCDDM to make draws from a truncated distribution, which is especially useful in insurance applications, then you can specify the `LEFTTRUNCATION=` or `RIGHTTRUNCATION=` options in the PROC CCDDM statement.

Let L and R denote the left-truncation threshold and right-truncation limit that you specify in the `LEFTTRUNCATION=` and `RIGHTTRUNCATION=` options, respectively. Let F and $F_{L,R}$ denote the cumulative distribution functions (CDF) of the untruncated and truncated severity distributions, respectively. Not specifying the `LEFTTRUNCATION=` option is equivalent to specifying $L = F^{-1}(0)$ —that is, $F(L) = 0$. Not specifying the `RIGHTTRUNCATION=` option is equivalent to specifying $R = F^{-1}(1)$ —that is, $F(R) = 1$.

PROC CCDDM makes a random draw from the truncated severity distribution by using the following steps:

1. Draw a value p_t from the uniform distribution. This is the CDF of the truncated distribution.
2. Compute p , which represents the CDF of the untruncated distribution, as

$$p = p_t(F(R) - F(L)) + F(L)$$

3. Compute the severity value x by inverting the CDF of the untruncated distribution as $x = F^{-1}(p)$. The value of x that is computed in this manner is essentially a random draw from $F_{L,R}$ —that is, $x = F_{L,R}^{-1}(p_t)$. The inversion of the untruncated CDF is described in the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.

Simulation Processes for Pure Premium and Custom Simulation Modes

This section describes the simulation processes for pure premium mode (when you specify `SIMULATIONMODE=PP`) and custom mode (when you specify `SIMULATIONMODE=CUSTOM`).

Like the simulation processes for the collective risk mode, which are described in the section “[Simulation Processes for Collective Risk Simulation Mode](#)” on page 139, the simulation processes for pure premium and custom modes also depend on whether you specify external counts or you request that PROC CCDDM simulate the counts and on whether or not the severity or frequency models contain regression effects. The following sections describe the processes for the different possibilities.

Simulation with No Regressors and No External Counts

If you specify severity and frequency models that contain no regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution, *dist*. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. Use the frequency model that you specify in the COUNTSTORE= item store or the COUNTMODEL statement to draw a value N from the count distribution.
2. Use the parameter estimates of the severity distribution *dist* (which are read from the SEVERITYEST= data table or the SEVERITYSTORE= item store that you specify) to draw a severity value X . (For more information about how PROC CCDM draws an individual severity value, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Set the _FREQ_ and _SEV_ placeholder symbols equal to N and X , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 151.)
 - b) Execute your severity adjustment program.
 - c) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.
 - d) Set the sample point S^d of the d th adjusted aggregate loss sample equal to X_k^d ($d = 1, \dots, D$).
 - For SIMULATIONMODE=PUREPREMIUM, multiply the frequency and severity values to compute the sample point S of the unadjusted aggregate loss sample as $S = NX$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.
 - Compute S^d as $S^d = NX^d$ ($d = 1, \dots, D$).

Simulation with Regressors and No External Counts

If the severity or frequency models contain regression effects and if you do not specify externally simulated counts in the EXTERNALCOUNTS statement, then you must specify a DATA= data table to provide values of the regression variables, which together represent a scenario for which you want to simulate the CDM. In this case, PROC CCDM uses the simulation process that this section describes.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

Note that you are performing scenario analysis when regression effects are present. Let K denote the number of observations that form the scenario. This is the number of observations either in the current BY group or in the entire DATA= data table if you do not specify the BY statement. If $K > 1$, then you are modeling the scenario for a group of entities. If $K = 1$, then you are modeling the scenario for one entity.

The following steps are repeated M times to generate a compound distribution sample of size M , where M is the value of the NREPLICATES= option:

1. For each observation k ($k = 1, \dots, K$), draw a value N_k from the frequency model. If you specify the COUNTSTORE= item store, then the parameters of the frequency model are determined by the frequency regressors in observation k . If you specify the COUNTMODEL statement, the frequency model does not depend on the regressors, so N_k is drawn by using the information that you specify in the COUNTMODEL statement.
2. For each observation k ($k = 1, \dots, K$), draw a severity value X_k from the severity model. If you specify a scale regression model for the severity distribution, then the scale parameter of the severity distribution is determined by the values of the severity regressors in the k th observation. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Execute your severity adjustment program K times, once per observation. To execute the program for the k th observation, set the _FREQ_ and _SEV_ placeholder symbols equal to N_k and X_k , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 151.)
 - b) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.
 - c) After all K observations are processed, compute the sample point S^d of the d th adjusted aggregate loss sample as $S^d = \sum_{k=1}^K X_k^d$ ($d = 1, \dots, D$).
 - For SIMULATIONMODE=PUREPREMIUM, compute the sample point S of the unadjusted aggregate loss sample as $S = \sum_{k=1}^K N_k X_k$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.

- Compute S^d as $S^d = \sum_{k=1}^K N_k X_k^d$ ($d = 1, \dots, D$).

If you specify the BY statement, then a separate sample of size M is created for each BY group in the DATA= data table.

Simulation with External Counts

If you specify externally simulated counts by using the EXTERNALCOUNTS statement, then each replication in the input data table represents the loss events that are generated by an entity. An entity can be an individual or organization for which you want to estimate the compound distribution. If an entity has any characteristics that are used as external factors (regressors) in developing the severity scale regression model, then you must specify the values of those factors in the DATA= data table. If you specify the ID= variable, then multiple observations for the same replication ID represent different entities in a group for which you are simulating the CDM.

This section describes the simulation process that PROC CCDM uses in the presence of externally simulated counts.

The process is described for one severity distribution. If you specify multiple severity distributions in the SEVERITYMODEL statement, then the process is repeated for each specified distribution. Let D denote the number of adjusted severity symbols that you specify in the ADJUSTEDSEVERITY= option.

Suppose there are M distinct replications in the current BY group of the DATA= data table or in the entire DATA= data table if you do not specify the BY statement. A replication is identified either by the observation number or by the value of the ID= variable that you specify in the EXTERNALCOUNTS statement.

For each of the M values of the replication identifier, the following steps are executed R times, where R is the value of the NREPLICATES= option:

1. Read the value of the COUNT= variable, N_k , for each observation k ($k = 1, \dots, K$), where K denotes the number of observations in the current replication—that is, the number of observations in which the replication identifier has the same value.
2. For each observation k ($k = 1, \dots, K$), draw a severity value X_k from the severity model. If you specify a scale regression model for the severity distribution, then the scale parameter of the severity distribution is determined by the values of the severity regressors in the k th observation. (For more information about how PROC CCDM makes a random severity draw, see the section “[Making a Random Draw from a Severity Distribution](#)” on page 146.)
3. Compute one sample point of different aggregate loss samples as follows:
 - For SIMULATIONMODE=CUSTOM, do not compute the unadjusted aggregate loss sample point. Because D must be nonzero for this mode, compute D sample points S^d of the D adjusted aggregate loss samples as follows:
 - a) Execute your severity adjustment program K times, once per observation. To execute the program for the k th observation, set the _FREQ_ and _SEV_ placeholder symbols equal to N_k and X_k , respectively. (For more information about the placeholder symbols, see the section “[Simulation of Adjusted Compound Distribution Sample](#)” on page 151.)
 - b) Retrieve the adjusted severity values X_k^d ($d = 1, \dots, D$) that your adjustment program computes.

- c) After all K observations are processed, compute the sample point S^d of the d th adjusted aggregate loss sample as $S^d = \sum_{k=1}^K X_k^d$ ($d = 1, \dots, D$).
- For SIMULATIONMODE=PUREPREMIUM, compute the sample point S of the unadjusted aggregate loss sample as $S = \sum_{k=1}^K N_k X_k$. If $D > 0$, then follow the same steps as the custom mode to compute D sample points S^d of the D adjusted aggregate loss samples, except for the following two differences:
 - Do not fill the _FREQ_ placeholder symbol, because it is valid only in the custom mode.
 - Compute S^d as $S^d = \sum_{k=1}^K N_k X_k^d$ ($d = 1, \dots, D$).

This process generates a compound distribution sample of size $M \times R$. If you specify the BY statement, then a separate sample of size $M \times R$ is created for each BY group in the DATA= data table.

Simulation of Adjusted Compound Distribution Sample

If you specify a list of D adjusted severity symbols in the ADJUSTEDSEVERITY= option and specify programming statements that compute those symbols, then a separate compound distribution sample is generated for each adjusted severity symbol. The set of all programming statements that you specify in the PROC CCDM step is called the severity adjustment program.

Formally, the severity adjustment program is expected to implement D adjustment functions for all D adjusted severity symbols. The adjustment function for the d th symbol is denoted by f^d ($d = 1, \dots, D$). The function f^d uses the unadjusted severity value, X_j along with other values, some of which depend on the simulation mode, to compute and return an adjusted severity value, X_j^d . The information that is available to compute X_j^d depends on the simulation mode as follows:

- For the collective risk simulation mode, if N denotes the number of loss events that are simulated for the current replication of the simulation process, then for the severity draw, X_j , of the j th loss event ($j = 1, \dots, N$), the adjusted severity value for the d th symbol is

$$X_j^d = f^d(X_j, S_{j-1}, S_{j-1}^d, \Psi_j)$$

where $S_{j-1} = \sum_{l=1}^{j-1} X_l$ is the aggregate unadjusted loss, $S_{j-1}^d = \sum_{l=1}^{j-1} X_l^d$ is the aggregate value of the d th adjusted severity symbol before the j th severity draw that generates X_j , and Ψ_j denotes the vector of values of the stochastic symbols that you specify in the SIMULATEDSYMBOL statement. The initial values of both S and S^d are set to 0—that is, $S_0 = 0$ and $S_0^d = 0$ ($d = 1, \dots, D$). Prior to executing your adjustment program, PROC CCDM makes a random draw for each stochastic symbol by using that symbol's probability distribution and parameter values. The aggregate adjusted loss for the replication is S_N^d , which is denoted by S^d for simplicity and is defined as

$$S^d = \sum_{j=1}^N X_j^d$$

- For pure premium and custom simulation modes, there is only one severity draw for every frequency draw, so the index j is always equal to 1.

- For pure premium mode, the adjusted severity value depends only on the unadjusted severity value and stochastic symbols as $X_1^d = f^d(X_1, \Psi_1)$ and the aggregate adjusted loss for the replication is $S^d = NX_1^d$.
- For custom simulation mode, you are free to combine the values of frequency, severity, and stochastic symbols in any manner to compute the adjusted severity value as $X_1^d = f^d(N, X_1, \Psi_1)$ and PROC CCDM sets the aggregate adjusted loss equal to the adjusted severity value as $S^d = X_1^d$.

In your severity adjustment program, you can use the following keywords as placeholder symbols for the input arguments of the function f^d :

_ADJSEVSUM d _

_CUMADJSEV d _

indicates the placeholder for S_{j-1}^d , the sum of adjusted severity values for the d th symbol that are computed by your program before X_j is generated and adjusted. PROC CCDM supplies this value to your program. For $d = 1$, you can use the symbol `_ADJSEVSUM_` (alias `_CUMADJSEV_`).

This placeholder is always meaningful for `SIMULATIONMODE=COLLECTIVERISK`, but for `SIMULATIONMODE=PUREPREMIUM` and `SIMULATIONMODE=CUSTOM`, it is meaningful only for a multi-entity scenario, which is described in the section “[Aggregate Adjusted Loss Simulation for a Multi-entity Scenario](#)” on page 153.

FREQ

indicates the placeholder for frequency N . PROC CCDM supplies this value to your program.

You can use this placeholder only when you specify `SIMULATIONMODE=CUSTOM`.

SEV

indicates the placeholder for X_j , the unadjusted severity value. PROC CCDM generates this value as described in the section “[Simulation with No Regressors and No External Counts](#)” on page 140 (step 2) or the section “[Simulation with Regressors and No External Counts](#)” on page 140 (step 3). PROC CCDM supplies this value to your program.

You can use this placeholder for all simulation modes.

SEVSUM

CUMSEV

indicates the placeholder for S_{j-1} , the sum of unadjusted severity values that PROC CCDM generates before X_j is generated. PROC CCDM supplies this value to your program.

This placeholder is always meaningful for `SIMULATIONMODE=COLLECTIVERISK`, but for `SIMULATIONMODE=PUREPREMIUM` and `SIMULATIONMODE=CUSTOM`, it is meaningful only for a multi-entity scenario, which is described in the section “[Aggregate Adjusted Loss Simulation for a Multi-entity Scenario](#)” on page 153.

Your severity adjustment program must assign the output of the d th function f^d to the d th symbol that you specify in the `ADJUSTEDSEVERITY=` option in the PROC CCDM statement. PROC CCDM uses the final assigned value of the d th symbol as the value of X_j^d .

You can use most DATA step statements and functions in your program. The DATA step file and the data table I/O statements (for example, INPUT, FILE, SET, and MERGE) are not available. However, some

functionality of the PUT statement is supported. For more information, see the section “PROC FCMP and DATA Step Differences” in *Base SAS Procedures Guide*.

The simulation process that generates the aggregate adjusted loss sample is identical to the process that is described in the section “Simulation with Regressors and No External Counts” on page 140 or the section “Simulation with External Counts” on page 143, except that after making each of the N severity draws, PROC CCDM executes your severity adjustment program to compute the adjusted severity symbols (X_j^d). All the N adjusted severity values are added together to compute S^d , which forms one point of the d th aggregate adjusted loss sample. The process is illustrated using an example in the section “Illustration of the Aggregate Adjusted Loss Simulation Process” on page 155.

Using Severity Adjustment Variables

If you do not specify the DATA= data table, then your ability to adjust the severity value is limited, because you can use only the current severity draw, sums of unadjusted and adjusted severity draws that are made before the current draw, and some constant numbers to encode your adjustment policy. That is sufficient if you want to estimate the distribution of aggregate adjusted loss for only one entity. However, if you are simulating a scenario that contains more than one entity, then it might be more useful if the adjustment policy depends on factors that are specific to each entity that you are simulating. To do that, you must specify the DATA= data table and encode such factors as *adjustment variables* in the DATA= data table. If A denotes the set of values of the adjustment variables, then each adjustment function f^d accepts the set A as one of its inputs, in addition to the other input values that it accepts according to the simulation mode. PROC CCDM reads the values of adjustment variables from the DATA= data table and supplies the set of those values (A) to your severity adjustment program. For an invocation of each f^d with an unadjusted severity value of X_j , the values in set A are read from the same observation that is used to simulate X_j .

All adjustment variables that you use in your program must be present in the DATA= data table. You must not use any keyword for a placeholder symbol as a name of any variable in the DATA= data table, whether the variable is a severity adjustment variable or a regressor in the frequency or severity model. Further, the following restrictions apply to the adjustment variables in your severity adjustment program:

- You can use only numeric-valued variables. This restriction also implies that you cannot use SAS functions or call routines that require character-valued arguments, unless you pass those arguments as constant (literal) strings or characters.
- You cannot use functions that create lagged versions of a variable. If you need lagged versions, then you can use a DATA step before the PROC CCDM step to add those versions to the input data table.

The use of adjustment variables is illustrated using an example in the section “Illustration of the Aggregate Adjusted Loss Simulation Process” on page 155.

Aggregate Adjusted Loss Simulation for a Multi-entity Scenario

If you are simulating a scenario that consists of multiple entities, then you can use some additional pieces of information in your severity adjustment program. Let the scenario consist of K entities, and let N_k denote the number of loss events that are incurred by the k th entity ($k = 1, \dots, K$) in the current iteration of the simulation process. The simulation mode determines the processing of multiple entities as follows:

- For the collective risk simulation mode, each value of N_k is adjusted to conform to the upper limit that is imposed by the value of the MAXCOUNTDRAW= option. The total number of severity draws that

need to be made is $N = \sum_{k=1}^K N_k$. The d th aggregate adjusted loss is now defined as

$$S^d = \sum_{k=1}^K \sum_{j=1}^{N_k} X_{k,j}^d$$

where $X_{k,j}^d$ is an adjusted severity value of the d th symbol for the k th entity in the j th draw ($j = 1, \dots, N_k$). The form of the adjustment function f^d that computes $X_{k,j}^d$ is

$$X_{k,j}^d = f^d(X_{k,j}, S_{k,j-1}, S_{k,j-1}^d, S_{n-1}, S_{n-1}^d, \Psi_{k,j}, A_k)$$

where $X_{k,j}$ is the value of the j th draw of unadjusted severity for the k th entity; $S_{k,j-1} = \sum_{l=1}^{j-1} X_{k,l}$ and $S_{k,j-1}^d = \sum_{l=1}^{j-1} X_{k,l}^d$ are the aggregate unadjusted loss and the aggregate adjusted loss, respectively, for the k th entity before $X_{k,j}$ is generated; $\Psi_{k,j}$ denotes the vector of values of the j th draw of stochastic symbols that you specify in the **SIMULATEDSYMBOL** statement for the k th entity; and A_k is the set of values of the severity adjustment variables that PROC CCDM reads from the DATA= data table for the k th entity.

The index n ($n = 1, \dots, N$) keeps track of the total number of severity draws, across all entities, that are made before $X_{k,j}$ is generated. So $S_{n-1} = \sum_{m=1}^{k-1} \sum_{l=1}^{N_m} X_{m,l} + \sum_{l=1}^{j-1} X_{k,l}$ and $S_{n-1}^d = \sum_{m=1}^{k-1} \sum_{l=1}^{N_m} X_{m,l}^d + \sum_{l=1}^{j-1} X_{k,l}^d$ are the aggregate unadjusted loss and aggregate adjusted loss, respectively, for all the entities that are processed before $X_{k,j}$ is generated. Note that S_{n-1} and S_{n-1}^d include the $j - 1$ draws that are made for the k th entity before $X_{k,j}$ is generated.

The initial values of all types of aggregate losses are set to 0. In other words, $S_0 = 0$; for all values of d , $S_0^d = 0$; for all values of k , $S_{k,0} = 0$; and for all combinations of k and d , $S_{k,0}^d = 0$.

In your severity adjustment program, you can use the following two additional placeholder keywords:

SEVSUMFOROBS

CUMSEVFOROBS

indicates the placeholder for $S_{k,j-1}$, which is the total loss that the k th entity incurs before the current loss event. PROC CCDM supplies this value to your program.

ADJSEVSUMFOROBSd

CUMADJSEVFOROBSd

indicates the placeholder for $S_{k,j-1}^d$, which is the total adjusted loss for the d th symbol that the k th entity incurs before the current loss event. PROC CCDM supplies this value to your program. For $d = 1$, you can use the symbol **_ADJSEVSUMFOROBS_** (alias **_CUMADJSEVFOROBS_**).

The previously described placeholder symbols **_SEVSUM_** and **_ADJSEVSUM_** represent S_{n-1} and S_{n-1}^d , respectively. If you have only one entity in the scenario ($K = 1$), then the values of **_SEVSUMFOROBS_** and **_ADJSEVSUMFOROBSd_** are identical to the values of **_SEVSUM_** and **_ADJSEVSUMd_**, respectively.

- For pure premium and custom simulation modes, there is one frequency draw per entity and only one severity draw for every frequency draw, so the index j is always equal to 1. PROC CCDM draws the frequency value N_k for each entity k and uses it according to the simulation mode as follows:

- For the pure premium simulation mode, the form of the adjustment function f^d that computes $X_{k,1}^d$ is

$$X_{k,1}^d = f^d(X_{k,1}, S_{n-1}, S_{n-1}^d, \Psi_{k,1}, A_k)$$

and PROC CCDDM computes the aggregate adjusted loss as

$$S^d = \sum_{k=1}^K N_k X_{k,1}^d$$

- For the custom simulation mode, you are free to combine the values of frequency, severity, stochastic symbols, and the adjustment variables in any manner to compute the adjusted severity value $X_{k,1}^d$. The form of the adjustment function f^d that computes $X_{k,1}^d$ is

$$X_{k,1}^d = f^d(N_k, X_{k,1}, S_{n-1}, S_{n-1}^d, \Psi_{k,1}, A_k)$$

and PROC CCDDM computes the aggregate adjusted loss as

$$S^d = \sum_{k=1}^K X_{k,1}^d$$

For both the pure premium and custom simulation modes, your adjustment program can use the previously described placeholder symbols `_SEVSUM_` and `_ADJSEVSUM_` to represent S_{n-1} and S_{n-1}^d , respectively.

Your severity adjustment program must implement all D adjustment functions and assign the output of the d th function f^d to the d th adjusted severity symbol that you specify in the `ADJUSTEDSEVERITY=` option. PROC CCDDM executes the severity adjustment program for each j th severity value that it draws for each k th entity, and it uses the final value of the d th symbol as the value of $X_{k,j}^d$.

There is one caveat when a scenario consists of more than one entity ($K > 1$) and when you use any of the symbols for cumulative severity values (`_SEVSUM_`, `_ADJSEVSUMd_`, `_SEVSUMFOROBS_`, or `_ADJSEVSUMFOROBSd_`) in your severity adjustment program. In this case, to make the simulation realistic, it is important to randomize the order of N severity draws across K entities. For more information, see the section “Randomizing the Order of Severity Draws across Observations of a Scenario” on page 158.

Illustration of the Aggregate Adjusted Loss Simulation Process

This section continues the example in the section “Simulation with Regressors and No External Counts” on page 140 to illustrate the simulation of aggregate adjusted loss for the collective risk simulation mode.

Recall that the earlier example simulates a scenario that consists of five policyholders. Assume that you want to compute the distribution of the aggregate amount paid to all the policyholders in a year, where the payment for each loss is determined by a deductible and a per-payment limit. To begin with, you must record the deductible and limit information in the input `DATA=` data table. The following table shows the `DATA=` data table from the earlier example, extended to include two variables, `Deductible` and `Limit`:

Obs	age	gender	carType	deductible	limit
1	30	2	1	250	5000
2	25	1	2	500	3000
3	45	2	2	100	2000
4	33	1	1	500	5000
5	50	1	1	200	2000

The variables Deductible and Limit are referred to as severity adjustment variables, because you need to use them to compute the adjusted severity. Let AmountPaid represent the value of adjusted severity that you are interested in. Further, let the following SAS programming statements encode your logic of computing the value of AmountPaid:

```
amountPaid = MAX(_sev_ - deductible, 0);
amountPaid = MIN(amountPaid, MAX(limit - _adjsevsumforobs_, 0));
```

PROC CCDM supplies your program with values of the placeholder symbols `_SEV_` and `_ADJSEVSUMFOROBS_`, which represent the value of the current unadjusted severity draw and the sum of adjusted severity values from the previous draws, respectively, for the observation that is being processed. The use of `_ADJSEVSUMFOROBS_` helps you ensure that the payment that is made to a particular policyholder in a year does not exceed the limit that is recorded in the variable Limit.

To simulate a sample for the aggregate of AmountPaid, you need to submit a PROC CCDM step whose structure is like the following:

```
proc ccdm data=<data table name> adjustedseverity=amountPaid
    severityest=<severity parameter estimates data table>
    countstore=<count model store>;
    severitymodel <severity distribution name(s)>;

    amountPaid = MAX(_sev_ - deductible, 0);
    amountPaid = MIN(amountPaid, MAX(limit - _adjsevsumforobs_, 0));
run;
```

The simulation process of one replication that generates one point of the aggregate loss sample and the corresponding point of the aggregate adjusted loss sample is as follows:

1. Use the values Age=30, Gender=2, and CarType=1 in the first observation to draw a count from the count distribution. Let that count be 3. Repeat the process for the remaining four observations. Let the counts be as shown in the Count column in the following table:

Obs	age	gender	carType	deductible	limit	count
1	30	2	1	250	5000	2
2	25	1	2	500	3000	1
3	45	2	2	100	2000	2
4	33	1	1	500	5000	3
5	50	1	1	200	2000	0

The Count column is shown for illustration only; it is not added as a variable to the DATA= data table.

2. The simulated counts from all the observations are added together to get a value of $N = 8$. This means that for this particular replication, you expect a total of eight loss events in a year from these five policyholders.

3. For the first observation, the scale parameter of the severity distribution is computed by using the values Age=30, Gender=2, and CarType=1. That value of the scale parameter is used together with estimates of the other parameters from the SEVERITYEST= data table to make two draws from the severity distribution. The process is repeated for the remaining four policyholders. The fifth policyholder does not generate any loss event for this particular replication, so no severity draws are made by using the fifth observation. Let the severity draws, rounded to integers for convenience, be as shown in the `_SEV_` column in the following table:

Obs	age	gender	carType	deductible	limit	count	<code>_sev_</code>		
1	30	2	1	250	5000	2	350	2100	
2	25	1	2	500	3000	1	4500		
3	45	2	2	100	2000	2	700	4300	
4	33	1	1	200	5000	3	600	1500	950
5	50	1	1	200	2000	0			

The `_SEV_` column is shown for illustration only; it is not added as a variable to the DATA= data table. The sample point for the aggregate unadjusted loss is computed by adding together the severity values of eight draws, which gives an aggregate loss value of 15,000. The aggregate unadjusted loss is also referred to as the ground-up loss.

For each severity draw, your severity adjustment program is executed to compute the adjusted severity, which is the value of AmountPaid in this case. For the draws in the preceding table, the values of AmountPaid are as follows:

Obs	deductible	limit	<code>_sev_</code>	<code>_adjsevsumforobs_</code>	amountPaid
1	250	5000	350	0	100
1	250	5000	2100	100	1850
2	500	3000	4500	0	3000
3	100	2000	700	0	600
3	100	2000	4300	600	1400
4	200	5000	600	0	400
4	200	5000	1500	400	1300
4	200	5000	950	1700	750

The adjusted severity values are added together to compute the cumulative payment value of 9,400, which forms the first sample point for the aggregate adjusted loss.

After recording the aggregate unadjusted and aggregate adjusted loss values in their respective samples, the process returns to step 1 to compute the next sample point unless the specified number of sample points have been simulated.

In this particular example, you can verify that the order in which the eight loss events are simulated does not affect the aggregate adjusted loss. As a simple example, consider the following order of draws, which is different from the consecutive order that the preceding table uses:

Obs	deductible	limit	<code>_sev_</code>	<code>_adjsevsumforobs_</code>	amountPaid
4	200	5000	600	0	400
3	100	2000	4300	0	2000
1	250	5000	350	0	100
3	100	2000	700	2000	0
4	200	5000	950	400	750

1	250	5000	2100	100	1850
2	500	3000	4500	0	3000
4	200	5000	1500	1150	1300

Although the payments that are made for individual loss events differ, the aggregate adjusted loss is still 9,400.

However, in general, when you use a cumulative severity value such as `_ADJSEVSUMFOROBS_` in your program, the order in which the draws are processed affects the final value of aggregate adjusted loss. For more information, see the sections “[Randomizing the Order of Severity Draws across Observations of a Scenario](#)” on page 158 and “[Illustration of the Need to Randomize the Order of Severity Draws](#)” on page 159.

Randomizing the Order of Severity Draws across Observations of a Scenario

If you specify a scenario that consists of more than one entity, then it is assumed that each entity generates its loss events independently from the other entities. In other words, the time at which the loss event of one entity is generated or recorded is independent of the time at which the loss event of another entity is generated or recorded. To honor this assumption of independence among entities, the order in which those entities are processed must be randomized across K entities such that no entity is preferred over another. The K entities are represented by K observations of the scenario in the `DATA=` data table. If you specify external counts, the K observations correspond to the observations that have the same replication identifier value. If you do not specify the external counts, then the K observations correspond to all the observations in the `BY` group, or to all the observations in the entire `DATA=` data table if you do not specify the `BY` statement.

The randomization process depends on the simulation mode:

- For the collective risk simulation mode, if entity k generates N_k loss events, where N_k is adjusted to conform to the upper limit that is imposed by the value of the `MAXCOUNTDRAW=` option, then the total number of loss events for a group of K entities is $N = \sum_{k=1}^K N_k$. To simulate the aggregate loss for this group, N severity draws are made and aggregated to compute one point of the compound distribution sample. To honor the assumption of independence among entities, the order of those N severity draws must be randomized across K entities such that no entity is preferred over another.

PROC CCDM implements the randomization process over N severity draws as follows: First, it chooses one of the K observations at random, say the k th observation, and draws one severity value ($X_{k,j}$) from the severity distribution that is implied by that observation. It then supplies the severity value, along with the values of the severity adjustment variables (A_k) and random draws of simulated symbols ($\Psi_{k,j}$), to your severity adjustment program to compute the adjusted severity values ($X_{k,j}^d$) and updates the respective aggregate adjusted losses (S^d). Next, it chooses another observation at random, draws one severity value from its implied severity distribution, and repeats the analysis of computing $X_{k,j}^d$ and updating S^d . In each step, PROC CCDM increments by 1 the total number of events that are simulated for the selected observation k . When it simulates all N_k events for an observation k , it retires that observation and continues the process with the remaining observations until it makes a total of N severity draws.

- For pure premium and custom simulation modes, there is one frequency and severity draw per entity. To honor the assumption of independence among entities, PROC CCDM randomizes the order in which it processes the entities as follows: First, it chooses one of the K observations at random, say the k th observation. It draws one frequency value and one severity value from the count and severity

distributions, respectively, that are implied by the k th observation. It then supplies those frequency (N_k) and severity ($X_{k,1}$) values, along with the values of the severity adjustment variables (A_k) and random draws of simulated symbols ($\Psi_{k,1}$), to your severity adjustment program to compute the adjusted severity values ($X_{k,1}^d$) and updates the respective aggregate adjusted losses (S^d). Next, PROC CCDM retires the processed observation, chooses another observation at random from the remaining $K - 1$ observations, and repeats the analysis of computing $X_{k,1}^d$ and updating S^d . The process continues until all K observations are exhausted.

The randomization process is especially important in the context of simulating an adjusted compound distribution sample when your severity adjustment program uses the aggregate adjusted severity to adjust the next severity value. For an illustration of the need to randomize in such cases for the collective risk simulation mode, see the next section.

Illustration of the Need to Randomize the Order of Severity Draws

This section uses the example in the section “[Illustration of the Aggregate Adjusted Loss Simulation Process](#)” on page 155, but with the following PROC CCDM step:

```
proc ccdm data=<data table name> adjustedseverity=amountPaid
    severityest=<severity parameter estimates data table>
    countstore=<count model store>;
    severitymodel <severity distribution name(s)>;

    if (_adjsevsum_ > 15000) then
        amountPaid = 0;
    else do;
        penaltyFactor = MIN(3, 15000/(15000 - _adjsevsum_));
        amountPaid = MAX(0, _sev_ - deductible * penaltyFactor);
    end;
run;
```

The severity adjustment statements in the preceding code compute the value of AmountPaid by using the collective risk simulation mode and the following provisions in the insurance policy:

- There is a limit of 15,000 on the total amount that can be paid in a year to the group of policyholders that is being simulated. The amount of payment for each loss event depends on the total amount of payments before that loss event.
- The penalty for incurring more losses is imposed in the form of an increased deductible. In particular, the deductible is increased by the ratio of the maximum cumulative payment (15,000) to the amount that remains available to pay for future losses in the year. The factor by which the deductible can be raised has a limit of three.

This example illustrates only step 3 of the simulation process, where randomization is done. It assumes that step 2 of the simulation process is identical to step 2 of the example in the section “[Illustration of the Aggregate Adjusted Loss Simulation Process](#)” on page 155. At the beginning of step 3, let the severity draws from all the observations be as shown in the `_SEV_` column in the following table:

Obs	age	gender	carType	deductible	count	_sev_	
1	30	2	1	250	2	350	2100
2	25	1	2	500	1	4500	
3	45	2	2	100	2	700	4300
4	33	1	1	200	3	600	1500 950
5	50	1	1	200	0		

If the order of these eight draws is not randomized, then all the severity draws for the first observation are adjusted before all the severity draws of the second observation, and so on. The execution of the severity adjustment program leads to the following sequence of values for AmountPaid:

Obs	deductible	_sev_	_adjsevsum_	penaltyFactor	amountPaid
1	250	350	0	1	100
1	250	2100	100	1.0067	1848.32
2	500	4500	1948.32	1.1493	3925.36
3	100	700	5873.68	1.6436	535.64
3	100	4300	6409.32	1.7461	4125.39
4	200	600	10534.72	3	0
4	200	1500	10534.72	3	900
4	200	950	11434.72	3	350

The preceding sequence of simulating loss events results in a cumulative payment of 11,784.72.

If the sequence of draws is randomized over observations, then the computation of the cumulative payment might proceed as follows for one instance of randomization:

Obs	deductible	_sev_	_adjsevsum_	penaltyFactor	amountPaid
2	500	4500	0	1	4000
1	250	350	4000	1.3636	9.09
3	100	700	4009.09	1.3648	563.52
4	200	950	4572.61	1.4385	662.30
4	200	1500	5234.91	1.5361	1192.78
1	250	2100	6427.69	1.7498	1662.54
4	200	600	8090.24	2.1708	165.83
3	100	4300	8256.07	2.2242	4077.58

In this example, a policyholder is identified by the value in the Obs column. As the table indicates, PROC CCDM randomizes the order of loss events not only across policyholders but also across the loss events that a given policyholder incurs. The particular sequence of loss events that is shown in the table results in a cumulative payment of 12,333.65. This differs from the cumulative payment that results from the previously considered nonrandomized sequence of loss events, which tends to penalize the fourth policyholder by always processing her payments after all other payments, with a possibility of underestimating the total amount paid. This comparison not only illustrates that the order of randomization affects the aggregate adjusted loss sample but also corroborates the arguments about the importance of order randomization that are made at the beginning of the section “[Randomizing the Order of Severity Draws across Observations of a Scenario](#)” on page 158.

Parameter Perturbation Analysis

It is important to realize that most of the parameters of the frequency and severity models are estimated and that there is uncertainty associated with the parameter estimates. Any compound distribution estimate that you compute by using these uncertain parameter estimates is inherently uncertain. The aggregate loss sample that you simulate by using the mean estimates of the parameters is just one possible sample from the compound distribution. If information about the uncertainty in parameter estimates is available, then it is recommended that you perform parameter perturbation analysis that generates multiple samples of the compound distribution, in which each sample is simulated by using a set of perturbed parameter estimates. You can use the **NPERTURBEDSAMPLES=** option in the PROC CCDDM statement to specify the number of perturbed samples to be generated. PROC CCDDM creates the set of perturbed parameter estimates by making a random draw of the parameter values from their joint probability distribution. If you specify **NPERTURBEDSAMPLES=P**, then PROC CCDDM creates P sets of perturbed parameters, and each set is used to simulate a full aggregate sample. The summary analysis of P such aggregate loss samples results in a set of P estimates for each summary statistic and percentile of the compound distribution. The mean and standard deviation of this set of P estimates quantify the uncertainty that is associated with the compound distribution.

The parameter estimate uncertainty information is available in the form of either the variance-covariance matrix of the parameter estimates or standard errors of the parameter estimates. If the variance-covariance matrix is available and is positive definite, then PROC CCDDM assumes that the joint probability distribution of the parameter estimates is a multivariate normal distribution, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where the mean vector $\boldsymbol{\mu}$ is the set of point parameter estimates and $\boldsymbol{\Sigma}$ is the variance-covariance matrix. If the variance-covariance matrix is not available or is not positive definite, or if you specify the **IGNORECOV** option, then PROC CCDDM assumes that each parameter has a univariate normal distribution, $\mathcal{N}(\mu, \sigma^2)$, where μ is the point estimate of the parameter and σ is the standard error of the parameter estimate.

Specifying Parameter Uncertainty

If you specify the severity models by using the SEVERITYEST= data table, then the point parameter estimates are expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='EST'**, the standard errors are expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='STDERR'**, and the variance-covariance matrix is expected to be available in the SEVERITYEST= data table in observations for which **_TYPE_='COV'**. When you use the SEVSELECT procedure to create the SEVERITYEST= data table, you need to specify the COVOUT option in the PROC SEVSELECT statement to make the variance-covariance estimates available in the SEVERITYEST= data table.

If you specify the severity models by using the SEVERITYSTORE= item store, then you need to specify the OUTSTORE= option in the PROC SEVSELECT statement to create that item store, which includes the point parameter estimates and standard errors by default. In addition, you need to specify the COVOUT option in the PROC SEVSELECT statement to make the variance-covariance estimates available in the SEVERITYSTORE= item store.

If you specify the frequency model by using the COUNTSTORE= item store, then the uncertainty estimates are available in that item store, because the CNTSELECT procedure always writes the point estimates, standard errors, and variance-covariance matrix of the parameters to the item store.

If you specify the frequency model by using the **COUNTMODEL** statement, then it is recommended that you use the **STDERROR=** option to specify the standard error for each parameter.

If you specify an external frequency model by using the EXTERNALCOUNTS statement, then there is no way to specify the uncertainty in the frequency model, so PROC CCDM conducts the perturbation analysis by perturbing only the parameters of the severity model.

Output of Perturbation Analysis

PROC CCDM displays the effect of parameter perturbation on the summary statistics and percentiles of the compound distribution in the ODS tables PerturbedSummary and PerturbedPctlSummary, respectively. For each statistic, the tables report the mean and standard error that PROC CCDM computes by aggregating the statistics from all P perturbed samples. For more information, see the section “[Displayed Output](#)” on page 172.

You can specify the OUTSUM statement to write the summary statistics and percentiles of individual perturbed samples to an output data table. You can also specify the PERTURBOUT option in the OUTPUT statement to write all perturbed samples in their entirety to an output data table.

If you specify the ADJUSTEDSEVERITY= option in the PROC CCDM statement, then PROC CCDM performs a separate perturbation analysis for the distribution of each aggregate adjusted loss.

Fixed Parameter-Value Distributions

Table 5.2 and Table 5.3 define the continuous and discrete probability distributions, respectively, that you can use in the SIMULATEDSYMBOL and COUNTMODEL statements. Note the following points when reading both tables:

- You can use any one of the names in the Names column to specify the distribution that the Distribution column identifies. The names are not case-sensitive.
- When you specify the parameter values for a distribution in the COUNTMODEL or SIMULATEDSYMBOL statement, the values must appear in the order in which the parameters are listed in the Parameters column. For example, specifying burr(3, 0.5, 1) means you are specifying $\theta = 3$, $\alpha = 0.5$, and $\gamma = 1$; similarly, specifying negbin(0.3, 5) means you are specifying $p = 0.3$ and $n = 5$.

The following points pertain to the continuous distributions in Table 5.2:

- You can use a continuous distribution in the COUNTMODEL statement only if you use the SIMULATIONMODE= option to specify a simulation mode other than the collective risk mode.
- The definitions of probability density function (PDF) and cumulative distribution functions (CDF) use the following mathematical notation:
 - x denotes the value of the continuous random variable in Table 5.2.
 - $z = x/\theta$, wherever z is used.
 - θ denotes the scale parameter for the continuous distributions.
 - μ denotes the mean parameter, except for the lognormal distribution, where μ is the log-scale parameter ($\mu = \log(\theta)$).
 - $\gamma(a, b) = \int_0^b t^{a-1} e^{-t} dt$ is the lower incomplete gamma function.

– $\Phi(y) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{y}{\sqrt{2}} \right) \right)$ is the standard normal CDF.

- Parameters of the inverse Gaussian (Wald) distribution are related to the parameters of the scaled inverse Gaussian distribution as $\mu = \theta$ and $\lambda = \theta\alpha$.
- The parameters of the Tweedie and scaled Tweedie distribution are related to each other as described in the section “[Tweedie Distributions](#)” on page 865 in Chapter 15, “[The SEVSELECT Procedure](#).” PROC SEVSELECT uses the same parameterizations.
- The parameters of the Wald and scaled Wald distributions are related to each other as $\theta = \mu$ and $\alpha = \lambda/\mu$. PROC SEVSELECT uses the scaled Wald parameterization because it has a scale parameter.

Table 5.2 Fixed Parameter-Value Continuous Distributions

Distribution	Names	Parameters	PDF (f) and CDF (F)
Burr’s Type XII	BurrTypeXII	$\theta > 0$	$f(x) = \frac{\alpha\gamma z^\gamma}{x(1+z^\gamma)^{(\alpha+1)}}$
	Burr	$\alpha > 0$	$F(x) = 1 - \left(\frac{1}{1+z^\gamma} \right)^\alpha$
	Bur	$\gamma > 0$	
Exponential	Exponential	$\theta > 0$	$f(x) = \frac{1}{\theta} e^{-x/\theta}$
	Expo		$F(x) = 1 - e^{-x/\theta}$
	Exp		
Gamma	Gamma	$\theta > 0$	$f(x) = \frac{z^\alpha e^{-z}}{x\Gamma(\alpha)}$
	Gamm	$\alpha > 0$	$F(x) = \frac{\gamma(\alpha, z)}{\Gamma(\alpha)}$
	Gam		
Generalized Pareto	GeneralizedPareto	$\theta > 0$	$f(x) = \frac{1}{\theta} (1 + \xi z)^{-1-1/\xi}$
	GPD	$\xi > 0$	$F(x) = 1 - (1 + \xi z)^{-1/\xi}$
	GPare		
Inverse Gaussian (Wald)	Wald	$\mu > 0$	$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} e^{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}}$
	IGauss	$\lambda > 0$	$F(x) = \Phi\left(\left(\frac{x}{\mu} - 1\right)\sqrt{\frac{\lambda}{x}}\right) +$
	IGau		$\Phi\left(-\left(\frac{x}{\mu} + 1\right)\sqrt{\frac{\lambda}{x}}\right) e^{2\lambda/\mu}$
Lognormal	Lognormal	μ	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$
	Logn	$\sigma > 0$	$F(x) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$
	Lgn		
Normal	Normal	μ	$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
	Gauss	$\sigma > 0$	$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$
	Norm		
Pareto (Type I)	ParetoTypeI	$\theta > 0$	$f(x) = \frac{\alpha}{\theta} \left(\frac{1}{x}\right)^{\alpha+1}$ if $x \geq \theta$; 0 otherwise
	Pareto1	$\alpha > 0$	$F(x) = 1 - \left(\frac{\theta}{x}\right)^\alpha$
	Pare		

Table 5.2 *continued*

Distribution	Names	Parameters	PDF (f) and CDF (F)
Pareto (Type II)	ParetoTypeII	$\theta > 0$	$f(x) = \frac{\alpha\theta^\alpha}{(x+\theta)^{\alpha+1}}$
	Pareto2	$\alpha > 0$	$F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^\alpha$
	Par2		
Scaled inverse	ScaledWald	$\theta > 0$	$f(x) = \frac{1}{\theta} \sqrt{\frac{\alpha}{2\pi z^3}} e^{-\frac{\alpha(z-1)^2}{2z}}$
Gaussian (Wald)	SWald	$\alpha > 0$	$F(x) = \Phi\left((z-1)\sqrt{\frac{\alpha}{z}}\right) +$
	SIGau		$\Phi\left(-(z+1)\sqrt{\frac{\alpha}{z}}\right) e^{2\alpha}$
Scaled Tweedie	ScaledTweedie	$\theta > 0$	$f(x) = a(x, \theta, \lambda, p) \exp\left(-\frac{x}{\theta} - \lambda\right)$
	STweedie	$\lambda > 0$	$F(x) = \int_0^x f(t) dt$
	STwdy	$1 < p < 2$	
Tweedie	Tweedie	$p > 1$	$f(x) = a(x, \phi) \exp\left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$
	Twee	$\mu > 0$	$F(x) = \int_0^x f(t) dt$
	Twdy	$\phi > 0$	
Uniform	Uniform	l	$f(x) = \frac{1}{r-l}$ if $l \leq x \leq r$; 0 otherwise
	Unif	r	$F(x) = \frac{x-l}{r-l}$ if $l \leq x < r$;
	Uni		1 if $x \geq r$; 0 otherwise
Weibull	Weibull	$\theta > 0$	$f(x) = \frac{\tau}{\theta} z^{\tau-1} e^{-z^\tau}$
	Weib	$\tau > 0$	$F(x) = 1 - e^{-z^\tau}$
	Wbl		

The following points pertain to the discrete distributions in [Table 5.3](#):

- The definition of the probability mass functions (PMF) uses m to denote the value of the discrete random variable. The table does not show an explicit expression for the CDF because the CDF of each distribution is defined in the standard manner as $F(m) = \sum_{i=0}^m f(i)$.
- PROC CCDM supports three parameterizations of the negative binomial distribution to offer flexibility of specification. The standard parameterization (Negbin) is the same as the one you specify in the SAS PDF and CDF functions. The other two are inspired by the parameterizations that the CNTSELECT procedure supports:
 - The NegbinP1 parameterization allows the variance to be linear in mean. NegbinP1(μ, α) is equivalent to Negbin($p = 1/(1 + \alpha)$, $n = \mu/\alpha$).
 - The NegbinP2 parameterization allows the variance to be quadratic in mean. NegbinP2(μ, α) is equivalent to Negbin($p = 1/(1 + \mu\alpha)$, $n = 1/\alpha$).
- The logarithmic versions of the discrete distributions correspond to the distributions that the CNTSELECT procedure uses for the count regression models. The relationship of the parameters of the logarithmic versions and their nonlogarithmic counterparts is as follows:

- $\text{LogCMPoissonLambda}(\beta, \delta) = \text{CMPoisson}(\lambda = e^\beta, \nu = e^{-\delta})$. This distribution corresponds to the parameterization of the Conway-Maxwell-Poisson distribution that PROC CNTSELECT uses when you specify the `DIST=CMPOISSON` and `PARAMETER=LAMBDA` options in the `MODEL` statement of PROC CNTSELECT. The δ parameter corresponds to the `_lnNu` parameter that PROC CNTSELECT reports.
- $\text{LogCMPoissonMu}(\beta, \delta) = \text{CMPoisson}(\lambda = e^{\beta e^{-\delta}}, \nu = e^{-\delta})$. This distribution corresponds to the default parameterization of the Conway-Maxwell-Poisson distribution that PROC CNTSELECT uses, which is equivalent to specifying the `DIST=CMPOISSON` and `PARAMETER=MU` options in the `MODEL` statement of PROC CNTSELECT. The δ parameter corresponds to the `_lnNu` parameter that PROC CNTSELECT reports.
- $\text{LogNegbinP1}(\beta, \alpha) = \text{NegbinP1}(\mu = e^\beta, \alpha)$. This distribution corresponds to the parameterization of the negative binomial distribution that PROC CNTSELECT uses when you specify the `DIST=NEGBIN(P=1)` option in the `MODEL` statement of PROC CNTSELECT.
- $\text{LogNegbinP2}(\beta, \alpha) = \text{NegbinP2}(\mu = e^\beta, \alpha)$. This distribution corresponds to the parameterization of the default negative binomial distribution that PROC CNTSELECT uses when you specify the `DIST=NEGBIN` option in the `MODEL` statement of PROC CNTSELECT.
- $\text{LogPoisson}(\beta) = \text{Poisson}(\lambda = e^\beta)$. This distribution corresponds to the parameterization of the Poisson distribution that PROC CNTSELECT uses when you specify the `DIST=POISSON` option in the `MODEL` statement of PROC CNTSELECT.

Table 5.3 Fixed Parameter-Value Discrete Distributions

Distribution	Names	Parameters	Probability Mass Function (f)
Conway-Maxwell-Poisson (Standard)	CMPOisson	$\lambda > 0$	$f(m) = \frac{\lambda^m}{(m!)^\nu} \frac{1}{Z(\lambda, \nu)}$
	ConMaxPoi CMP	$\nu \geq 0$	where $Z(\lambda, \nu) = \sum_{j=0}^{\infty} \frac{\lambda^j}{(j!)^\nu}$
Log Conway-Maxwell-Poisson (λ -parameterization)	LogCMPoissonLambda	β	$f(m) = \frac{e^{\beta m}}{(m!)^{e^{-\delta}}} \frac{1}{Z(e^\beta, e^{-\delta})}$
	LogCMPL LCMPL	δ	where $Z(e^\beta, e^{-\delta}) = \sum_{j=0}^{\infty} \frac{e^{\beta j}}{(j!)^{e^{-\delta}}}$
Log Conway-Maxwell-Poisson (μ -parameterization)	LogCMPoissonMu	β	$f(m) = \frac{e^{\beta m e^{-\delta}}}{(m!)^{e^{-\delta}}} \frac{1}{Z(e^{\beta e^{-\delta}}, e^{-\delta})}$
	LogCMP LCMP	δ	where $Z(e^{\beta e^{-\delta}}, e^{-\delta}) = \sum_{j=0}^{\infty} \frac{e^{\beta j e^{-\delta}}}{(j!)^{e^{-\delta}}}$
Log negative binomial (variance = $e^\beta + \alpha e^\beta$)	LogNegbinP1	β	$f(m) = \frac{\Gamma(m+\alpha^{-1}e^\beta)}{m!\Gamma(\alpha^{-1}e^\beta)} (1-q)^{\alpha^{-1}e^\beta} q^m$
	LogNB1 LNB1	$\alpha > 0$	where $q = \frac{1}{\alpha^{-1}+1}$
Log negative binomial (variance = $e^\beta + \alpha e^{2\beta}$)	LogNegbinP2	β	$f(m) = \frac{\Gamma(m+\alpha^{-1})}{m!\Gamma(\alpha^{-1})} (1-q)^{\alpha^{-1}} q^m$
	LogNB2 LNB2	$\alpha > 0$	where $q = \frac{e^\beta}{\alpha^{-1}+e^\beta}$

Table 5.3 *continued*

Distribution	Names	Parameters	Probability Mass Function (f)
Log Poisson	LogPoisson LogPoi LPoi	β	$f(m) = e^{-e^\beta} \left(\frac{e^{\beta m}}{m!} \right)$
Negative binomial (standard)	NegativeBinomial Negbin Negb	$0 \leq p \leq 1$ $n > 0$	$f(m) = \binom{n+m-1}{n-1} p^n (1-p)^m$
Negative binomial (variance = $\mu + \alpha\mu$)	NegbinP1 Negb1 NB1	$\mu > 0$ $\alpha > 0$	$f(m) = \frac{\Gamma(m+\alpha^{-1}\mu)}{m!\Gamma(\alpha^{-1}\mu)} (1-q)^{\alpha^{-1}\mu} q^m$ where $q = \frac{1}{\alpha^{-1}+1}$
Negative binomial (variance = $\mu + \alpha\mu^2$)	NegbinP2 Negb2 NB2	$\mu > 0$ $\alpha > 0$	$f(m) = \frac{\Gamma(m+\alpha^{-1})}{m!\Gamma(\alpha^{-1})} (1-q)^{\alpha^{-1}} q^m$ where $q = \frac{\mu}{\alpha^{-1}+\mu}$
Poisson	Poisson Pois Poi	$\lambda > 0$	$f(m) = e^{-\lambda} \frac{\lambda^m}{m!}$

Descriptive Statistics

This section provides computational details for the descriptive statistics that are computed for each aggregate loss sample. You can also save these statistics in an OUTSUM= data table by specifying appropriate keywords in the OUTSUM statement.

This section gives specific details about the moment statistics. For more information about the methods of computing percentile statistics, see the description of the PCTLDEF= option in the UNIVARIATE procedure in the *Base SAS Procedures Guide: Statistical Procedures*.

Standard algorithms (Fisher 1973) are used to compute the moment statistics. The computational methods that the CCDM procedure uses are consistent with those that other SAS procedures use for calculating descriptive statistics.

Mean

The sample mean is calculated as

$$\bar{y} = \frac{\sum_{i=1}^n y_i}{n}$$

where n is the size of the generated aggregate loss sample and y_i is the i th value of the aggregate loss.

Standard Deviation

The standard deviation is calculated as

$$s = \sqrt{\frac{1}{d} \sum_{i=1}^n (y_i - \bar{y})^2}$$

where n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, and d is the divisor controlled by the **VARDEF=** option in the PROC CCDDM statement:

$$d = \begin{cases} n - 1 & \text{if VARDEF=DF (default)} \\ n & \text{if VARDEF=N} \end{cases}$$

Skewness

The sample skewness, which measures the tendency of the deviations to be larger in one direction than in the other, is calculated as

$$\frac{1}{d_s} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^3$$

where n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, s is the sample standard deviation, and d_s is the divisor controlled by the **VARDEF=** option in the PROC CCDDM statement:

$$d_s = \begin{cases} \frac{(n-1)(n-2)}{n} & \text{if VARDEF=DF (default)} \\ n & \text{if VARDEF=N} \end{cases}$$

If VARDEF=DF, then n must be greater than 2.

The sample skewness can be positive or negative; it measures the asymmetry of the data distribution and estimates the theoretical skewness $\sqrt{\beta_1} = \mu_3 \mu_2^{-\frac{3}{2}}$, where μ_2 and μ_3 are the second and third central moments. Observations that are normally distributed should have a skewness near zero.

Kurtosis

The sample kurtosis, which measures the heaviness of tails, is calculated as in Table 5.4, depending on the value that you specify in the **VARDEF=** option.

Table 5.4 Formulas for Kurtosis

VARDEF= Value	Formula
DF (default)	$\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$
N	$\frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \bar{y}}{s} \right)^4 - 3$

In these formulas, n is the size of the generated aggregate loss sample, y_i is the i th value of the aggregate loss, $\bar{y}$ is the sample mean, and s is the sample standard deviation. If VARDEF=DF, then n must be greater than 3.

The sample kurtosis measures the heaviness of the tails of the data distribution. It estimates the adjusted theoretical kurtosis denoted as $\beta_2 - 3$, where $\beta_2 = \frac{\mu_4}{\mu_2^2}$ and μ_4 is the fourth central moment. Observations that are normally distributed should have a kurtosis near zero.

Input Specification

PROC CCDM accepts the COUNTSTORE= item store, DATA= data table, SEVERITYEST= data table, and SEVERITYSTORE= item store as input. This section describes the information that they are expected to contain.

COUNTSTORE= Item Store

The COUNTSTORE= item store is expected to be created by using the STORE= option in the CNTSELECT procedure. You must specify the COUNTSTORE= item store when you do not specify either the COUNTMODEL statement or the EXTERNALCOUNTS statement. For more information, see the description of the STORE= option in the CNTSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

If you specify the BY statement, then the COUNTSTORE= item store must have been created by using a PROC CNTSELECT step that uses an identical BY statement.

DATA= Data Table

If you specify the BY statement, then the DATA= data table must contain all the BY variables that you specify in the BY statement.

If the severity models in the SEVERITYEST= data table or SEVERITYSTORE= item store contain any scale regressors, then all those regressors must be present in the DATA= data table.

If you specify the programming statements to compute an aggregate adjusted loss, and if any of your specified ADJUSTEDSEVERITY= option symbols depend on the severity adjustment variables, then the DATA= data table must contain all such adjustment variables.

The rest of the contents of the DATA= data table depends on whether you specify the EXTERNALCOUNTS statement. If you specify the EXTERNALCOUNTS statement, then the DATA= data table is expected to contain the COUNT= and ID= variables that you specify in the EXTERNALCOUNTS statement. If you specify the COUNTSTORE= item store but you do not specify the EXTERNALCOUNTS statement, then the DATA= data table must contain all the regressors that are present in the count model that the COUNTSTORE= item store contains.

You do not need to specify the DATA= data table if *all* the following conditions are true:

- You do not specify the BY statement.
- You specify the severity models such that none of them are scale regression models.
- You do not specify the EXTERNALCOUNTS statement.

- You specify the COUNTMODEL statement, or you specify a COUNTSTORE= item store such that the count model contains no count regressors.
- Your severity adjustment programming statements, if you specify any, do not use any external input.

If you specify the BY statement, then PROC CCDM analyzes only the BY groups that are present in the input source of the severity models and count models. If neither the severity models nor the count models contain regression effects, then the DATA= data table must contain BY variables and one row for each BY group that you want PROC CCDM to analyze.

SEVERITYEST= Data Table

The SEVERITYEST= data table is expected to contain the parameter estimates of the severity models. This is a required data table; you must specify it whenever you use PROC CCDM.

The SEVERITYEST= data table must be an OUTEST= data table that the SEVSELECT procedure creates. For more information, see the description of the OUTEST= data table in the SEVSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

If you specify the BY statement, then the SEVERITYEST= data table must contain all the BY variables that you specify in the BY statement.

SEVERITYSTORE= Item Store

The SEVERITYSTORE= item store is expected to be created by using the OUTSTORE= option in a PROC SEVSELECT statement. For more information, see the description of the OUTSTORE= option in the SEVSELECT procedure in *SAS Econometrics: Econometrics Procedures*.

You must specify this item store when you do not specify the SEVERITYEST= data table. Also, if your severity model is a scale regression model that contains classification or interaction effects, then you cannot use the SEVERITYEST= data table. You must specify such severity models by specifying the SEVERITYSTORE= item store.

If you specify the BY statement, then the SEVERITYSTORE= item store must have been created by using a PROC SEVSELECT step that uses an identical BY statement.

Output Data Tables

PROC CCDM writes the output data tables that you specify in the OUT= option of the OUTPUT and OUTSUM statements. It also produces an output data table of individual severity draws when you specify the OUTDRAW= option. The contents of these output data tables are described in the following sections.

OUTDRAW= Data Table

The OUTDRAW= data table records the individual severity draws.

If you specify the BY statement, then the data table contains variables that you specify in the BY statement. In addition, the OUTDRAW= data table contains the following variables:

`_SEVERITYMODEL_`

indicates the name of the severity distribution model.

`_COUNTMODEL_`

indicates the name of the count model. If you specify the `EXTERNALCOUNTS` statement, then the value of this variable is “`_EXTERNAL_`”.

`_DRAWID_`

indicates the identifier for the perturbed sample. This variable is created only when you specify the `NPERTURBEDSAMPLES=` option in the `PROC CCDM` statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

`_BEGINAGG_`

indicates whether the observation starts a new aggregate sample point or continues the previous aggregate sample point. A value of 1 indicates that a new aggregate sample point begins. A value of 0 indicates that the observation continues the aggregate sample point.

`_SEVERITYDRAW_`

indicates the value of the severity draw.

OUTSAMPLE= Data Table

The `OUTSAMPLE=` data table records the full sample of the aggregate loss and aggregate adjusted loss.

If you specify the `BY` statement, then the data table contains variables that you specify in the `BY` statement. In addition, the `OUTSAMPLE=` data table contains the following variables:

`_SEVERITYMODEL_`

indicates the name of the severity distribution model.

`_COUNTMODEL_`

indicates the name of the count model. If you specify the `EXTERNALCOUNTS` statement, then the value of this variable is “`_EXTERNAL_`”.

<sample variable>

indicates the value of the aggregate unadjusted loss. The name of this variable is the value of the `SAMPLEVAR=` option in the `OUTPUT` statement. If you do not specify the `SAMPLEVAR=` option, then this variable is named `_AGGSEV_`.

<adjusted sample variable 1> ... <adjusted sample variable *D*>

indicates the values of the *D* aggregate adjusted losses, where *D* is the number of adjusted severity symbols that you specify in the `ADJUSTEDSEVERITY=` option in the `PROC CCDM` statement. The names of these variables are the values of the `ADJSAMPLEVAR=` option in the `OUTPUT` statement, which are applied to the symbols in the order in which you specify them in the `ADJUSTEDSEVERITY=` option. If you specify fewer names in the `ADJSAMPLEVAR=` option than symbols in the `ADJUSTEDSEVERITY=` option, then the variable for each remaining adjusted severity symbol has the same name as the corresponding symbol. If you do not specify the `ADJSAMPLEVAR=` option in the `OUTPUT` statement, if you specify only one adjusted severity symbol in the `ADJUSTEDSEVERITY=` option, and if you specify the `ONEADJSEVCOMPAT` option, then the adjusted sample variable is named `_AGGADJSEV_`.

DRAWID indicates the identifier for the perturbed sample. This variable is created only when you specify the **NPERTURBEDSAMPLES=** option in the PROC CCDM statement and the **PERTURBOUT** option in the OUTPUT statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

Observations that have the same value for the **_DRAWID_** variable are always distributed across all worker nodes for the unperturbed sample (**_DRAWID_=0**). For each of the perturbed samples (**_DRAWID_ > 0**), observations are either distributed or partitioned across worker nodes depending on the value of the **PERTURBMODE=** option. For more information, see the description of the **PERTURBMODE=AUTO** option.

OUTSUM= Data Table

The OUTSUM= data table records the summary statistics and percentiles of the compound distributions of aggregate loss and aggregate adjusted loss. Only the estimates that you request in the OUTSUM statement are written to the OUTSUM= data table. For more information about the method of naming the variables that correspond to the summary statistics or percentiles, see the description of the **OUTSUM** statement.

If you specify the BY statement, then the data table contains variables that you specify in the BY statement. In addition, the OUTSUM= data table contains the following variables:

SEVERITYMODEL indicates the name of the severity distribution model.

COUNTMODEL indicates the name of the count model. If you specify the **EXTERNALCOUNTS** statement, then the value of this variable is “**_EXTERNAL_**”.

SAMPLEVAR indicates the name of the aggregate loss sample. For an unadjusted sample, the value of the variable is the value of the **SAMPLEVAR=** option that you specify in the OUTPUT statement. If you do not specify the **SAMPLEVAR=** option, then the default value is **_AGGSEV_**.

For an adjusted sample, the value of the variable is decided by the value of the **ADJSAMPLEVAR=** option that you specify in the OUTPUT statement or the name of the adjusted severity symbol. If you do not specify the **ADJSAMPLEVAR=** option, if you specify only one adjusted severity symbol in the **ADJUSTEDSEVERITY=** option, and if you specify the **ONEADJSEVCOMPAT** option, then the value is **_AGGADJSEV_**.

DRAWID indicates the identifier for the perturbed sample. This variable is created only when you specify the **NPERTURBEDSAMPLES=** option in the PROC CCDM statement. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

Displayed Output

The CCDM procedure optionally produces displayed output by using the Output Delivery System (ODS). You control all output by using the PRINT= option in the PROC CCDM statement. Table 5.5 relates the PRINT= options to ODS tables.

Table 5.5 ODS Tables Produced in PROC CCDM

ODS Table Name	Description	Option
CompoundInfo	Compound distribution information	Default
DataSummary	Input data summary	Default
Percentiles	Percentiles of the aggregate loss sample	PRINT=PERCENTILES
PerturbedPctlSummary	Perturbation analysis of percentiles	PRINT=PERTURBSUMMARY and NPerturbedSamples > 0
PerturbedSummary	Perturbation analysis of summary statistics	PRINT=PERTURBSUMMARY and NPerturbedSamples > 0
SummaryStatistics	Summary statistics of the aggregate loss sample	PRINT=SUMMARYSTATISTICS
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

PRINT= Option

This section provides detailed descriptions of the tables that you display by using different PRINT= options.

- If you do not specify the PRINT= option and if you do not specify the NOPRINT or PRINT=NONE options, then by default PROC CCDM produces the CompoundInfo, DataSummary, and SummaryStatistics ODS tables.

The “Compound Distribution Information” table (ODS name CompoundInfo) displays the information about the severity and count models.

The “Input Data Summary” table (ODS name DataSummary) is displayed when you specify the DATA= data table. The table displays the total number of observations and the valid number of observations in the data table. If you specify the EXTERNALCOUNTS statement, then the table also displays the number of replications and total number of loss events across all replications.

- If you specify PRINT=PERCENTILES, the “Percentiles” table (ODS name Percentiles) is displayed for the distribution of the aggregate loss. The table contains estimates of all predefined percentiles in addition to the percentiles that you request in the OUTSUM statement.

If you specify the programming statements and the ADJUSTEDSEVERITY= option symbols, then an additional table is displayed for the distribution of each aggregate adjusted loss. These tables also contain estimates of all predefined percentiles in addition to the percentiles that you request in the OUTSUM statement.

- If you specify `PRINT=PERTURBSUMMARY`, two tables are displayed for the distribution of the aggregate loss. The “Perturbed Summary Statistics” table (ODS name `PerturbedSummary`) displays the summary of the effect of perturbing model parameters on the following five summary statistics of the distribution: mean, standard deviation, variance, skewness, and kurtosis. The “Perturbed Percentiles” table (ODS name `PerturbedPctlSummary`) displays the perturbation summary for all predefined percentiles in addition to the percentiles that you request in the `OUTSUM` statement.

The tables are displayed only if you specify a value greater than 0 for the `NPERTURBEDSAMPLES=` option.

If you specify a value of P for the `NPERTURBEDSAMPLES=` option, then for each summary statistic and percentile, an average and standard error of the set of P values of that summary statistic or percentile are displayed in the respective perturbation summary tables.

If you specify the programming statements and the `ADJUSTEDSEVERITY=` option symbols, then additional perturbation summary tables are displayed for the distribution of each aggregate adjusted loss.

- If you specify `PRINT=SUMMARYSTATISTICS`, the “Summary Statistics” table (ODS name `SummaryStatistics`) is displayed for the distribution of the aggregate loss. The table contains estimates of the following summary statistics: number of observations in the sample, maximum value in the sample, minimum value in the sample, mean, median, standard deviation, interquartile range, variance, skewness, and kurtosis.

If you specify the programming statements and the `ADJUSTEDSEVERITY=` option symbols, then an additional table of summary statistics is displayed for the distribution of each aggregate adjusted loss.

ODS Graphics

This section describes the use of ODS for creating graphics with the CCDDM procedure.

NOTE: To create the plots, PROC CCDDM prepares a plot data table that contains an estimate of the empirical distribution function (EDF) for a carefully selected subsample of the simulated aggregate loss sample and brings that table back to the client machine where your SAS session is running. You can control the size of the plot data table by specifying the `NOBS=` and `NADJSEVTOPILOT=` options. The larger the size of this table, the more accurate are the plots, but the more expensive it is to prepare and display the plots. So it is recommended that you request plots only when the full aggregate loss sample is not too large and that you do not specify large values for the `NOBS=` and `NADJSEVTOPILOT=` options.

ODS Graph Names

PROC CCDDM assigns a name to each graph that it creates by using ODS. You can use these names to selectively refer to the graphs. The names are listed in Table 5.6.

Table 5.6 ODS Graphics Produced by PROC CCDDM

ODS Graph Name	Plot Description	PLOTS= Option
ConditionalDensityPlot	Conditional density plot	CONDITIONALDENSITY
DensityPlot	Probability density function plot	DENSITY
EDFPlot	Empirical distribution function plot	EDF

Conditional Density Plot

The conditional density plot helps you visually analyze two or three regions of the compound distribution by displaying a density function estimate that is conditional on the values of the aggregate loss that fall in those regions. You can specify the region boundaries in terms of quantiles by using the `LEFTQ=` and `RIGHTQ=` suboptions of the `PLOTS=CONDITIONALDENSITY` option. This is especially useful if you want to see the distribution of aggregate loss values in the right- and left-tail regions.

If you specify m symbols in the `ADJUSTEDSEVERITY=` option and if you specify `NADJSEVTOPLOT= $k$` in the `PLOTS=` option, then separate sets of conditional density plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Probability Density Function Plot

The probability density function (PDF) plot shows the nonparametric estimates of the PDF of the aggregate loss distribution. This plot includes histogram and kernel density estimates.

If you specify m symbols in the `ADJUSTEDSEVERITY=` option and if you specify `NADJSEVTOPLOT= $k$` in the `PLOTS=` option, then separate density plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Empirical Distribution Function Plot

The empirical density function (EDF) plot shows the nonparametric estimate of the cumulative distribution function of the aggregate loss distribution. You can specify the `ALPHA=` suboption of the `PLOTS=EDF` option to request that the upper and lower confidence limits be plotted for each EDF estimate. By default, the confidence interval is not plotted.

If you specify m symbols in the `ADJUSTEDSEVERITY=` option and if you specify `NADJSEVTOPLOT= $k$` in the `PLOTS=` option, then separate EDF plots are displayed for the first l aggregate adjusted loss samples, where $l = \min(k, m)$.

Examples: CCDM Procedure

Example 5.1: Estimating the Probability Distribution of Insurance Payments

The primary outcome of running the CCDM procedure is the estimate of the compound distribution of aggregate loss, given the distributions of frequency and severity of the individual losses. This aggregate loss is often referred to as the ground-up loss. If you are an insurance company or a bank, you are also interested in acting on the ground-up loss by computing an entity that is derived from the ground-up loss. For example, you might want to estimate the distribution of the amount that you are expected to pay for the losses or the distribution of the amount that you can offload to another organization, such as a reinsurance company. PROC CCDM enables you to specify a severity adjustment program, which is a sequence of SAS programming statements that adjust the severity of the individual loss event to compute the entity of interest. Your severity adjustment program can use external information that is recorded as variables in the observations of the `DATA=` data table in addition to placeholder symbols for information that PROC CCDM generates internally, such as the severity of the current loss event (`_SEV_`) and the sum of the adjusted severity

values of the events that have been simulated thus far for the current sample point (`_ADJSEVSUM_`). If you are analyzing a scenario that contains more than one observation, then you can also access the cumulative severity and cumulative adjusted severity for the current observation by using the `_SEVSUMFOROBS_` and `_ADJSEVSUMFOROBS_` symbols.

This example continues the example of the section “[Scenario Analysis](#)” on page 105 to illustrate how you can estimate the distribution of the aggregate amount that is paid to a group of policyholders. Let the amount that is paid to an individual policyholder be computed by using what is usually referred to as a *disappearing deductible* (Klugman, Panjer, and Willmot 1998, Ch. 2). If X denotes the ground-up loss that a policyholder incurs, d denotes the lower limit on the deductible, d' denotes the upper limit on the deductible, and u denotes the limit on the total payments that are made to a policyholder in a year, then Y , the amount that is paid to the policyholder for each loss event, is defined as follows:

$$Y = \begin{cases} 0 & X \leq d \\ d' \frac{X-d}{d'-d} & d < X \leq d' \\ X & d' < X \leq u \\ u & X > u \end{cases}$$

You can use a set of SAS programming statements to encode this logic.

The following DATA step extends the data table `mycas.GroupOfPolicies` from the example in the section “[Scenario Analysis](#)” on page 105 to include three additional variables for each policyholder: `LowDeductible` to record d , `HighDeductible` to record d' , and `Limit` to record u .

```
/* Generate the scenario data table for multiple policyholders */
data groupOfPolicies(keep=policyholderId age gender carType annualMiles
                    education carSafety income
                    lowDeductible highDeductible limit annualLimit);
  call streaminit(67897);

  do policyholderId=1 to 5;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then gender = 1; * female;
    else gender = 2; * male;

    if (rand('UNIFORM') < 0.7) then carType = 1; * sedan;
    else carType = 2; * SUV;

    annualMiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

    educationLevel = rand('UNIFORM');
    if (educationLevel < 0.5) then education = 1; *high school graduate;
    else if (educationLevel < 0.85) then education = 2; *college graduate;
    else education = 3; *advanced degree;

    carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

    income = MAX(15000,int(rand('NORMAL', education*30000, 50000)))/100000;

    lowDeductible = 100*(1+floor(rand('UNIFORM')*5));
    highDeductible = lowDeductible + 500*(1+floor(rand('UNIFORM')*2));
```

```

        limit = 2500*(1+floor(rand('UNIFORM')*3));
        annualLimit = 10000*(1+floor(rand('UNIFORM')*2));

        output;
    end;
run;

/* Load data into the CAS server */
data mycas.groupOfPolicies;
    set groupOfPolicies;
run;

```

The data table contains the observations as shown in [Output 5.1.1](#).

Output 5.1.1 Scenario Analysis Data for Multiple Policyholders with Policy Provisions

policyholderid	age	gender	carType	annualMiles	education	carSafety	income
1	1.18	2	1	2.2948	3	0.99532	1.59870
2	0.66	2	2	2.8148	1	0.05625	0.67539
3	0.82	1	2	1.6130	2	0.84146	1.05940
4	0.44	1	1	1.2280	3	0.14324	0.24110
5	0.44	1	1	0.9670	2	0.08656	0.65979

lowDeductible	highDeductible	limit	annualLimit
400	1400	7500	10000
300	1300	2500	20000
100	1100	5000	10000
300	800	5000	20000
100	1100	5000	20000

The following PROC CCDM step estimates the compound distributions of the aggregate loss and the aggregate amount that is paid to the group of policyholders in the data table mycas.GroupOfPolicies by using the count model that is stored in the item store mycas.CountregModel and the lognormal severity model that is stored in the data table mycas.SevRegEst:

```

/* Simulate the aggregate loss distribution and aggregate adjusted
   loss distribution for the scenario of multiple policyholders */
proc ccdm data=mycas.groupOfPolicies nreplicates=10000 seed=13579 print=all
    countstore=mycas.countregmodel severityest=mycas.sevregest
    nperturbedSamples=50 adjustedseverity=amountPaid;
    severitymodel logn;

    if (_sev_ <= lowDeductible) then
        amountPaid = 0;
    else do;
        if (_sev_ <= highDeductible) then
            amountPaid = highDeductible *
                (_sev_-lowDeductible)/(highDeductible-lowDeductible);
        else
            amountPaid = MIN(_sev_, limit); /* imposes per-loss payment limit */
        end;
    end;
run;

```

The preceding step uses a severity adjustment program to compute the value of the symbol AmountPaid and specifies that symbol in the ADJUSTEDSEVERITY= option in the PROC CCDM step. The program is executed for each simulated loss event. The PROC CCDM supplies your program with the value of the severity in the _SEV_ placeholder symbol.

The “Sample Summary Statistics” table in [Output 5.1.2](#) shows the summary statistics of the compound distribution of the aggregate ground-up loss. The “Adjusted Sample Summary Statistics” table shows the summary statistics of the compound distribution of the aggregate AmountPaid. The average aggregate payment is about 4,352, as compared to the average aggregate ground-up loss of 5,904.

Output 5.1.2 Summary Statistics of Compound Distributions of the Total Loss and Total Amount Paid

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Compound Distribution Information	
Severity Model	Lognormal Distribution
Scale Model Effects	carSafety carType income
Count Model	NegBin(p=2) Model in Item Store COUNTREGMODEL

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Sample Summary Statistics			
Mean	5903.9	Median	4746.5
Standard Deviation	4961.4	Interquartile Range	5133.0
Variance	24615924.9	Minimum	0
Skewness	4.07140	Maximum	138633.3
Kurtosis	62.07742	Sample Size	10000

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

Summary Statistics for amountPaid			
Mean	4352.4	Median	3770.4
Standard Deviation	3152.0	Interquartile Range	4086.7
Variance	9935181.2	Minimum	0
Skewness	1.12775	Maximum	31285.3
Kurtosis	1.92631	Sample Size	10000

The perturbation summary of the distribution of AmountPaid is shown in [Output 5.1.3](#). It shows that you can expect to pay a median of $3,780 \pm 229$ to this group of five policyholders in a year. Also, if the 99.5th percentile defines the worst case, then you can expect to pay $15,614 \pm 772$ in the worst case.

Output 5.1.3 Perturbation Summary of the Total Amount Paid

Percentile Perturbation Analysis for amountPaid		
Percentile	Estimate	Standard Error
1	0.48525	2.41371
5	389.26392	60.97476
25	1985.1	159.72872
50	3780.1	228.80921
75	6122.2	325.59015
95	10414.4	499.01931
99	14103.5	677.80532
99.5	15614.3	771.92423
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Now consider that, in the future, you want to modify your company's policy provisions to add a limit on the total amount of payment that an individual policyholder receives in one year and to impose a group limit of 15,000 on the total amount of payments the group as a whole receives in one year. You can analyze the effects of these modified policy provisions on the distribution of the aggregate paid amount by recording the individual policyholder's annual limit in the variable `AnnualLimit` of the input data table and then modifying your severity adjustment program by using the placeholder symbols `_ADJSEVSUMFOROBS_` and `_ADJSEVSUM_`, as in the following PROC CCDM step:

```

/* Simulate the aggregate loss distribution and aggregate adjusted
   loss distribution for the modified set of policy provisions */
proc ccdm data=mycas.groupOfPolicies nreplicates=10000 seed=13579 print=all
      countstore=mycas.countregmodel severityest=mycas.sevregist
      nperturbedSamples=50 adjustedseverity=amountPaid;
severitymodel logn;

if (_sev_ <= lowDeductible) then
  amountPaid = 0;
else do;
  if (_sev_ <= highDeductible) then
    amountPaid = highDeductible *
      (_sev_-lowDeductible)/(highDeductible-lowDeductible);
  else
    amountPaid = MIN(_sev_, limit); /* imposes per-loss payment limit */

  /* impose policyholder's annual limit */
  amountPaid = MIN(amountPaid, MAX(0,annualLimit - _adjsevsumforobs_));

  /* impose group's annual limit */
  amountPaid = MIN(amountPaid, MAX(0,15000 - _adjsevsum_));
end;
run;

```

The results of the perturbation analysis for these modified policy provisions are shown in [Output 5.1.4](#). When you compare them to the results in [Output 5.1.3](#), you see that the additional policy provisions of restricting

the total payment to the policyholder and the group have reduced the median payment from $3,780 \pm 229$ to $3,787 \pm 244$, and reduced the worst-case payment (99.5th percentile) from $15,614 \pm 772$ to $14,815 \pm 357$.

Output 5.1.4 Perturbation Summary of the Total Amount Paid after Modified Policy Provisions

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin($p=2$)

Percentile Perturbation Analysis for amountPaid		
Percentile	Estimate	Standard Error
1	0.22295	1.35646
5	390.74702	57.33827
25	1991.7	166.23963
50	3786.8	243.80003
75	6140.3	332.63944
95	10395.7	483.31055
99	13849.3	658.19047
99.5	14815.0	357.24279
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Example 5.2: Using Externally Simulated Count Data

The COUNTREG procedure in SAS/ETS and the CNTSELECT procedure in SAS Econometrics enable you to estimate count regression models that are based on the most commonly used discrete distributions, such as the Poisson, negative binomial (both $p = 1$ and $p = 2$), and Conway-Maxwell-Poisson distributions. Those procedures also enable you to fit zero-inflated models that are based on Poisson, negative binomial ($p = 2$), and Conway-Maxwell-Poisson distributions. However, you might encounter situations in which you want to use a different method of fitting count regression models. For example, if you are modeling the number of loss events that are incurred by two financial instruments such that there is some dependency between the two, then you might use multivariate frequency modeling methods and simulate the counts for each instrument by using the dependency structure between the count model parameters of the two instruments. As another example, you might want to use different types of count models for different BY groups in your data; this is not possible in PROC COUNTREG or PROC CNTSELECT. So you need to simulate the counts for such BY groups externally. The CCDM procedure enables you to supply externally simulated counts by using the EXTERNALCOUNTS statement. PROC CCDM then does not need to simulate the counts internally; it simulates only the severity of each loss event by using the severity model estimates that you specify in the SEVERITYEST= data table or the SEVERITYSTORE= item store. The simulation process is described and illustrated in the section “[Simulation with External Counts](#)” on page 143.

Consider that you are a bank, and as part of quantifying your operational risk, you want to estimate the aggregate loss distributions for two lines of business, retail banking and commercial banking, by using some key risk indicators (KRIs). Assume that your model fitting and model selection process has determined that the Poisson regression model and negative binomial regression model are the best-fitting count models for the

number of loss events that are incurred in the retail banking and commercial banking lines, respectively. Let CorpKRI1, CorpKRI2, CbKRI1, CbKRI2, and CbKRI3 be the KRIs that you use in the count regression model of the commercial banking line, and let CorpKRI1, RbKRI1, and RbKRI2 be the KRIs that you use in the count regression model of the retail banking line. Examples of corporate-level KRIs (CorpKRI1 and CorpKRI2 in this example) are the ratio of temporary to permanent employees and the number of security breaches that are reported during a year. Examples of KRIs that are specific to the commercial banking business (CbKRI1, CbKRI2, and CbKRI3 in this example) are number of credit defaults, proportion of financed assets that are movable, and penalty claims against your bank because of processing delays. Examples of KRIs that are specific to the retail banking business (RbKRI1 and RbKRI2 in this example) are number of credit cards that are reported stolen, fraction of employees who have not undergone fraud detection training, and number of forged drafts and checks that are presented in a year.

Let the severity of each loss event in the commercial banking business be dependent on two KRIs, CorpKRI1 and CbKRI2. Let the severity of each loss event in the retail banking business be dependent on three KRIs, CorpKRI2, RbKRI1, and RbKRI3. Note that for each line of business, the set of KRIs that are used for the severity model is different from the set of KRIs that are used for the count model, although the two sets overlap. Further, the severity model for retail banking includes a new regressor (RbKRI3) that is not used for any of the count models. Such use of different sets of KRIs for count and severity models is typical of real-world applications.

Let the parameter estimates of the negative binomial and Poisson regression models be available in the data sets Work.CountEstEx2NB2 and Work.CountEstEx2Poisson, respectively. The following statements produce them by using the OUTEST= option in the respective PROC COUNTREG statements:

```
/* Simulate count data */
data cntdataex2(keep=line corpKRI1 corpKRI2
                  cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 numloss);
  call streaminit(12345);
  array cx{7} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2;
  array cbetaR{8} _TEMPORARY_ (0.35 1 0 0 0 0 0.5 0.25);
  array cbetaC{8} _TEMPORARY_ (0.9 0.75 0.3 0.1 0.25 0.5 0 0);

  alpha = 0.3;
  theta = 1/alpha;
  do obs=1 to 5000;
    do i=1 to dim(cx);
      cx(i) = rand('NORMAL');
    end;

    line = 'CommercialBanking';
    xbeta = cbetaC(1);
    do i=1 to dim(cx);
      xbeta = xbeta + cx(i) * cbetaC(i+1);
    end;
    Mu = exp(xbeta);
    p = theta/(Mu+theta);
    numloss = rand('NEGB',p,theta);
    output;

    line = 'RetailBanking';
    xbeta = cbetaR(1);
    do i=1 to dim(cx);
```

```

        xbeta = xbeta + cx(i) * cbetaR(i+1);
    end;
    lambda = exp(xbeta);
    numloss = rand('POISSON', lambda);
    output;
end;
run;

proc sort data=cntdataex2;
    by line;
run;

/* Fit negative binomial (p=2) regression model for each business line */
proc countreg data=cntdataex2 outest=countEstEx2NB2;
    by line;
    model numloss = corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                    rbKRI1 rbKRI2 / dist=negbin;
run;

/* Fit Poisson regression model for each business line */
proc countreg data=cntdataex2 outest=countEstEx2Poisson;
    by line;
    model numloss = corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                    rbKRI1 rbKRI2 / dist=poisson;
run;

```

NOTE: PROC CNTSELECT does not support the OUTEST= option, so unlike other examples that use PROC CNTSELECT to estimate count models, the preceding statements use PROC COUNTREG.

Let the parameter estimates of the best-fitting severity models, as determined by PROC SEVSELECT, be available in the data table mycas.SevEstEx2, as prepared by the following statements:

```

/* Simulate severity data */
data sevdataEx2(keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 lossValue);
    array sx{5} corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3;
    array sbetaC{6} _TEMPORARY_ (5 1 0 0.3 0 0);
    array sbetaR{6} _TEMPORARY_ (3.5 0 0.5 0 -0.8 0.6);
    if (_n_=1) then call streaminit(67890);

    set cntdataex2(keep=line numloss corpKRI1 corpKRI2 cbKRI2 rbKRI1);
    sigma = 1;
    alpha = 2.5;
    if (numloss > 0) then do;
        sx(5) = rand('NORMAL'); /* simulate rbKRI3 value */

        if (line='CommercialBanking') then do;
            /* lognormal */
            Mu = sbetaC(1);
            do i=1 to dim(sx);
                Mu = Mu + sx(i) * sbetaC(i+1);
            end;
            lossValue = exp(Mu) * rand('LOGNORMAL')**Sigma;
        end;
    else do;

```

```

        /* gamma */
        Mu = sbetaR(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbetaR(i+1);
        end;
        lossValue = quantile('Gamma', rand('UNIFORM'), Alpha, exp(Mu));
    end;
    output;
end;
run;

/* Load data into the CAS server */
data mycas.sevdataEx2;
    set sevdataEx2;
run;

/* Fit severity models for each business line */
proc sevselect data=mycas.sevdataEx2 outest=mycas.sevestEx2;
    by line;
    loss lossValue;
    scalemodel corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3;
    dist logn gamma;
run;

```

Now, consider that you want to estimate the distribution of the aggregate loss for a scenario, which is represented by a specific set of KRI values. The following DATA step illustrates one such scenario:

```

/* Generate a scenario data table for a single operating condition */
data singleScenario (keep=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3
                      rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do i=1 to dim(x);
        x(i) = rand('NORMAL');
    end;
    output;
run;

/* Load data into the CAS server */
data mycas.singleScenario;
    set singleScenario;
run;

```

The data table `mycas.SingleScenario` contains all the KRIs that are included in the count and severity models of both business lines. Note that if you standardize or scale the KRIs while fitting the count and severity models, then you must apply the same standardization or scaling method to the values of the KRIs that you specify in the scenario. In this particular example, all KRIs are assumed to be standardized.

The following DATA step uses the scenario in the data table `mycas.SingleScenario` to simulate 10,000 replications of the number of loss events that you might observe for each business line and writes the simulated counts to the variable `NumLoss` of the data table `mycas.LossCounts1`:


```

/* Simulate multiple replications of the number of loss events that
you can expect in the scenario being analyzed */
data lossCounts1 (keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 numloss);
  array cxR{3} corpKRI1 rbKRI1 rbKRI2;
  array cbetaR{4} _TEMPORARY_;
  array cxC{5} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3;
  array cbetaC{6} _TEMPORARY_;

  retain theta;
  if _n_ = 1 then do;
    call streaminit(5151);
    /* read count model estimates */
    set countEstEx2NB2(where=(line='CommercialBanking' and _type_='PARM'));
    cbetaC(1) = Intercept;
    do i=1 to dim(cxC);
      cbetaC(i+1) = cxC(i);
    end;
    alpha = _Alpha;
    theta = 1/alpha;

    set countEstEx2Poisson(where=(line='RetailBanking' and _type_='PARM'));
    cbetaR(1) = Intercept;
    do i=1 to dim(cxR);
      cbetaR(i+1) = cxR(i);
    end;
  end;

  set singleScenario;
  do iline=1 to 2;
    if (iline=1) then line = 'CommercialBanking';
    else line = 'RetailBanking';
    do repid=1 to 10000;
      /* draw from count distribution */
      if (iline=1) then do;
        xbeta = cbetaC(1);
        do i=1 to dim(cxC);
          xbeta = xbeta + cxC(i) * cbetaC(i+1);
        end;
        Mu = exp(xbeta);
        p = theta/(Mu+theta);
        numloss = rand('NEGB',p,theta);
      end;
      else do;
        xbeta = cbetaR(1);
        do i=1 to dim(cxR);
          xbeta = xbeta + cxR(i) * cbetaR(i+1);
        end;
        numloss = rand('POISSON', exp(xbeta));
      end;
      output;
    end;
  end;
run;

```

```

/* Load data into the CAS server */
data mycas.lossCounts1;
  set lossCounts1;
run;

```

The mycas.LossCounts1 data table contains the variable NumLoss in addition to the KRIs that the severity regression model uses, which PROC CCDM needs in order to simulate the aggregate loss.

Now, you are ready to estimate the aggregate loss distribution for each line of business by submitting the following PROC CCDM step, in which you specify the EXTERNALCOUNTS statement to request that external counts in the variable NumLoss of the DATA= data table be used for simulation of the aggregate loss:

```

/* Estimate the distribution of the aggregate loss for both
   lines of business by using the externally simulated counts */
proc ccdm data=mycas.lossCounts1 seed=13579 print=all
  severityest=mycas.sevestEx2;
  by line;
  externalcounts count=numloss;
  severitymodel logn gamma;
run;

```

Each observation in the mycas.LossCounts1 data table represents one replication of the external counts simulation process. For each such replication, the preceding PROC CCDM step makes as many severity draws from the severity distribution as the value of the NumLoss variable, and it adds together the severity values from those draws to compute one sample point of the aggregate loss. The severity distribution that is used for making the severity draws has a scale parameter value that is determined by the KRI values in the given observation and by the regression parameter values that are read from the mycas.SevEstEx2 data table.

The summary statistics and percentiles of the aggregate loss distribution for the commercial banking business, which uses the lognormal severity model, are shown in [Output 5.2.1](#). The “Input Data Summary” table indicates that each of the 10,000 observations in the BY group is treated as one replication and that a total of 19,028 loss events were produced by all the replications together. For the scenario in the data table mycas.SingleScenario, you can expect the commercial banking line to incur an average aggregate loss of 600 units, as shown in the “Sample Summary Statistics” table, and the chance that the loss will exceed 4,428 units is 0.5%, as shown in the “Sample Percentiles” table.

Output 5.2.1 Aggregate Loss Summary for Commercial Banking Line

The CCDM Procedure	
line=CommercialBanking	
Input Data Summary	
Name	LOSSCOUNTS1
Observations	10000
Valid Observations	10000
Replications	10000
Total Count	19028

Output 5.2.1 *continued***The CCDM Procedure**

Severity Model: Logn
Count Model: External

line=CommercialBanking

Sample Summary Statistics			
Mean	599.56756	Median	344.00869
Standard Deviation	786.88364	Interquartile Range	785.04322
Variance	619185.9	Minimum	0
Skewness	2.80654	Maximum	11176.8
Kurtosis	13.58752	Sample Size	10000

line=CommercialBanking

Sample Percentiles	
Percentile	Value
1	0
5	0
25	52.25800
50	344.00869
75	837.30121
95	2107.5
99	3637.3
99.5	4428.4
Percentile Method = 5	

For the retail banking line, which uses the gamma severity model, the “Sample Percentiles” table in [Output 5.2.2](#) indicates that the median operational loss of that line of business is about 70 units and the chance that the loss will exceed 378 units is about 1%.

Output 5.2.2 Aggregate Loss Percentiles for Retail Banking Line

line=RetailBanking

Sample Percentiles	
Percentile	Value
1	0
5	0
25	0
50	70.46309
75	142.15990
95	271.76555
99	378.32158
99.5	437.12308
Percentile Method = 5	

When you conduct simulation and estimation for a scenario that contains only one observation, you assume that the operating environment does not change over the period of time that is being analyzed. That assumption

might be valid for shorter durations and stable business environments, but often the operating environments change, especially if you are estimating the aggregate loss over a longer period of time. So you might want to include in your scenario all the possible operating environments that you expect to see during the analysis time period. Each environment is characterized by its own set of KRI values. For example, the operating conditions might change from quarter to quarter, and you might want to estimate the aggregate loss distribution for the entire year. You start the estimation process for such scenarios by creating a scenario data table. The following DATA step creates the data table `mycas.MultiConditionScenario`, which consists of four operating environments, one for each quarter:

```
/* Generate a scenario data table for multiple operating conditions */
data multiConditionScenario (keep=opEnvId corpKRI1 corpKRI2
    cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do opEnvId=1 to 4;
        do i=1 to dim(x);
            x(i) = rand('NORMAL');
        end;
        output;
    end;
run;

/* Load data into the CAS server */
data mycas.multiConditionScenario;
    set multiConditionScenario;
run;
```

All four observations of the data table `mycas.MultiConditionScenario` together form one scenario. When you simulate the external counts for such multi-entity scenarios, one replication consists of the possible number of loss events that can occur as a result of all four operating environments. In any given replication, some operating environments might not produce any loss event, or all four operating environments might produce some loss events. The following DATA step creates the data table `mycas.LossCounts2`, which contains, for each business line, 10,000 replications of the loss counts, and which identifies each replication by using the variable `Repld`:

```
/* Simulate multiple replications of the number of loss events that you can
    expect for all operating environments in the scenario being analyzed */
data lossCounts2 (keep=line opEnvId corpKRI1 corpKRI2 cbKRI2
    rbKRI1 rbKRI3 repid numloss);
    array cxR{3} corpKRI1 rbKRI1 rbKRI2;
    array cbetaR{4} _TEMPORARY_;
    array cxC{5} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3;
    array cbetaC{6} _TEMPORARY_;

    /* Read the count model estimates */
    retain theta;
    if _n_ = 1 then do;
        call streaminit(5151);
        set countEstEx2NB2(where=(line='CommercialBanking' and _type_='PARM'));
        cbetaC(1) = Intercept;
        do i=1 to dim(cxC);
            cbetaC(i+1) = cxC(i);
        end;
    end;
```

```

alpha = _Alpha;
theta = 1/alpha;

set countEstEx2Poisson(where=(line='RetailBanking' and _type_='PARM'));
cbetaR(1) = Intercept;
do i=1 to dim(cxR);
    cbetaR(i+1) = cxR(i);
end;
end;

/* Find the number of observations in the scenario data set */
nobs = 0;
do while(last=0);
    set multiConditionScenario end=last;
    nobs = nobs+1;
end;
nobstotal=nobs;

do iline=1 to 2;
    if (iline=1) then line = 'CommercialBanking';
    else line = 'RetailBanking';
    do repid=1 to 10000;
        do nobs=1 to nobstotal;
            set multiConditionScenario point=nobs;
            /* Draw from the appropriate count distribution */
            if (line = 'CommercialBanking') then do;
                xbeta = cbetaC(1);
                do i=1 to dim(cxC);
                    xbeta = xbeta + cxC(i) * cbetaC(i+1);
                end;
                Mu = exp(xbeta);
                p = theta/(Mu+theta);
                numloss = rand('NEGB',p,theta);
            end;
            else if (line = 'RetailBanking') then do;
                xbeta = cbetaR(1);
                do i=1 to dim(cxR);
                    xbeta = xbeta + cxR(i) * cbetaR(i+1);
                end;
                numloss = rand('POISSON', exp(xbeta));
            end;
            output;
        end;
    end;
end;
run;

```

To use the replication identifier variable with PROC CCDM, you need to partition the input data table of counts. The following DATA step loads the data set Work.LossCounts into a data table in your CAS session that is associated with the mycas CAS engine libref. The PARTITION= data table option partitions the data table such that a group of observations that have the same value for the variables line and RepId are located on the same worker node. The DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```

/* Load data into the CAS server and partition by the specified variables. */
data mycas.lossCounts2(partition=(line repid));
  set lossCounts2;
run;

```

Output 5.2.3 shows some observations of the data table mycas.LossCounts2 for each business line. For the first replication (Repld=1) of the commercial banking line, only operating environments 3 and 4 incur loss events; the other environments incur no loss events. For the second replication (Repld=2), all operating environments incur at least one loss event. For the first replication (Repld=1) of the retail banking line, operating environments 2, 3, and 4 incur two, one, and three loss events, respectively.

Output 5.2.3 Snapshot of the External Counts Data with Replication Identifier

line	opEnvld	corpKRI1	corpKRI2	cbKRI2	rbKRI1	rbKRI3	repid	numloss
CommercialBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	1	0
CommercialBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	1	0
CommercialBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	1	3
CommercialBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	1	1
CommercialBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	2	2
CommercialBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	2	5
CommercialBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	2	12
CommercialBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	2	12
RetailBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	1	0
RetailBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	1	2
RetailBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	1	1
RetailBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	1	3
RetailBanking	1	0.45224	0.40661	-0.33680	-1.08692	-2.20557	2	2
RetailBanking	2	-0.03799	0.98670	-0.03752	1.94589	1.22456	2	2
RetailBanking	3	-0.29120	-0.45239	0.98855	-0.37208	-1.51534	2	0
RetailBanking	4	0.87499	-0.67812	-0.04839	-1.44881	0.78221	2	1

You can now use this simulated count data to estimate the distribution of the aggregate loss that is incurred in all four operating environments by submitting the following PROC CCDM step, in which you specify the replication identifier variable Repld in the ID= option of the EXTERNALCOUNTS statement:

```

/* Estimate the distribution of the aggregate loss for both
   lines of business by using the externally simulated counts
   for the multiple operating environments */
proc ccdm data=mycas.lossCounts2 seed=13579 print=all
  severityest=mycas.sevestEx2;
  by line;
  externalcounts count=numloss id=repid;
  severitymodel logn gamma;
run;

```

Within each BY group, for each value of the variable Repld, one point of the aggregate loss sample is simulated by using the process that is described in the section “Simulation with External Counts” on page 143.

The summary statistics and percentiles of the distribution of the aggregate loss, which is the aggregate of the losses across all four operating environments, are shown in Output 5.2.4 for the commercial banking

line. The “Input Data Summary” table indicates that the BY group contains 10,000 replications and that a total of 145,721 loss events are generated across all replications. The “Sample Percentiles” table indicates that you can expect a median aggregate loss of 4,817 units and a worst-case loss, as defined by the 99.5th percentile, of 17,521 units from the commercial banking line when you combine losses from all four operating environments.

Output 5.2.4 Aggregate Loss Summary for the Commercial Banking Line in Multiple Operating Environments

The CCDM Procedure

line=CommercialBanking

Input Data Summary	
Name	LOSSCOUNTS2
Observations	40000
Valid Observations	40000
Replications	10000
Total Count	145721

line=CommercialBanking

Sample Percentiles	
Percentile	Value
1	771.75856
5	1507.8
25	3097.6
50	4816.6
75	7000.8
95	11346.5
99	15740.3
99.5	17520.5
Percentile Method = 5	

Example 5.3: Scenario Analysis with Rich Regression Effects and BY Groups

This example illustrates scenario analysis when frequency and severity models use regression models that contain classification and interaction effects. It also illustrates how you can analyze scenarios for multiple groups of observations in one PROC CCDM step without having to simulate counts externally.

The example in the section “[Scenario Analysis](#)” on page 105 encodes the discrete-valued, nominal (nonordinal) variables Gender, CarType, and Education as numerical variables that have an implied order. For example, a high school diploma is assigned a smaller number than an advanced degree. This method of forcing an order on otherwise nonordinal (categorical) variables is not natural and might lead to biased estimates. A more accurate approach is to treat such variables as classification variables that enter the statistical analysis or model not through their values but through their levels. For example, when you specify Education as a classification variable, the modeling process creates different parameters for the Education = ‘High School’ and Education = ‘Advanced Degree’ levels and estimates a regression coefficient for each. When you specify such variables in the CLASS statement of PROC CNTSELECT and PROC SEVSELECT, those procedures perform the appropriate levelization for you, which is the process of finding and transforming levels into

regression parameters. For more information, see the description of the CLASS statement in Chapter 15, “The SEVSELECT Procedure.”

In addition to specifying nominal variables as classification (CLASS) variables, you can include interaction effects in severity and frequency models. For example, you might want to evaluate how the distribution of losses that are incurred by a policyholder who has a college degree and drives an SUV differs from that of a policyholder who has an advanced degree and drives a sedan. You can do this by including an interaction between CarType and Education in your severity model. Similarly, if you want to evaluate how the number of losses that a policyholder incurs per year varies by the number of annual miles for different types of cars, you can include an interaction between CarType and AnnualMiles in your frequency model. Analyzing such a rich set of regression effects can help you make more accurate predictions about the frequency and severity distributions of losses. PROC CCDM is designed to use such rich models to simulate a more accurate distribution of the aggregate loss.

To start the scenario analysis process, let the following programming statements simulate the frequency and severity events that are affected by the regression effects:

```
/* Define a macro to simulate regressor data */
%macro generateRegvars;
    age = MAX(int(rand('NORMAL', 35, 15)),16)/50;

    if (rand('UNIFORM') < 0.5) then do;
        genderNum = 1; * female;
        gender = 'F';
    end;
    else do;
        genderNum = 2; * male;
        gender = 'M';
    end;

    if (rand('UNIFORM') < 0.7) then do;
        carTypeNum = 1; * sedan;
        carType = 'Sedan';
    end;
    else do;
        carTypeNum = 2; * SUV;
        carType = 'SUV';
    end;

    educationLevel = rand('UNIFORM');
    if (educationLevel < 0.5) then do;
        educationNum = 1; *high school graduate;
        education = 'High School';
    end;
    else if (educationLevel < 0.85) then do;
        educationNum = 2; *college graduate;
        education = 'College';
    end;
    else do;
        educationNum = 3; *advanced degree;
        education = 'Advanced Degree';
    end;
end;
```



```

annualmiles = MAX(1000, int(rand('NORMAL', 12000, 5000)))/5000;

carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */

income = MAX(15000, int(rand('NORMAL', educationNum*30000, 50000)))/100000;
%mend;

/* Simulate loss severity data */
%let npolicy=1000;

data losses(keep=region policyholderId age gender carType
              annualmiles education carSafety
              income lossAmount noloss);
  call streaminit(12345);
  array cx{6} age genderF milesSUV milesSedan educationAdv educationColl;
  array cbeta{7} _TEMPORARY_ (1 0.75 -1 -1.2 -0.6 0.5 0.75);

  array czx{4} age carTypeSUV educationAdv educationColl;
  array czbeta{5} _TEMPORARY_ (-0.75 -1 -1.2 0.6 0.7);

  array sx{8} _temporary_;
  array sbeta{9} _TEMPORARY_ (5 0.5 1.15 -0.75 -0.3 0.4 0.7 -0.5 -0.3);

  length gender $1 carType $8 education $16;

  alpha = 0.9; theta = 1/alpha;

do iregion=1 to 2;
  if (iregion = 1) then do;
    region = 'East';
    sigma = 0.5;
  end;
  else do;
    region = 'West';
    /* slightly change parameter estimates for this group */
    do i=1 to dim(cbeta);
      cbeta(i) = cbeta(i) + 0.1;
    end;
    do i=1 to dim(czbeta);
      czbeta(i) = czbeta(i) + 0.1;
    end;
    do i=1 to dim(sbeta);
      sbeta(i) = sbeta(i) + 0.05;
    end;
    sigma = 0.3;
    alpha = 1.6; theta = 1/alpha;
  end;

do policyholderId=1 to &npolicy;
  /* simulate policyholder and vehicle attributes */
  %generateRegvars;

  do i=1 to dim(sx);
    sx(i) = 0;
  end;
end;

```

```

end;
if (carTypeNum = 2) then do;
    sx(1) = 1;
    carTypeSUV = 1;
    milesSUV = annualmiles;
    milesSedan = 0;
end;
else do;
    carTypeSUV = 0;
    milesSUV = 0;
    milesSedan = annualmiles;
end;

if (genderNum = 1) then do;
    sx(2) = 1;
    genderF = 1;
end;
else genderF = 0;

sx(3) = carSafety;
sx(4) = income;
if (educationNum = 3 and carTypeNum = 2) then sx(5) = 1;
if (educationNum = 2 and carTypeNum = 2) then sx(6) = 1;
if (educationNum = 3 and carTypeNum = 1) then sx(7) = 1;
if (educationNum = 2 and carTypeNum = 1) then sx(8) = 1;

if (educationNum = 3) then do;
    educationAdv = 1;
    educationColl = 0;
end;
else do;
    educationAdv = 0;
    if (educationNum = 2) then educationColl = 1;
    else educationColl = 0;
end;

/* simulate number of losses incurred by this policyholder */
czxbeta = czbeta(1);
do i=1 to dim(czx);
    czxbeta = czxbeta + czx(i) * czbeta(i+1);
end;
phi = exp(czxbeta)/(1+exp(czxbeta));

if (rand('UNIFORM') < phi) then do;
    * zero process is selected with probability phi;
    numloss = 0;
end;
else do;
    * negbin process is selected with probability (1-phi);
    cxbeta = cbeta(1);
    do i=1 to dim(cx);
        cxbeta = cxbeta + cx(i) * cbeta(i+1);
    end;
    Mu = exp(cxbeta);

```

```

        numloss = rand('POISSON',Mu);
    end;

    /* simulate severity of each loss */
    if (numloss > 0) then do;
        noloss = 0;
        Mu = sbeta(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbeta(i+1);
        end;
        do iloss=1 to numloss;
            lossAmount = exp(Mu) * rand('LOGNORMAL')**Sigma;
            output;
        end;
    end;
    else do;
        noloss = 1;
        lossAmount = .;
        output;
    end;
end;
end;
run;

/* Load loss severity data into the CAS server */
data mycas.losses;
    set losses;
run;

/* Simulate loss counts data */
proc sort data=losses out=lossesByPolicy;
    by region policyholderId;
run;

data losscounts(drop=policyholderId carSafety lossAmount noloss);
    set lossesByPolicy;
    by region policyholderId;
    retain numloss zeroloss 0;
    if (noloss eq 1) then
        zeroloss = zeroloss + 1;
    else
        numloss = numloss + 1;
    if (last.policyholderId) then do;
        output;
        numloss = 0;
    end;
run;

/* Load loss counts data into the CAS server */
data mycas.losscounts;
    set losscounts;
run;

```

The following programming statements use the data tables that the preceding statements create to fit the

severity and count models that contain a certain set of regression effects:

```
proc sevselect data=mycas.losses(where=(lossAmount ne .))
    covout outstore=mycas.sevstore print=all;
    by region;
    loss lossAmount;
    class carType gender education;
    scalemodel carType gender carSafety income education*carType
        income*gender carSafety*income;
    selection method=backward(select=sbc) details=summary;
    dist logn burr;
run;

proc cntselect data=mycas.losscounts store=mycas.cstore;
    by region;
    class gender carType education;
    model numloss = age income gender carType*annualmiles
        education / dist=poisson;
    zeromodel numloss ~ age income carType education;
    selection method=backward(select=sbc) details=summary;
run;
```

Note the following points about these statements:

- The data tables are organized in groups of observations that represent data from two regions, East and West. You can analyze both groups at the same time by specifying the BY statement with **Region** as the BY variable. The observations in the input data table do not need to be sorted by the BY variables.
- Both severity and count models use three CLASS variables. The severity model includes three interaction effects (Education*CarType, Income*Gender, and CarSafety*Income) and four main effects.
- The count model is a zero-inflated model, which is essentially a mixture of two models: a model to estimate the occurrence of zero loss events and a model to estimate nonzero counts. The model for zero counts, which is specified in the ZEROMODEL statement, is a regression model that has four main effects and uses the default logistic link function. The model for nonzero counts, which is specified in the MODEL statement, is a Poisson model that has one interaction effect (CarType*AnnualMiles) and four main effects.
- Both the PROC SEVSELECT and PROC CNTSELECT steps use the SELECTION statement to perform automatic effect selection. In particular, they use the backward selection method to choose a subset of regression effects that maximize the Schwarz Bayesian information criterion (SBC). PROC SEVSELECT conducts the effect selection process independently for each distribution.
- The PROC SEVSELECT step uses the OUTSTORE= option to store the parameter estimates in an item store. When your scale regression model contains classification or interaction effects, you must store the parameter estimates in an item store instead of storing them in an OUTEST= data table because PROC CCDM cannot obtain the necessary information about classification or interaction effects from an OUTEST= data table.

For severity models, you can inspect the “All Fit Statistics” table for each BY group to decide which severity distribution has the best fit for the data in that BY group. When you specify the SELECTION statement, each

row in this table shows the fit statistics of the scale regression model that contains only the final selected set of regression effects for the candidate distribution of that row. The tables in [Output 5.3.1](#) indicate that the model that uses the lognormal distribution is the best according to the majority of fit statistics for both regions, so you can just specify the LOGN distribution in the SEVERITYMODEL statement of the subsequent PROC CCDM step. In cases where different distributions are deemed best for different BY groups, you have two alternatives. The first alternative is to run one PROC CCDM step with the SEVERITYMODEL statement that specifies the best distributions from all BY groups and to later process the results of only the best severity distribution in each BY group. The second alternative is to run separate PROC CCDM steps for each BY group by specifying the best severity distribution for that BY group in the SEVERITYMODEL statement. The latter alternative requires more coding effort but it is more efficient than the first alternative, which runs the analysis for all best severity distributions for all BY groups.

Also note that in some cases, you might see that the likelihood-based fit statistics ($-2 \log$ likelihood, AIC, AICC, and BIC) mark one distribution as the best distribution and the EDF-based statistics (KS, AD, and CvM) mark another distribution as the best distribution. In such cases, it is recommended that you conduct aggregate loss simulation by using both severity distributions and compare the summary statistics and percentiles of resulting aggregate loss distributions.

Output 5.3.1 Comparison of Final Models of Severity Distributions

The SEVSELECT Procedure

Burr Distribution Selected Model

region=East

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	8715*	8735*	8735*	8782*	2.68389*	37.94117	2.64223*
Burr	8721	8743	8743	8794	2.76081	33.56827*	2.92479

Asterisk (*) denotes the best model in the column.

region=West

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	12989*	13009*	13009*	13060*	6.34841	305.19555	15.48743*
Burr	13004	13026	13026	13082	6.31408*	233.46457*	15.52778

Asterisk (*) denotes the best model in the column.

If you want to examine the summary of the effect selection process and the final selected set of regression effects for a severity distribution, you can examine the “Selection Summary” and “Selected Effects” tables as shown in [Output 5.3.2](#). These tables indicate that the best scale regression model for the lognormal distribution in the region=East group is obtained after removing the carSafety*income and income*gender effects.

Output 5.3.2 Summary of the Effect Selection Process for LOGN Severity Model for Region=East**The SEVSELECT Procedure****Logn Distribution
Selection Details**

region=East

Selection Summary			
Step	Effect Removed	Number Effects In	SBC
0		8	8793.6358
1	carSafety*income	7	8787.0918
2	income*gender	6	8781.8787*
3	income	5	8807.5961
4	carType*education	4	8921.5365
* Optimal Value Of Criterion			

region=East

Selected Effects: Intercept carType gender carSafety income carType*education

You can examine a similar summary of the selection process that the CNTSELECT procedure follows to select the best effect subset for the count distribution, as shown in [Output 5.3.3](#). The best model for the zero-inflated Poisson (ZIP) distribution in the region=East group is obtained after removing the education, age, and income effects from the model for the zero counts and removing the income effect from the model for nonzero counts. So the final model contains only the carType effect in the zero-model and age, income, gender, carType*annualmiles, and education effects in the nonzero-model.

Output 5.3.3 Summary of the Effect Selection Process for ZIP Count Model for Region=East**The CNTSELECT Procedure**

region=East

Selection Summary			
Step	Effect Removed	Number Effects In	SBC
0		11	2099.9659
1	Inf_education	10	2092.2022
2	Inf_age	9	2085.4951
3	Inf_income	8	2079.0596
4	income	7	2074.1768*
5	Inf_carType	6	2074.7483
6	age	5	2101.6437
7	education	4	2162.7788
* Optimal Value Of Criterion			

PROC SEVSELECT conducts a separate effect selection analysis for each severity distribution within each BY group, so scale regression models for different severity distributions in the same BY group can contain different final sets of selected effects. Similarly, scale regression models for the same severity distribution in different BY groups can contain different final sets of selected effects. PROC CNTSELECT also conducts a

separate effect selection analysis for each BY group, so the final sets of selected effects in the count regression model can be different for different BY groups.

After the CNTSELECT and SEVSELECT procedures have selected the best frequency and severity models, it is time to estimate the distribution of the aggregate loss by using the CCDM procedure. Given that the models depend on regressors, first you need to create the scenario data table, which must contain the final set of regressors that are used in both the severity model and the frequency model. Even if your models contain classification or interaction effects, your scenario data table needs to contain only the columns for individual variables of the effects. PROC CCDM internally performs *levelization* of each observation, which is the process of expanding the variable values to match them with the parameters of each effect. For example, you do not need to create multiple columns for the parameters of the carType*education effect. You just need to create two columns for carType and education. PROC CCDM internally creates the six columns that correspond to the six parameters that are associated with the carType*education effect. However, you must make sure that the values of each classification variable, such as carType, in the scenario data table belong to the same set of values that appear in the input data tables of PROC SEVSELECT and PROC CNTSELECT.

A typical scenario for an insurance application usually consists of a large number of policyholders. However, for illustration purposes, this example uses a small scenario of only a few policyholders per region that is generated by the following DATA step and shown in [Output 5.3.4](#):

```
/* Create scenario for internal count simulation */
data scenario(keep=region policyholderId age gender
               carType annualmiles education carSafety income);
  call streaminit(456);

  length region $4 gender $1 carType $8 education $16;

  do iregion=1 to 2;
    if (iregion = 1) then do;
      region = 'East';
      npolicy = 3;
    end;
    else do;
      region = 'West';
      npolicy = 5;
    end;
    do policyholderId = 1 to npolicy;
      %generateRegvars;
      output;
    end;
  end;
run;

/* Load scenario data into the CAS server */
data mycas.scenario;
  set scenario;
run;
```

Output 5.3.4 Data Table mycas.Scenario for BY-Group Processing

Obs	region	gender	carType	education	age	annualmiles	carSafety	income
1	East	F	SUV	High School	1.16	2.1540	0.29288	0.26090
2	East	F	Sedan	High School	0.86	2.3978	0.69844	0.15000
3	East	F	Sedan	Advanced Degree	0.78	1.9926	0.59421	0.58808
4	West	M	Sedan	College	0.82	1.8550	0.66849	0.15000
5	West	M	SUV	College	0.40	3.6240	0.23194	1.25274
6	West	M	Sedan	High School	0.62	3.6162	0.86477	0.42597
7	West	F	Sedan	College	0.32	3.4598	0.66294	0.36132
8	West	M	Sedan	Advanced Degree	0.90	3.2580	0.37172	0.15000

The following PROC CCDM step simulates the aggregate losses for the scenario in the data table mycas.Scenario by each region:

```
proc ccdm data=mycas.scenario nreplicates=10000 seed=123 print=all
    severitystore=mycas.sevstore countstore=mycas.cstore
    nperturb=30;
    by region;
    severitymodel logn;
    outsum out=mycas.agglossStats mean stddev skewness kurtosis
    pctlpts=(90 97.5 99.5);
run;
```

The BY statement must contain the same set of variables and in the same order as the variables in the BY statements that you specify in PROC SEVSELECT and PROC CNTSELECT. The **SEVERITYSTORE=** and **COUNTSTORE=** options specify the item stores that contain the effect information and parameter estimates of the severity and count models, respectively, for both BY groups. The count item store that PROC CNTSELECT creates always contains the covariance estimates of the count model parameters, and the COVOUT option in the PROC SEVSELECT step ensures that the severity item store includes the covariance estimates. Availability of these covariance estimates makes the perturbation analysis that the NPERTURB= option requests more accurate.

One of the main points that this example illustrates is that irrespective of the complexity of the regression effects in your frequency or severity models, you do not have to do any extra work to convey the effect information to PROC CCDM, except to specify the item stores. The frequency and severity item stores contain all the necessary information that PROC CCDM needs for computing the values of the regressor-dependent parameters of the underlying frequency and severity distributions before making random draws from those distributions. When you use the BY and SELECTION statements in PROC CNTSELECT and PROC SEVSELECT, the respective item stores contain the information about the final selected regression effects for each distribution in each BY group. PROC CCDM uses this information to process only the regression parameters that correspond to the selected regression effects for each distribution in each BY group.

Output 5.3.5 and Output 5.3.6 show the summary of the perturbation analysis for the two regions. You can deduce that for the collection of three policyholders in the East region of the specified scenario, the 97.5th percentile of their collective aggregate loss is $5,232 \pm 448$ units, and for the collection of five policyholders in the West region of the specified scenario, the 99.5th percentile of their collective aggregate loss is $1,944 \pm 97$.

Output 5.3.5 Aggregate Loss Simulation Results for Region=East**The CCDM Procedure**

Severity Model: Logn
Count Model: ZIP

region=East

**Sample Percentile Perturbation
Analysis**

Percentile	Estimate	Standard Error
1	0	0
5	0	0
25	0	0
50	578.23143	68.90165
75	2338.3	241.58977
90	3694.0	338.64518
95	4508.1	396.22789
97.5	5231.8	447.58106
99	6118.2	498.74995
99.5	6727.4	530.48825

**Number of Perturbed
Samples = 30**

Size of Each Sample = 10000

Output 5.3.6 Aggregate Loss Simulation Results for Region=West

region=West

**Sample Percentile Perturbation
Analysis**

Percentile	Estimate	Standard Error
1	23.83457	33.34455
5	172.44406	66.32633
25	477.45805	69.43476
50	715.60190	69.54735
75	981.50423	73.02344
90	1252.9	77.04198
95	1429.6	79.56728
97.5	1594.0	84.24221
99	1793.8	87.77441
99.5	1944.3	96.77442

**Number of Perturbed
Samples = 30**

Size of Each Sample = 10000

Example 5.4: Illustrating Pure Premium and Custom Simulation Modes

The CCDM procedure supports simulation modes other than the default collective risk simulation mode. This example illustrates how you can use those modes to simulate samples for different types of aggregate losses. Suppose you are an actuary at an insurance company and you want to estimate the distribution of the pure premium for the claims that you expect to be filed by the policyholders in a particular time period. The pure premium method of estimating claims cost is useful for certain types of peril. Assume that you have estimated a simple severity distribution model for the average severity of each claim by submitting the following statements, where the data table `mycas.ClaimSev` contains a simulation of historic average severity data:

```
/* Simulate average severity data and load them into the CAS server */
data claimsev(keep=averagesev);
  call streaminit(789);
  label averagesev='Average severity per claim';
  do n = 1 to 10000;
    averagesev = rand('Weibull', 3, 500);
    output;
  end;
run;
data mycas.claimsev;
  set claimsev;
run;

/* Fit severity models and write estimates to an item store */
proc sevselect data=mycas.claimsev crit=sbc covout outstore=mycas.sevstore;
  loss averagesev;
  dist gamma igauss logn weibull;
run;
```

Output 5.4.1 Comparison of Severity Distributions for Average Severity

The SEVSELECT Procedure

Model Selection			
Distribution	Converged	SBC	Selected
Gamma	Yes	130839	No
Igauss	Yes	132718	No
Logn	Yes	132081	No
Weibull	Yes	130163	Yes

The “Model Selection” table in [Output 5.4.1](#) shows that the Weibull distribution is the best-fitting distribution for the average severity. Next, assume that you have used a time-series analysis method to create a model for average number of claims that you expect to process in a particular time period. Let the average frequency follow a normal distribution $\mathcal{N}(5000, 100)$, which implies that the point forecast of the average frequency for the time period of interest is 5,000 with an associated standard error of 100. To analyze the distribution of the pure premium, which is the product of average severity and average frequency, you can use the `SIMULATIONMODE=PUREPREMIUM` option in PROC CCDM as shown in the following statements:

```
proc ccdm severitystore=mycas.sevstore simulationmode=purepremium
    seed=135 nreplicates=500000 print=all plots=density;
    severitymodel weibull;
    countmodel normal(5000, 100);
run;
```

The COUNTMODEL statement specifies the distribution of the average frequency. The SEVERITYMODEL statement requests that only the Weibull distribution be read from the SEVERITYSTORE= item store. The PRINT=ALL option displays the quantitative summary of the pure premium distribution, which is shown in the “Sample Summary Statistics” and “Sample Percentiles” tables in [Output 5.4.2](#). The PLOTS= option creates a plot of the nonparametric estimate of the probability density function (PDF) of the pure premium distribution, which is shown in [Output 5.4.3](#).

Output 5.4.2 Summary Statistics and Percentiles of the Pure Premium Distribution

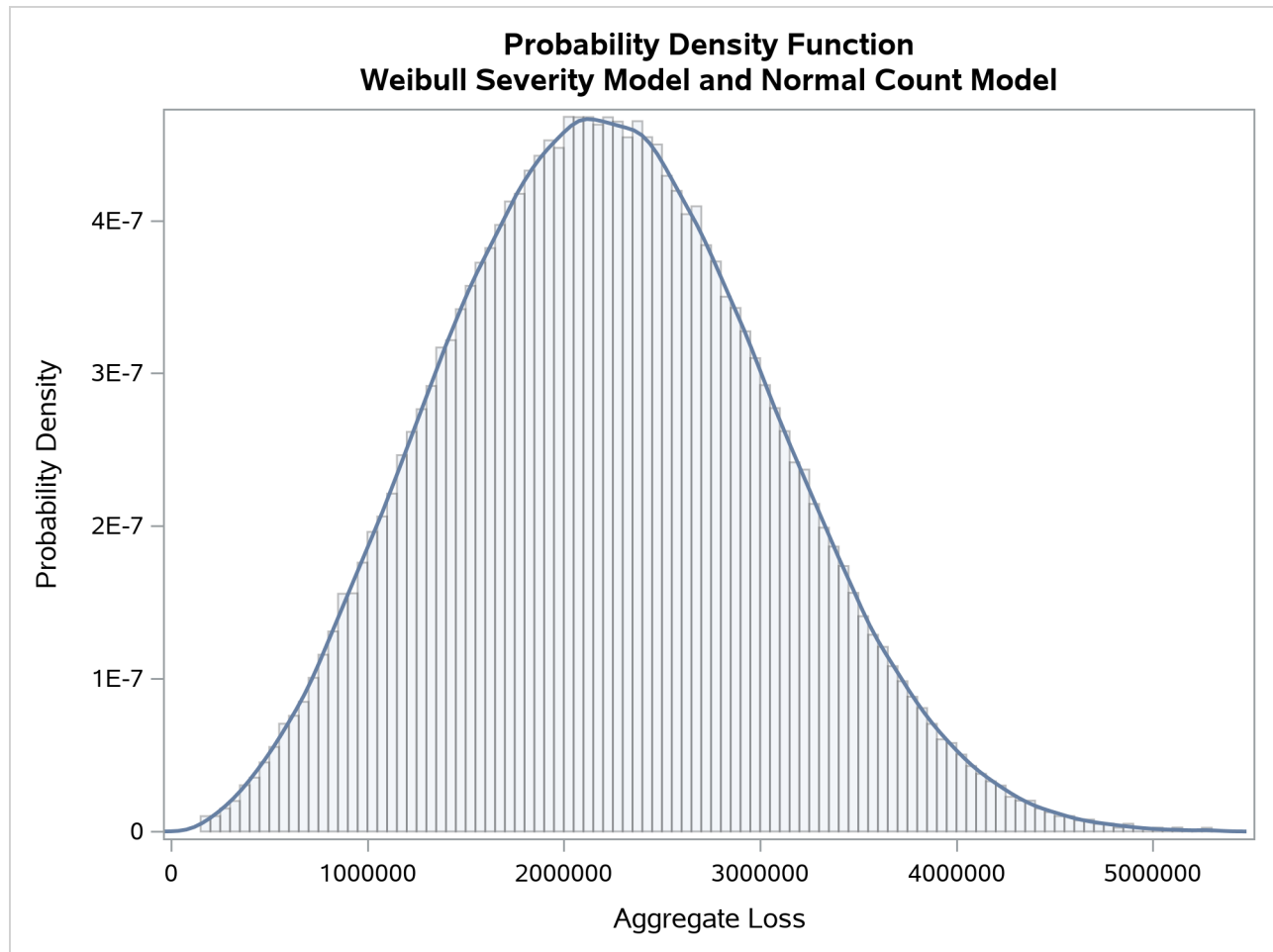
The CCDM Procedure

Severity Model: Weibull
Count Model: Normal(5000,100)

Sample Summary Statistics			
Mean	2226362.8	Median	2205122.2
Standard Deviation	817503.2	Interquartile Range	1144968.3
Variance	6.683115E11	Minimum	38283.8
Skewness	0.18161	Maximum	6046468.1
Kurtosis	-0.26273	Sample Size	500000

Sample Percentiles	
Percentile	Value
1	527178.5
5	916805.7
25	1638583.0
50	2205122.2
75	2783551.3
95	3613722.4
99	4176629.8
99.5	4376132.1

Percentile
Method = 5

Output 5.4.3 Nonparametric PDF Plot of the Pure Premium Distribution

Now consider a situation where the average frequency is defined as the average number of claims *per policy* and you have estimated a separate distribution model for the number of policies, denoted by `nPolicy`, that you expect to be in force for the particular time period. If you want to estimate a distribution of the total expected loss across all policies according to the pure premium method, then you need to multiply the pure premium by an estimate of `nPolicy`. Because `nPolicy` is stochastic, the simulation process needs to make a random draw from its distribution. Assuming that the average number of claims per policy follows a standard negative binomial distribution with parameters $p = 0.75$ and $n = 5$ and that `nPolicy` follows a lognormal distribution with parameters $\mu = 10$ and $\sigma = 0.35$, the following PROC CCDM step simulates a sample of the total expected loss, `tLoss`, across all policies:

```
proc ccdm severitystore=mycas.sevstore simulationmode=purepremium
    seed=135 nreplicates=500000 print=all adjustedseverity=tloss;
    severitymodel weibull;
    countmodel negbin(0.75, 5);
    simulatedsymbol npolicy ~ lognormal(10,0.35);
    tloss = _sev_ * npolicy;
run;
```

The COUNTMODEL statement specifies that the frequency follows the `Negbin(0.75, 5)` distribution. The SIMULATEDSYMBOL statement specifies the stochastic variable `nPolicy`. The severity adjustment program

contains just one statement in this example, and it computes tLoss by multiplying the average severity value in the `_SEV_` placeholder symbol by the simulated value of `nPolicy`. PROC CCDM supplies the simulated values of both the `_SEV_` and `nPolicy` symbols to your adjustment program. In the pure premium simulation mode, for each replication, PROC CCDM multiplies the tLoss value that your adjustment program returns by the frequency value that it draws from the `Negbin(0.75, 5)` distribution to compute the final value of tLoss for that replication. The distribution of tLoss is summarized in the tables shown in [Output 5.4.4](#).

Output 5.4.4 Summary Statistics and Percentiles of the Total Loss Distribution

The CCDM Procedure

Severity Model: Weibull
Count Model: NegativeBinomial(0.75,5)

Summary Statistics for tloss			
Mean	17353145	Median	12043346
Standard Deviation	19831070	Interquartile Range	21217543
Variance	3.932713E14	Minimum	0
Skewness	2.41337	Maximum	422595056
Kurtosis	10.60798	Sample Size	500000

Percentiles for tloss	
Percentile	Value
1	0
5	0
25	3059651.5
50	12043346
75	24277194
95	55310963
99	91050716
99.5	108085612

Percentile
Method = 5

If you want to further account for the loss adjustment expenses (LAE) in your simulations and if you have estimated that the average LAE per policy follows a normal distribution $\mathcal{N}(100, 10)$, then the total loss and expense, denoted by `tLossExp`, is computed as `tLossExp = ((severity/claim) * (claims/policy) + LAE)*nPolicy`. Because this does not fit the standard pure premium formula, you need to use the custom simulation mode by specifying the `SIMULATIONMODE=CUSTOM` option in the PROC CCDM statement as illustrated by the following statements:

```
proc ccdm severitystore=mycas.sevstore simulationmode=custom nperturb=50
    seed=135 nreplicates=500000 print=all adjustedseverity=(tloss tlossexp)
    plot(nadjsev=2)=(conditionaldensity(leftq=0.2 rightq=0.9));
    severitymodel weibull;
    countmodel negbin(0.75(se=0.05), 5(se=0.5));
    simulatedsymbol npolicy ~ lognormal(10,0.35) lae ~ normal(100,10);
    tloss = _sev_ * _freq_ * npolicy;
    tlossexp = (_sev_ * _freq_ + lae) * npolicy;
run;
```

The `SIMULATEDSYMBOL` statement now contains the specification for the stochastic variable `LAE`. For the custom simulation mode, `PROC CCDM` analyzes only the symbols that you specify in the `ADJUSTED-SEVERITY=` option; unlike collective risk or pure premium simulation modes, it does not create a sample for the unadjusted aggregate loss. However, you can compute the pure premium loss in the custom mode yourself as illustrated by the programming statement that computes `tLoss` in the preceding `PROC CCDM` step.

For each replication, `PROC CCDM` draws one random value each from the probability distributions of severity, frequency, `LAE`, and `nPolicy`, and it supplies those values to your severity adjustment program in appropriate symbols (the `_FREQ_` and `_SEV_` symbols contain frequency and severity values, respectively). The values of `tLoss` and `tLossExp` symbols that your severity adjustment program returns are added to the `tLoss` and `tLossExp` samples, respectively. The summary of the `tLossExp` sample thus generated is shown in [Output 5.4.5](#). As expected, the mean, median, and percentile estimates of total loss and expenses are higher than the corresponding estimates for the total loss distribution in [Output 5.4.4](#). The conditional PDF plot in [Output 5.4.6](#) shows the nonparametric estimate of the density function in different ranges of values of `tLossExp`. The left-tail plot indicates that there are at least 20% zero-valued observations in the sample of `tLossExp`, which implies that many policies do not result in a loss event. The right-tail plot starts at the 90th percentile and the density is nonzero for large losses, which implies that the distribution of `tLossExp` has a heavy tail.

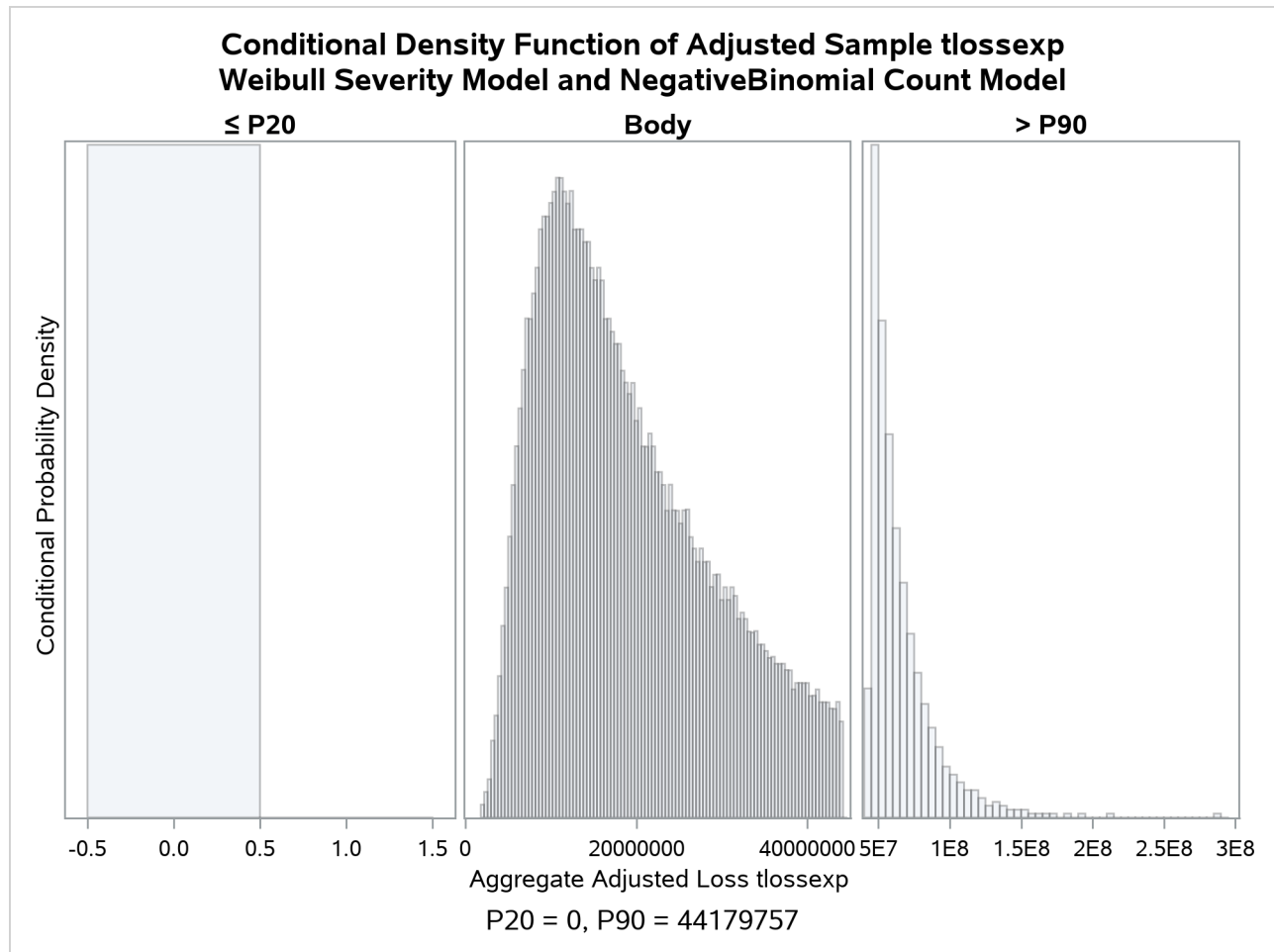
Output 5.4.5 Summary Statistics and Percentiles of the Total Loss and Expenses Distribution

The CCDM Procedure

Severity Model: Weibull
Count Model: NegativeBinomial(0.75,5)

Summary Statistics for <code>tlossexp</code>			
Mean	19125595	Median	14244882
Standard Deviation	20614384	Interquartile Range	22135331
Variance	4.249528E14	Minimum	0
Skewness	2.27570	Maximum	374747745
Kurtosis	9.79975	Sample Size	500000

Percentiles for <code>tlossexp</code>	
Percentile	Value
1	0
5	0
20	0
25	4600707.0
50	14244882
75	26736038
90	44179757
95	58186717
99	93911815
99.5	111999737
Percentile Method = 5	

Output 5.4.6 Nonparametric Density Estimate of the Total Loss and Expenses Distribution

The NPRTURB= option in the preceding PROC CCDM step requests perturbation analysis, which requires the variability estimates of distribution parameters. For the severity distribution, PROC CCDM reads the variability estimates from the SEVERITYSTORE= item store. For the externally estimated count distribution, you need to specify the standard errors of its parameters by using the SE= option after each parameter value in the COUNTMODEL statement, as shown in the preceding PROC CCDM step. For each perturbed sample, PROC CCDM uses perturbed values of the severity and frequency distribution parameters, but it does not perturb the parameters of the simulated symbols. The perturbation summary for the total loss and expense distribution is shown in [Output 5.4.7](#).

Output 5.4.7 Perturbation Analysis of the Total Loss and Expenses Distribution

Sample Perturbation Analysis for tlossexp		
Statistic	Estimate	Standard Error
Mean	18843608	4234253.8
Standard Deviation	20412647	3168495.5
Variance	4.26716E14	1.29673E14
Skewness	2.28819	0.04059
Kurtosis	9.96243	0.59615
Number of Perturbed Samples = 50		
Size of Each Sample = 500000		

Percentile Perturbation Analysis for tlossexp		
Percentile	Estimate	Standard Error
1	0	0
5	0	0
20	1751419.7	2846171.6
25	3501757.6	3860198.2
50	13964518	3200855.3
75	26388579	5332399.7
90	43670880	8161777.2
95	57596615	10307023
99	93281229	15780831
99.5	110240289	18304700
Number of Perturbed Samples = 50		
Size of Each Sample = 500000		

References

- Fisher, R. A. (1973). *Statistical Methods for Research Workers*. 14th ed. New York: Hafner Publishing.
- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.

Chapter 6

The CCOPULA Procedure

Contents

Overview: CCOPULA Procedure	208
PROC CCOPULA Features	208
PROC CCOPULA Compared with Other SAS Procedures	208
Using CAS Sessions and CAS Engine Librefs	209
Getting Started: CCOPULA Procedure	210
Syntax: CCOPULA Procedure	212
Functional Summary	212
PROC CCOPULA Statement	213
BY Statement	214
DEFINE Statement	214
DISPLAY Statement	216
DISPLAYOUT Statement	217
FIT Statement	218
SIMULATE Statement	222
VAR Statement	223
VIEWSTORE Statement	223
Details: CCOPULA Procedure	224
Sklar's Theorem	224
Dependence Measures	225
Data Processing and Fitting	226
Normal Copula	227
Student's t copula	228
Archimedean Copulas	230
Canonical Maximum Likelihood Estimation (CMLE)	236
Exact Maximum Likelihood Estimation (MLE)	237
Calibration Estimation	237
Nonlinear Optimization Options	238
Displayed Output	238
OUTPSEUDO= and OUTUNIFORM= Data Tables	240
ODS Table Names	240
ODS Graph Names	241
Examples: CCOPULA Procedure	242
Example 6.1: Simulating Default Times	242
Example 6.2: Copula Analysis of Stock Returns	244
References	248

Overview: CCOPULA Procedure

A multivariate distribution for a random vector contains a description of both the marginal distributions and their dependence structure. A copula approach to formulating a multivariate distribution provides a way to isolate the description of the dependence structure from the marginal distributions. A copula is a function that combines marginal distributions of variables into a specific multivariate distribution. All of the one-dimensional marginals in the multivariate distribution are the cumulative distribution functions of the factors. Copulas help perform large-scale multivariate simulation from separate models, each of which can be fitted using different, even nonnormal, distributional specifications.

The CCOPULA procedure enables you to fit multivariate distributions or copulas from a specified sample data table. You can do the following:

- estimate the parameters for a specified copula type
- simulate a given copula

PROC CCOPULA Features

The CCOPULA procedure enables you to simulate or fit (or both) a specified copula, and it supports the following types of copulas:

- normal copula
- t copula
- Clayton copula
- Gumbel copula
- Frank copula

PROC CCOPULA Compared with Other SAS Procedures

The CCOPULA procedure provides copula simulation and model fitting functionality comparable to that of the HPCOPULA and COPULA procedures in SAS/ETS software.

PROC CCOPULA Compared with the HPCOPULA Procedure

The functionality of the CCOPULA procedure closely resembles that of the HPCOPULA procedure, which is a high-performance procedure. The CCOPULA procedure is the next generation of the HPCOPULA procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CCOPULA procedure provides all the functionality of the HPCOPULA procedure.

PROC CCOPULA Compared with the COPULA Procedure

The CCOPULA procedure is specifically designed to operate on SAS Viya and performs computations in multiple threads on multiple machines. By contrast, the COPULA procedure runs only in a single thread on a single machine.

This release of the CCOPULA procedure contains some, but not all, of the functionality present in the COPULA procedure. The COPULA procedure provides the following features, which are not present in the CCOPULA procedure:

- hierarchical Archimedean copulas
- tail dependence plots

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CCOPULA Procedure

The following example illustrates the use of PROC CCOPULA. The data are daily returns on several major stocks. The main purpose of this example is to estimate the joint distribution of stock returns and then use this distribution to simulate a new sample of specified size.

Figure 6.1 shows the first 10 observations of the daily stock return data set.

Figure 6.1 First 10 Observations of Daily Returns

Obs	date	ret_msft	ret_ko	ret_ibm	ret_duk	ret_bp
1	01/03/2008	0.004182	0.010367	0.002002	0.003503	0.019114
2	01/04/2008	-0.027960	0.001913	-0.035861	-0.000582	-0.014536
3	01/07/2008	0.006732	0.023607	-0.010671	0.025611	0.017922
4	01/08/2008	-0.033435	0.004239	-0.024610	-0.002838	-0.016049
5	01/09/2008	0.029560	0.026680	0.007301	0.010814	-0.027078
6	01/10/2008	-0.003054	0.004441	0.016414	-0.001689	-0.004395
7	01/11/2008	-0.012255	-0.027346	-0.022546	-0.012408	-0.018473
8	01/14/2008	0.013958	0.008418	0.053857	0.003427	0.001166
9	01/15/2008	-0.011318	-0.010851	-0.010689	-0.017075	-0.040925
10	01/16/2008	-0.022587	-0.015021	-0.001955	0.002316	-0.021336

The following statements fit a normal copula to the returns data (by using the FIT statement) and create a new SAS item store that contains parameter estimates of the model along with a complete description of the model (including variable names, copula type, and so on). The VAR statement specifies the list of variables, which in this case are the daily returns of stocks from five large companies.

```
/* Copula estimation */
proc ccopula data = mycas.returns;
  var ret_ibm ret_msft ret_bp ret_ko ret_duk;
  fit normal / store=mycas.estimates_norm;
run;
```

The first table in Figure 6.2 shows some general information about the copula fitting procedure: the number of observations, the name of the input data table, the type of model, and the correlations matrix.

Figure 6.2 Copula Estimation: Fit Summary and Correlations Matrices

The CCOPULA Procedure

Model Fit Summary	
Number of Observations	603
Data Set	RETURNS
Copula Type	Normal
Fit Method	MLE
Marginals	Empirical
Empirical Approximation	Adaptive Binning
Adaptive Refinement Resolution	1000
Adaptive Interpolation Method	Linear
Number of Adaptive Iterations	4

Then, in the following statements, the SIMULATE statement specifies a copula named COP, and the RESTORE= option specifies that the item store Estimates_Norm be used as the source for the variable names, copula type, and parameters. The NDRAWS=500 option in the SIMULATE statement generates 500 observations from the normal copula. The OUTUNIFORM= option specifies the name of the data table to contain the simulated sample with uniform marginal distributions. Note that this syntax does not require the DATA= option and that neither the VAR statement nor the DEFINE statement is needed.

```
/* Ccopula simulation of uniforms */
proc ccopula;
  simulate cop / restore=mycas.estimates_norm
              ndraws      = 500
              seed        = 1234
              outuniform  = mycas.simulated_uniforms;
run;
```

The simulated data are contained in the new data table, mycas.Simulated_Uniforms.

The preceding sequence of PROC CCOPULA usage—first fit, then simulate using estimated parameters—is a legitimate sequence but has a limitation in that the second PROC CCOPULA call does not generate the sample according to the empirical distribution of the raw data. It generates only marginally uniform series.

In the following statements, the FIT statement fits a t copula to the returns data and at the same time simulates the sample according to empirical marginal distributions. Because there is no VAR or DEFINE statement, PROC CCOPULA assumes that you want to initialize the simulation by using the information stored in an item store. Because the RESTORE= option is not used in the SIMULATE statement, PROC CCOPULA uses the item store that is created when the FIT statement is executed.

```
/* Ccopula estimation and simulation */
proc ccopula data = mycas.returns;
  var ret_ibm ret_msft ret_bp ret_ko ret_duk;
  fit T / store=mycas.estimates_t;
  simulate / ndraws = 1000
            seed    = 1234
            outuniform = mycas.simuni;
run;
```

The output of the statements is similar in structure to the output displayed in Figure 6.2, with the addition of parameter estimates and inference statistics that are specific to the copula model as shown in Figure 6.3. For a t copula, the degrees of freedom are displayed (as in Figure 6.3); for Archimedean copulas, the parameter “theta” is displayed; and for a normal copula, this table is not printed.

Figure 6.3 Copula Estimation: Specific Parameter Estimates

The CCOPULA Procedure

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Approx Pr > t
DF	3.659299	0.320796	11.41	<.0001

The simulated data are contained in the new data table, mycas.Simuni.

Syntax: CCOPULA Procedure

The following statements are available in the CCOPULA procedure:

```

PROC CCOPULA options ;
  BY variables ;
  DEFINE name copula-type <(parameter-value-options ...) > ;
  DISPLAY <table-list> </ options> ;
  DISPLAYOUT table-spec-list </ options> ;
  FIT type <NAME=name > <INIT=(parameter-value-options) > / options ;
  SIMULATE < copula-name-list > / options ;
  VAR variables ;
  VIEWSTORE / options ;

```

Functional Summary

Table 6.1 summarizes the statements and options that the CCOPULA procedure uses.

Table 6.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CCOPULA	DATA=
Specifies the input data table that contains the correlations matrix for elliptical copulas	DEFINE	CORR=
Specifies the input data table that contains the correlations matrix defined in Kendall's tau for elliptical copulas	DEFINE	KENDALL=
Specifies the input data table that contains the correlations matrix defined in Spearman's rho for elliptical copulas	DEFINE	SPEARMAN=
Copula Fitting Options		
Specifies the method to use	FIT	METHOD=
Specifies the input data table that contains the correlations matrix for elliptical copulas	FIT	CORR=
Specifies the parameter value for Archimedean copulas	FIT	THETA=
Specifies the degrees of freedom for <i>t</i> copulas	FIT	DF=
Declaring the Role of Variables		
Specifies the names of the variables to use in copula fitting or in simulation	VAR	
Specifies BY-group processing	BY	

Description	Statement	Option
Options to Control the Optimization Process		
Selects the iterative minimization method to use	FIT	OPTMETHOD=
Specifies the maximum number of iterations for the optimization when you are fitting a <i>t</i> or Archimedean copula and METHOD=MLE	FIT	MAXITER=
Sets initial values for parameters	FIT	INIT=
Copula Estimation Options		
Specifies the marginal distribution of the individual variables	FIT	MARGINALS=
Copula Simulation Options		
Specifies the random sample size	SIMULATE	NDRAWS=
Specifies the random number generator seed	SIMULATE	SEED=
Output Control Options		
Specifies the output data table to contain the normalized pseudo-data used for model fitting	FIT	OUTPSEUDO=
Specifies the output item store to preserve the properties of the copula that is fit and the results of fitting the copula	FIT	STORE=
Specifies the output data table to contain the random samples from the simulation with uniform marginal distribution	SIMULATE	OUTUNIFORM=
Prints timing information	PROC CCOPULA	PRINTTIMING
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	

PROC CCOPULA Statement

PROC CCOPULA <options> ;

The following *options* can be used in the PROC CCOPULA statement.

Input Data Table Options

DATA=CAS-libref.data-table

names the input data table for PROC CCOPULA to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the *caslib*, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME

statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

data-table specifies the name of the input data table.

When the procedure is used for simulation only, the input data table is not required in order to run the procedure.

Printing Options

PRINTTIMING

prints a timing report.

BY Statement

BY *variables* ;

The BY statement specifies groups in which separate copula fitting analyses or simulations are performed. When you use the BY statement with the FIT statement, the *variables* must be present in the input data table and are excluded from the model fitting. You must also provide a VAR statement. The BY statement can also be used when simulating a copula. When you use the BY statement with the SIMULATE statement, you must specify an item store in the RESTORE= option. The item store must contain model fitting results for the same BY groups that are defined by the *variables* in your BY statement.

DEFINE Statement

DEFINE *name copula-type* <(parameter-value-options ...)> ;

The DEFINE statement specifies the relevant information about the copula that is used for the simulation. You can specify the following arguments:

name specifies the name of the copula definition. You can use this *name* later in the SIMULATE statement.

copula-type specifies the type of copula. You must specify one of the following copula types, which are described in the section “Details: CCOPULA Procedure” on page 224:

NORMAL	fits the normal copula.
T	fits the <i>t</i> copula.
CLAYTON	fits the Clayton copula.
FRANK	fits the Frank copula.
GUMBEL	fits the Gumbel copula.

parameter-value-options

specify the input parameters that are used to simulate the specified copula. These options must be appropriate for the type of copula specified. You can specify the following *parameter-value-options*:

CORR=CAS-libref.SAS-data-table

specifies the data table that contains the correlations matrix to use for elliptical copulas. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains the correlations between the *n*th variable in the VAR statement and the other variables. You can use this option for normal and *t* copulas.

DF=value

specifies the degrees of freedom. You can use this option for *t* copulas.

KENDALL=CAS-libref.SAS-data-table

specifies the data table that contains the correlations matrix defined in Kendall’s tau. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains the correlations between the *n*th variable in the VAR statement and the other variables. You can use this option for normal and *t* copulas.

SPEARMAN=CAS-libref.SAS-data-table

specifies the data table that contains the correlations matrix defined in Spearman’s rho. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains the correlations between the *n*th variable in the VAR statement and the other variables. You can use this option for normal copulas.

THETA=value

specifies the parameter value for the Archimedean copulas.

The DEFINE statement is used with the SIMULATE statement and the FIT statement.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 240. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 240. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

FIT Statement

FIT *type* < **NAME**=*name* > < **INIT**=(*parameter-value-options*) / *options* ;

The FIT statement estimates the parameters for a specified copula type.

You must specify a *type*:

type

specifies the type of the copula to be estimated, which is one of the following:

CLAYTON	fits the Clayton copula.
FRANK	fits the Frank copula.
GUMBEL	fits the Gumbel copula.
NORMAL	fits the normal copula.
T	fits the <i>t</i> copula.

You can also specify the following options:

INIT=(*parameter-value-options*)

provides the initial values for the numerical optimization.

[*parameter-value-options*]

specify the input parameters that are used to initialize the specified copula. These options must be appropriate for the type of copula that you specify. You can specify the following options:

CORR=*CAS-libref.SAS-data-table*

specifies the data table that contains the correlations matrix to use for elliptical copulas. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA**= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains the correlations between the *n*th variable in the VAR statement and the other variables. You can use this option for *t* copulas.

DF=*value*

specifies the degrees of freedom. You can use this option for *t* copulas.

KENDALL=*CAS-libref.SAS-data-table*

specifies the data table that contains the correlations matrix defined in Kendall’s tau. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA**= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row *n* contains

the correlations between the n th variable in the VAR statement and the other variables. You can use this option for t copulas.

SPEARMAN=CAS-libref.SAS-data-table

specifies the data table that contains the correlations matrix defined in Spearman's rho. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

All columns of the matrix must be numeric. The order of the column names in the matrix must match their order in the VAR statement. The order of the rows must be such that row n contains the correlations between the n th variable in the VAR statement and the other variables. You can use this option for t copulas.

THETA=value

specifies the parameter value for Archimedean copulas.

For Archimedean copulas, the default initial values of the parameter are computed using the calibration method. The default initial value of the degrees-of-freedom parameter in the t copula is set to 2.0. The following statement shows an initialization for Student's t copula, where the Kendall's tau correlations matrix is stored in the mycas.corrmat data table and the DF is set to 2.5:

```
fit t init=(df=2.5 kendall=mycas.corrmat);
```

NAME=name

specifies an identifier for the fit.

You can specify the following *options* after a slash (/):

MARGINALS=EMPIRICAL<(approx-options)> | UNIFORM

specifies the marginal distribution of the individual variables. You can specify the following values:

EMPIRICAL<(approx-options)> uses an approximation of the marginal empirical cumulative distribution function (CDF) to transform the data and uses the transformed data to fit the copula. You can specify the following *approx-options*:

ALG=BIN | SORT

specifies the algorithm to use during the approximation of the marginal distribution. You can specify the following values:

BIN	uses bins to approximate the marginal distributions.
SORT	stores and sorts each variable's observations in the input data table.

By default, ALG=BIN.

INTERP=LINEAR | SPLINE | STEP

specifies how to interpolate values within intervals in the distributions' approximations. You can specify the following values:

LINEAR	specifies that values within an interval be linearly interpolated between the endpoints.
SPLINE	specifies that values within an interval be interpolated using a cubic spline function that is constrained to be monotonically increasing.
STEP	specifies that values within an interval take the value of the left endpoint.

By default, INTERP=LINEAR.

MAXITERS=integer

specifies the maximum number of iterations to perform when you are generating the distribution approximation function. By default, MAXITERS=10. For more information, see the section [“Adaptive Approximation of Marginal Distributions”](#) on page 226.

REFINERES=integer

specifies the maximum number of intervals to add to the distribution function during each iteration of the empirical approximation. By default, REFINERES=1000.

UNIFORM uses the input data without transformation to fit the copula.

METHOD=MLE | CAL

specifies the method to use to estimate parameters. You can specify the following values:

CAL	specifies the calibration method that uses the correlations matrix (only Kendall's tau is implemented in this procedure).
MLE	specifies canonical maximum likelihood estimation (CMLE) or maximum likelihood estimation (MLE).

For the t copula, if METHOD=CAL, then the correlations matrix is estimated using the calibration method with Kendall's tau and the degrees of freedom are estimated by the MLE. For the normal copula, only METHOD=MLE is supported and METHOD=CAL is ignored. By default for all copula types, METHOD=MLE.

OUTPSEUDO=CAS-libref.data-table

specifies the output data table for saving the pseudo-samples with uniform marginal distributions. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section [“Using CAS Sessions and CAS Engine Librefs”](#) on page 209.

The pseudo-samples are obtained by transforming the individual variables of the original data by using the empirical CDFs. The data table is not created if you omit this option.

PLOTS< (*global-plot-options*) > < =(*specific-plot-options*) >

controls the plots of the input data table that are produced by the CCOPULA procedure. By default, PROC CCOPULA produces a scatter plot matrix for variables (that is, it displays a symmetric matrix plot that contains the variables specified in the VAR statement).

You can specify the following *global-plot-options*:

NVAR=ALL | *n*

specifies the maximum number of variables that you specify in the VAR statement to display in the matrix plot. NVAR=ALL uses all variables that you specify in the VAR statement. By default, NVAR=5.

RESOLUTION=*n*

specifies the number of bins to use for the X and Y axes when you create plots. By default, RESOLUTION=50.

SCATTER

creates a scatter plot matrix for variables that you specify in the VAR statement. The maximum number of variables that you can display in the scatter plot matrix is 10.

UNPACKPANEL | UNPACK

creates scatter plots for pairs of variables. If you specify this option, PROC CCOPULA displays a scatter plot for each applicable pair of distinct variables that you specify in the VAR statement.

You can also specify the following *specific-plot-options*:

DATATYPE=ORIGINAL | UNIFORM | BOTH

specifies the data type to be plotted. DATATYPE=ORIGINAL presents the data in their original marginal distribution. DATATYPE=UNIFORM shows the transformed data with a uniform marginal distribution. DATATYPE=BOTH plots both the original and uniform data types. When MARGINALS=UNIFORM, the transformation is omitted and the DATATYPE= option is ignored.

NONE

suppresses all plots.

STORE=SAS-item-store

specifies the item store that preserves the properties of the model and the fit results.

Printing Options

ITPRINT

prints a summary iteration listing.

NOCORR

suppresses the correlations matrix.

NOPRINT

suppresses all output.

PRINTALL

prints all output. This is the default.

SIMULATE Statement

SIMULATE < *copula-name-list* > / *options* ;

The SIMULATE statement simulates data from a specified copula model. The copula name specification is the name of a defined copula as specified by *name* in the DEFINE statement. You can specify the following *options*:

NDRAWS=integer

specifies the number of draws to generate for this simulation. By default, NDRAWS=100.

OUTUNIFORM=CAS-libref.data-table

specifies the output data table to contain the result of the simulation in uniform margins. The data are not created if you do not specify this option. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 209.

PLOTS<(global-plot-options)> < = (specific-plot-options)>

controls the plots of simulated data that are produced by the CCOPULA procedure. By default, PROC CCOPULA produces a scatter plot matrix for variables (that is, it displays a symmetric matrix plot that contains the variables specified in the VAR statement).

You can specify the following *global-plot-options*:

NVAR=ALL | n

specifies the maximum number of variables that you specify in the VAR statement to display in the matrix plot. NVAR=ALL uses all variables that you specify in the VAR statement. By default, NVAR=5.

RESOLUTION=n

specifies the number of bins to use for the X and Y axes when you create plots. By default, RESOLUTION=50.

SCATTER

creates a scatter plot matrix for variables that you specify in the VAR statement. The maximum number of variables that you can display in the scatter plot matrix is 10.

UNPACKPANEL | UNPACK

creates scatter plots for pairs of variables. If you specify this option, PROC CCOPULA displays a scatter plot for each applicable pair of distinct variables that you specify in the VAR statement.

You can also specify the following *specific-plot-options*:

DATATYPE=ORIGINAL | UNIFORM | BOTH

specifies the data type to be plotted. DATATYPE=ORIGINAL presents the data in their original marginal distribution. DATATYPE=UNIFORM shows the transformed data with a uniform marginal distribution. DATATYPE=BOTH plots both the original and uniform data types. When MARGINALS=UNIFORM, the transformation is omitted and the DATATYPE= option is ignored.

NONE

suppresses all plots.

RESTORE=SAS-item-store

specifies the input item store that contains the results of fitting a copula. The final estimates, correlations, and other properties of the model that are preserved in the RESTORE= item store are used to define the simulation. The DEFINE and VAR statements are ignored if this option is specified.

SEED=integer

specifies the seed for generating random numbers for the simulation. If you do not specify the seed, then the seed is generated by reading the time of day from the computer's clock.

TOLERANCE=value

specifies the tolerance allowed for the simulation.

VAR Statement

VAR *variables* ;

The VAR statement specifies the variable names in the input data table that is specified by the DATA= option in the PROC CCOPULA statement. The subset of variables in the data table is used for the copula models in the FIT statement. If there is no input data table, the VAR statement creates the list of variable names for the SIMULATE statement.

VIEWSTORE Statement

VIEWSTORE / *options* ;

The VIEWSTORE statement enables you to print various reports that describe the model fitting information that is preserved in an item store. The item store must have been created using the STORE= option in the FIT statement. You can specify the following *options*:

ALL

prints the entire contents of the item store.

BYVARVALUES

prints the values of the BY-group variables that are used to fit the model.

CORRELATIONS

prints the correlations matrices.

FINALESTIMATES

prints the final parameter estimates.

FITMODELSUMMARY

prints the fit model summary.

INITIALESTIMATES

prints the initial parameter estimates.

INSTORE=*SAS-item-store*

specifies the item store that you want to view. If you omit this option, and if you use a FIT statement to specify a STORE= item store, then the STORE= item store that is specified in the FIT statement is used as the INSTORE= item store.

MINIMAL

prints only the fit model summary and the solution.

MODELDEFINITION

prints the statements that define the model.

NONE

suppresses all reports.

OPTIMIZERSETTINGS

prints the optimizer settings that are used to fit the model.

Details: CCOPULA Procedure

Sklar's Theorem

The copula models are tools for studying the dependence structure of multivariate distributions. The usual joint distribution function contains the information both about the marginal behavior of the individual random variables and about the dependence structure between the variables. The copula is introduced to decouple the marginal properties of the random variables and the dependence structures. An m -dimensional *copula* is a joint distribution function on $[0, 1]^m$, where all marginal distributions are standard uniform. The common notation for a copula is $C(u_1, \dots, u_m)$.

The Sklar (1959) theorem shows the importance of copulas in modeling multivariate distributions. The first part of the theorem states that a copula can be derived from any joint distribution functions, and the second part asserts the opposite: that any copula can be combined with any set of marginal distributions to result in a multivariate distribution function. The theorem follows:

- Let F be a joint distribution function, and let $F_j, j = 1, \dots, m$, be the marginal distributions. Then there exists a copula $C : [0, 1]^m \rightarrow [0, 1]$ such that

$$F(x_1, \dots, x_m) = C(F_1(x_1), \dots, F_m(x_m))$$

for all $x_1, \dots, x_m$ in $[-\infty, \infty]$. Moreover, if the margins are continuous, then C is unique; otherwise C is uniquely determined on $\text{Ran} F_1 \times \dots \times \text{Ran} F_m$, where $\text{Ran} F_j = F_j([-\infty, \infty])$ is the range of F_j .

- The converse is also true. That is, if C is a copula and $F_1, \dots, F_m$ are univariate distribution functions, then the multivariate function that is defined in the preceding equation is a joint distribution function with marginal distributions $F_j, j = 1, \dots, m$.

Dependence Measures

There are three basic types of dependence measures: linear correlation, rank correlation, and tail dependence. Linear correlation is given by

$$\rho \equiv \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sqrt{\text{var}(X)}\sqrt{\text{var}(Y)}}$$

The linear correlation coefficient contains very limited information about the joint properties of the variables. A well-known property is that zero correlation does not imply independence, whereas independence implies zero correlation. In addition, there are distinct bivariate distributions that have the same marginal distribution and the same correlation coefficient. These results suggest that caution must be used in interpreting the linear correlation.

Another statistical measure of dependence is rank correlation, which is nonparametric. For example, Kendall's tau is the covariance between the sign statistics $X_1 - \tilde{X}_1$ and $X_2 - \tilde{X}_2$, where $(\tilde{X}_1, \tilde{X}_2)$ is an independent copy of (X_1, X_2) :

$$\rho_\tau \equiv E[\text{sign}(X_1 - \tilde{X}_1)(X_2 - \tilde{X}_2)]$$

The sign function (sometimes written as sgn) is defined as

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x \leq 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Spearman's rho is the correlation between the transformed random variables:

$$\rho_S(X_1, X_2) \equiv \rho(F_1(X_1), F_2(X_2))$$

The variables are transformed by their distribution functions so that the transformed variables are uniformly distributed on $[0, 1]$. The rank correlations depend only on the copula of the random variables and are indifferent to the marginal distributions. Like linear correlation, rank correlation has its limitations. In particular, different copulas result in the same rank correlation.

A third measure, tail dependence, focuses on only part of the joint properties between the variables. Tail dependence measures the dependence when both variables have extreme values. Formally, they can be defined as the conditional probabilities of quantile exceedances. There are two types of tail dependence:

- Upper tail dependence is defined as

$$\lambda_u(X_1, X_2) \equiv \lim_{q \rightarrow 1^-} P(X_2 > F_2^{-1}(q) | X_1 > F_1^{-1}(q))$$

when the limit exists and $\lambda_u \in [0, 1]$. Here F_j^{-1} is the quantile function (that is, the inverse of the CDF).

- Lower tail dependence is defined symmetrically.

Data Processing and Fitting

Before you can fit a copula, you must transform the data into pseudo-uniform data. If you specify the MARGINALS=UNIFORM option, then it is assumed that the data are already in the proper form, so the required transformation is the trivial identity transformation. However, if you specify MARGINALS=EMPIRICAL (or if you omit the MARGINALS= option), then the transformation is nontrivial.

For relatively small data tables, each observation in the original input data table can be transformed into a pseudo-uniform observation by using a simple value-comparison algorithm. In this case, the initial estimates of the parameters in t copulas and Archimedean copulas can be derived by computing a Kendall correlations matrix on the basis of the transformed data.

However, for larger data tables, the simple value-comparison algorithm becomes computationally impractical as a means of transforming the original data into pseudo-uniform data. Thus, a data binning algorithm is used to transform the original data into pseudo-uniform data. After the data have been transformed in this way, it can still be computationally impractical to compute a Kendall correlations matrix directly from the data, because the algorithm for deriving the Kendall correlations matrix requires a pairwise comparison of every pair of observations. Therefore, after the data have been transformed via binning, a Pearson correlations matrix is computed on the basis of the transformed data, and then a Kendall correlations matrix is derived from the Pearson correlations matrix by applying an analytical formula. Then the initial estimates for the parameters in the t copulas and Archimedean copulas are derived from the Kendall correlations matrix.

Adaptive Approximation of Marginal Distributions

Simple binning into fixed-width intervals of each variable's empirical data can lead to a poor approximation of distributions for variables that have highly nonsmooth distributions. For smaller data tables, fixed-interval bins can be an inefficient representation of the empirical distributions because many bins are empty.

To accommodate variables that have a wide range of empirical distributions, PROC CCOPULA uses an adaptive binning algorithm to generate an approximation of each variable's cumulative distribution function (CDF). The following steps summarize how an approximate CDF is computed for each variable:

1. A partitioning of the equal-width intervals specified in the REFINERES= option of the FIT statement between the minimum and maximum values is created so that each value falls into one interval.
2. The number of values that fall within each interval is counted.
3. Each interval is assigned a weight on the basis of its count.
4. Each interval is subdivided into a number of subintervals proportional to its weight so that the total number of new subdivisions across all intervals equals the value specified in the REFINERES= option of the FIT statement.
5. The process of subdividing intervals is repeated starting at step 2 until either the weight of each interval falls below a threshold value or the algorithm has run the number of times specified in the MAXITERS= option of the FIT statement.

After a final partitioning with counts has been created, the count that is associated with each interval is modified to include the cumulative counts for all smaller intervals.

The CDF for a particular value is computed by interpolating the counts between the endpoints of its interval, by using the method that you specify in the INTERP= option.

Two alternative approaches are available for counting the number of observations in step 2 of the approximation algorithm. When you specify ALG=BIN, the interval counts are computed by locating the interval associated with each observation and incrementing that interval's count. When you specify ALG=SORT, all the observations are stored in memory and sorted to make the assignment of observations to intervals faster in step 2.

The adaptive algorithm executes its steps in parallel across all available threads and machines except for step 5, in which the partitioning available on each thread must be merged and shared with all other threads.

Normal Copula

Let $u_j \sim U(0, 1)$ for $j = 1, \dots, m$, where $U(0, 1)$ represents the uniform distribution on the $[0, 1]$ interval. Let Σ be the correlations matrix, where $m(m-1)/2$ parameters satisfy the positive semidefiniteness constraint. The normal copula can be written as

$$C_{\Sigma}(u_1, u_2, \dots, u_m) = \Phi_{\Sigma}(\Phi^{-1}(u_1), \dots, \Phi^{-1}(u_m))$$

where Φ is the distribution function of a standard normal random variable and Φ_{Σ} is the m -variate standard normal distribution with mean vector 0 and covariance matrix Σ . That is, the distribution Φ_{Σ} is $N_m(0, \Sigma)$.

Simulation

For the normal copula, the input of the simulation is the correlations matrix Σ . The normal copula can be simulated by the following steps, in which $\mathbf{U} = (U_1, \dots, U_m)$ denotes one random draw from the copula:

1. Generate a multivariate normal vector $\mathbf{Z} \sim N(0, \Sigma)$, where Σ is an m -dimensional correlations matrix.
2. Transform the vector $\mathbf{Z}$ into $\mathbf{U} = (\Phi(Z_1), \dots, \Phi(Z_m))^T$, where Φ is the distribution function of univariate standard normal.

The first step can be achieved by Cholesky decomposition of the correlations matrix $\Sigma = LL^T$, where L is a lower triangular matrix with positive elements on the diagonal. If $\tilde{\mathbf{Z}} \sim N(0, I)$, then $L\tilde{\mathbf{Z}} \sim N(0, \Sigma)$.

Fitting

To fit a normal copula is to estimate the covariance matrix Σ from an input sample data table. Given a random sample $\mathbf{u}_i = (u_{i,1}, \dots, u_{i,m})^T$ where $i = 1, \dots, n$, the log-likelihood function is

$$\begin{aligned} \log L(\Sigma; \mathbf{u}_1, \dots, \mathbf{u}_n) \\ = \sum_{t=1}^n \log f_{\Sigma}(\Phi^{-1}(u_{t,1}), \dots, \Phi^{-1}(u_{t,m})) - \sum_{t=1}^n \sum_{j=1}^m \log \phi(\Phi^{-1}(u_{t,j})) \end{aligned}$$

Here f_{Σ} is the joint density of the multivariate normal with mean zero and variance Σ , and ϕ is the univariate density of the standard normal distribution. Note that the second term is not related to the parameters Σ and, therefore, can be ignored during the optimization. The restriction that Σ is a correlations matrix is very inconvenient, and it is common practice to circumvent this problem by first assuming that Σ has the covariance form. Therefore, Σ can be estimated by

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n \xi_i \xi_i^T$$

where

$$\xi_i = (\Phi^{-1}(u_{i,1}), \Phi^{-1}(u_{i,2}), \dots, \Phi^{-1}(u_{i,m}))^T$$

This estimate is consistent with the form of a covariance matrix but not necessarily with the form of a correlations matrix. The approximation to the original MLE problem can be obtained using the normalizing operator defined as follows:

$$\begin{aligned} \Delta(\Sigma) &= \text{diag}(\sigma_{11}^{1/2}, \dots, \sigma_{mm}^{1/2}) \\ \mathcal{P}(\Sigma) &= (\Delta(\Sigma))^{-1} \Sigma (\Delta(\Sigma))^{-1} \end{aligned}$$

Student's t copula

Let $\Theta = \{(\nu, \Sigma) : \nu \in (1, \infty), \Sigma \in \mathbb{R}^{m \times m}\}$, and let t_{ν} be a univariate t distribution with ν degrees of freedom.

The Student's t copula can be written as

$$C_{\Theta}(u_1, u_2, \dots, u_m) = \mathbf{t}_{\nu, \Sigma} \left(t_{\nu}^{-1}(u_1), t_{\nu}^{-1}(u_2), \dots, t_{\nu}^{-1}(u_m) \right)$$

where $\mathbf{t}_{\nu, \Sigma}$ is the multivariate Student's t distribution that has a correlations matrix Σ with ν degrees of freedom.

Simulation

The input parameters for the simulation are (ν, Σ) . The t copula can be simulated by the following steps:

1. Generate a multivariate vector $\mathbf{X} \sim t_m(\nu, 0, \Sigma)$ that follows the centered t distribution with ν degrees of freedom and correlations matrix Σ .
2. Transform the vector $\mathbf{X}$ into $\mathbf{U} = (t_{\nu}(X_1), \dots, t_{\nu}(X_m))^T$, where t_{ν} is the distribution function of univariate t distribution with ν degrees of freedom.

To simulate centered multivariate t random variables, you can use the property that $\mathbf{X} \sim t_m(\nu, 0, \Sigma)$ if $\mathbf{X} = \sqrt{\nu/s} \mathbf{Z}$, where $\mathbf{Z} \sim N(0, \Sigma)$ and the univariate random variable $s \sim \chi_{\nu}^2$.

Fitting

To fit a *t* copula is to estimate the covariance matrix Σ and degrees of freedom ν from given multivariate data. Given a random sample $\mathbf{u}_i = (u_{i,1}, \dots, u_{i,m})^\top$, $i = 1, \dots, n$ that has uniform marginal distributions, the log likelihood is

$$\log L(\nu, \Sigma; u_{i,1}, \dots, u_{i,m}) \\ = \sum_{i=1}^n \log g_{\nu, \Sigma}(t_{\nu}^{-1}(u_{i,1}), \dots, t_{\nu}^{-1}(u_{i,m})) - \sum_{i=1}^n \sum_{j=1}^m \log g_{\nu}(t_{\nu}^{-1}(u_{i,j}))$$

where ν denotes the degrees of freedom of the *t* copula, $g_{\nu, \Sigma}$ denotes the joint density function of the centered multivariate *t* distribution with parameters (ν, Σ) , t_{ν} is the distribution function of a univariate *t* distribution with ν degrees of freedom, Σ is a correlations matrix, and g_{ν} is the density function of univariate *t* distribution with ν degrees of freedom.

The log likelihood can be maximized with respect to the parameters $\theta = (\nu, \Sigma) \in \Theta$ using numerical optimization. If you allow the parameters in Σ to be such that Σ is symmetric and with ones on the diagonal, then the MLE estimate for Σ might not be positive semidefinite. In that case, you need to apply the adjustment to convert the estimated matrix to positive semidefinite, as shown by McNeil, Frey, and Embrechts (2005), Algorithm 5.55.

When the dimension of the data m increases, the numerical optimization quickly becomes infeasible. It is common practice to estimate the correlations matrix Σ by calibration using Kendall's tau. Then, using this fixed Σ , the single parameter ν can be estimated by MLE. By proposition 5.37 in McNeil, Frey, and Embrechts (2005),

$$\rho_{\tau}(U_i, U_j) = \frac{2}{\pi} \arcsin \rho_{ij}$$

where ρ_{τ} is the Kendall's tau and ρ_{ij} is the off-diagonal elements of the correlations matrix Σ of the *t* copula. Therefore, an estimate for the correlation is

$$\hat{\rho}_{ij} = \sin \left(\frac{1}{2} \pi \hat{\rho}_{i,j}^{\tau} \right)$$

where $\hat{\rho}$ and $\hat{\rho}^{\tau}$ are the estimates of the sample correlations matrix and Kendall's tau, respectively. However, it is possible that the estimate of the correlations matrix $\hat{\Sigma}$ is not positive definite. In this case, there is a standard procedure that uses the eigenvalue decomposition to transform the correlations matrix into one that is positive definite. Let Σ be a symmetric matrix with ones on the diagonal, with off-diagonal entries in $[-1, 1]$. If Σ is not positive semidefinite, use Algorithm 5.55 from McNeil, Frey, and Embrechts (2005):

1. Compute the eigenvalue decomposition $\Sigma = EDE^T$, where D is a diagonal matrix that contains all the eigenvalues and E is an orthogonal matrix that contains the eigenvectors.
2. Construct a diagonal matrix $\tilde{D}$ by replacing all negative eigenvalues in D by a small value $\delta > 0$.
3. Compute $\tilde{\Sigma} = E\tilde{D}E^T$, which is positive definite but not necessarily a correlations matrix.
4. Apply the normalizing operator $\mathcal{P}$ on the matrix $\tilde{\Sigma}$ to obtain the correlations matrix desired.

The log-likelihood function and its gradient function for a single observation are listed as follows, where $\xi = (\xi_1, \dots, \xi_m)$, with $\xi_j = t_v^{-1}(u_j)$, and g is the derivative of the log Γ function:

$$\begin{aligned}
 l = \log(c) &= -\frac{1}{2} \log(|\Sigma|) + \log \Gamma\left(\frac{\nu+m}{2}\right) + (m-1) \log \Gamma\left(\frac{\nu}{2}\right) - m \log \Gamma\left(\frac{\nu+1}{2}\right) \\
 &\quad - \frac{\nu+m}{2} \log(1 + \xi^T \Sigma^{-1} \xi / \nu) + \frac{\nu+1}{2} \sum_{j=1}^m \log\left(1 + \frac{\xi_j^2}{\nu}\right) \\
 \frac{\partial l}{\partial \nu} &= \frac{1}{2} g\left(\frac{\nu+m}{2}\right) + \frac{m-1}{2} g\left(\frac{\nu}{2}\right) - \frac{m}{2} g\left(\frac{\nu+1}{2}\right) \\
 &\quad - \frac{1}{2} \log(1 + \xi^T \Sigma^{-1} \xi / \nu) + \frac{\nu+m}{2\nu^2} \frac{\xi^T \Sigma^{-1} \xi}{1 + \xi^T \Sigma^{-1} \xi / \nu} \\
 &\quad + \frac{1}{2} \sum_{j=1}^m \log(1 + \xi_j^2 / \nu) - \frac{\nu+1}{2\nu^2} \sum_{j=1}^m \frac{\xi_j^2}{1 + \xi_j^2 / \nu} \\
 &\quad - \frac{(\nu+m)}{\nu} \frac{\xi^T \Sigma^{-1} (d\xi/d\nu)}{1 + \xi^T \Sigma^{-1} \xi / \nu} + \frac{\nu+1}{\nu} \sum_{j=1}^m \frac{\xi_j (d\xi_j/d\nu)}{1 + \xi_j^2 / \nu}
 \end{aligned}$$

The derivative of the likelihood with respect to the correlations matrix Σ follows:

$$\begin{aligned}
 \frac{\partial l}{\partial \Sigma} &= -\frac{1}{2} (\Sigma^{-1})^T + \frac{\nu+m}{2} \frac{\Sigma^{-T} \xi \xi^T \Sigma^{-T} / \nu}{1 + \xi^T \Sigma^{-1} \xi / \nu} \\
 &= -\frac{1}{2} (\Sigma^{-1})^T + \frac{\nu+m}{2} \frac{\Sigma^{-T} \xi \xi^T \Sigma^{-T}}{\nu + \xi^T \Sigma^{-1} \xi}
 \end{aligned}$$

Archimedean Copulas

Overview of Archimedean Copulas

Let function $\phi : [0, 1] \rightarrow [0, \infty)$ be a strict Archimedean copula generator function, and suppose that its inverse ϕ^{-1} is completely monotonic on $[0, \infty)$. A strict generator is a decreasing function $\phi : [0, 1] \rightarrow [0, \infty)$ that satisfies $\phi(0) = \infty$ and $\phi(1) = 0$. A decreasing function $f(t) : [a, b] \rightarrow (-\infty, \infty)$ is completely monotonic if it satisfies

$$(-1)^k \frac{d^k}{dt^k} f(t) \geq 0, k \in \mathbb{N}, t \in (a, b)$$

An Archimedean copula is defined as follows:

$$C(u_1, u_2, \dots, u_m) = \phi^{-1}\left(\phi(u_1) + \dots + \phi(u_m)\right)$$

The Archimedean copulas available in the CCOPULA procedure are the Clayton copula, the Frank copula, and the Gumbel copula.

Clayton Copula

Let the generator function $\phi(u) = \theta^{-1} (u^{-\theta} - 1)$. A Clayton copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \left[\sum_{i=1}^m u_i^{-\theta} - m + 1 \right]^{-1/\theta}$$

where $\theta > 0$.

Frank Copula

Let the generator function be

$$\phi(u) = -\log \left[\frac{\exp(-\theta u) - 1}{\exp(-\theta) - 1} \right]$$

A Frank copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \frac{1}{\theta} \log \left\{ 1 + \frac{\prod_{i=1}^m [\exp(-\theta u_i) - 1]}{[\exp(-\theta) - 1]^{m-1}} \right\}$$

where $\theta \in (-\infty, \infty) \setminus \{0\}$ for $m = 2$ and $\theta > 0$ for $m \geq 3$.

Gumbel Copula

Let the generator function $\phi(u) = (-\log u)^\theta$. A Gumbel copula is defined as

$$C_\theta(u_1, u_2, \dots, u_m) = \exp \left\{ - \left[\sum_{i=1}^m (-\log u_i)^\theta \right]^{1/\theta} \right\}$$

where $\theta > 1$.

Simulation

Suppose that the generator of the Archimedean copula is ϕ . Then the simulation method that uses a Laplace-Stieltjes transformation of the distribution function is given by Marshall and Olkin (1988), where $\tilde{F}(t) = \int_0^\infty e^{-tx} dF(x)$:

1. Generate a random variable V that has the distribution function F such that $\tilde{F}(t) = \phi^{-1}(t)$.
2. Draw samples from the independent uniform random variables $X_1, \dots, X_m$.
3. Return $U = (\tilde{F}(-\log(X_1)/V), \dots, \tilde{F}(-\log(X_m)/V))^T$.

The Laplace-Stieltjes transformations are as follows:

- For the Clayton copula, $\tilde{F} = (1 + t)^{-1/\theta}$, and the distribution function F is associated with a gamma random variable that has a shape parameter of θ^{-1} and a scale parameter of 1.

- For the Gumbel copula, $\tilde{F} = \exp(-t^{1/\theta})$, and F is the distribution function of the stable variable $\text{St}(\theta^{-1}, 1, \gamma, 0)$, where $\gamma = [\cos(\pi/(2\theta))]^\theta$.
- For the Frank copula where $\theta > 0$, $\tilde{F} = -\log\{1 - \exp(-t)[1 - \exp(-\theta)]\}/\theta$, and F is a discrete probability function $P(V = k) = (1 - \exp(-\theta))^k/(k\theta)$. This probability function is related to a logarithmic random variable that has a parameter value of $1 - e^{-\theta}$.

For more information about simulating a random variable from a stable distribution, see Theorem 1.19 in Nolan (2010). For more information about simulating a random variable from a logarithmic series, see Chapter 10.5 in Devroye (1986).

For a Frank copula where $m = 2$ and $\theta < 0$, the simulation can be done through conditional distributions as follows:

1 Draw independent v_1, v_2 from a uniform distribution.

2 Let $u_1 = v_1$.

3 Let $u_2 = -\frac{1}{\theta} \log \left(1 + \frac{v_2(1-e^{-\theta})}{v_2(e^{-\theta v_1}-1)-e^{-\theta v_1}} \right)$.

Fitting

One method to estimate the parameters is to calibrate with Kendall's tau. The relation between the parameter θ and Kendall's tau is summarized in Table 6.2 for the three Archimedean copulas.

Table 6.2 Calibration Using Kendall's Tau

Copula Type	τ	Formula for θ
Clayton	$\theta/(\theta + 2)$	$2\tau/(1 - \tau)$
Gumbel	$1 - 1/\theta$	$1/(1 - \tau)$
Frank	$1 - 4\theta^{-1}(1 - D_1(\theta))$	No closed form

In Table 6.2, $D_1(\theta) = \theta^{-1} \int_0^\theta t/(\exp(t) - 1)dt$ for $\theta > 0$, and $D_1(\theta) = D_1(\theta) + 0.5\theta$ for $\theta < 0$. In addition, for the Frank copula, the formula for θ has no closed form. The numerical algorithm for root finding can be used to invert the function $\tau(\theta)$ to obtain θ as a function of τ .

Alternatively, you can use the MLE or CMLE method to estimate the parameter θ given the data $\mathbf{u} = \{u_{i,j}\}$ and $i = 1, \dots, n, j = 1, \dots, m$. The log-likelihood function for each type of Archimedean copula is provided in the following sections.

Fitting the Clayton Copula

For the Clayton copula, the log-likelihood function is as follows (Cherubini, Luciano, and Vecchiato 2004, Chapter 7):

$$l = n \left[m \log(\theta) + \log \left(\Gamma \left(\frac{1}{\theta} + m \right) \right) - \log \left(\Gamma \left(\frac{1}{\theta} \right) \right) \right] - (\theta + 1) \sum_{i,j} \log u_{ij} \\ - \left(\frac{1}{\theta} + m \right) \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right)$$

Let $g(\cdot)$ be the derivative of $\log(\Gamma(\cdot))$. Then the first-order derivative is

$$\frac{dl}{d\theta} = n \left[\frac{m}{\theta} + g \left(\frac{1}{\theta} + m \right) \frac{-1}{\theta^2} - g \left(\frac{1}{\theta} \right) \frac{-1}{\theta^2} \right] \\ - \sum_{i,j} \log(u_{ij}) + \frac{1}{\theta^2} \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right) \\ - \left(\frac{1}{\theta} + m \right) \sum_i \frac{-\sum_j u_{ij}^{-\theta} \log(u_{ij})}{\sum_j u_{ij}^{-\theta} - m + 1}$$

The second-order derivative is

$$\frac{d^2l}{d\theta^2} = n \left\{ \frac{-m}{\theta^2} + g' \left(\frac{1}{\theta} + m \right) \frac{1}{\theta^4} + g \left(\frac{1}{\theta} + m \right) \frac{2}{\theta^3} - g' \left(\frac{1}{\theta} \right) \frac{1}{\theta^4} - g \left(\frac{1}{\theta} \right) \frac{2}{\theta^3} \right\} \\ - \frac{2}{\theta^3} \sum_i \log \left(\sum_j u_{ij}^{-\theta} - m + 1 \right) \\ + \frac{2}{\theta^2} \sum_i \frac{-\sum_j u_{ij}^{-\theta} \log u_{ij}}{\sum_j u_{ij}^{-\theta} - m + 1} \\ - \left(\frac{1}{\theta} + m \right) \sum_i \left\{ \frac{\sum_j u_{ij}^{-\theta} (\log u_{ij})^2}{\sum_j u_{ij}^{-\theta} - m + 1} - \left(\frac{\sum_j u_{ij}^{-\theta} \log u_{ij}}{\sum_j u_{ij}^{-\theta} - m + 1} \right)^2 \right\}$$

Fitting the Gumbel Copula

A different parameterization $\alpha = \theta^{-1}$ is used for the following part, which is related to the fitting of the Gumbel copula. For the Gumbel copula, you need to compute $\phi^{-1(m)}$. It turns out that for $k = 1, 2, \dots, m$,

$$\phi^{-1(k)}(u) = (-1)^k \alpha \exp(-u^\alpha) u^{-k+\alpha} \Psi_{k-1}(u^\alpha)$$

where Ψ_{k-1} is a function that is described later. The copula density is given by

$$c = \phi^{-1(m)}(x) \prod_k \phi'(u_k) \\ = (-1)^m \alpha \exp(-x^\alpha) x^{-k+\alpha} \Psi_{m-1}(x^\alpha) \prod_k \phi'(u_k) \\ = (-1)^m f_1 f_2 f_3 f_4 f_5$$

where $x = \sum_k \phi(u_k)$, $f_1 = \alpha$, $f_2 = \exp(-x^\alpha)$, $f_3 = x^{-k+\alpha}$, $f_4 = \Psi_{m-1}(x^\alpha)$, and $f_5 = (-1)^m \prod_k \phi'(u_k)$.

The log density is

$$\begin{aligned} l &= \log(c) \\ &= \log(f_1) + \log(f_2) + \log(f_3) + \log(f_4) + \log((-1)^m f_5) \end{aligned}$$

Now the first-order derivative of the log density has the decomposition

$$\frac{dl}{d\alpha} = \frac{1}{c} \frac{dc}{d\alpha} = \sum_{j=1}^4 \frac{1}{f_j} \frac{df_j}{d\alpha} + \frac{d \sum_k \log(-\phi'(u_k))}{d\alpha}$$

Some of the terms are given by

$$\begin{aligned} \frac{1}{f_1} \frac{df_1}{d\alpha} &= \frac{1}{\alpha} \\ \frac{1}{f_2} \frac{df_2}{d\alpha} &= -x^\alpha \log(x) - \alpha x^{\alpha-1} \frac{dx}{d\alpha} \\ \frac{1}{f_3} \frac{df_3}{d\alpha} &= \log(x) + (-k + \alpha) x^{-1} \frac{dx}{d\alpha} \end{aligned}$$

where

$$\frac{dx}{d\alpha} = \sum (-\log u_k)^{1/\alpha} \log(-\log u_k) \left(\frac{-1}{\alpha^2} \right)$$

The last term in the derivative of the $dl/d\alpha$ is

$$\begin{aligned} \log(-\phi'(u_k)) &= \log\left(\frac{1}{\alpha} (-\log u_k)^{\frac{1}{\alpha}-1} \frac{1}{u_k}\right) \\ &= -\log \alpha - \log(u_k) + \left(\frac{1}{\alpha} - 1\right) \log(-\log(u_k)) \\ \frac{d \sum_k \log(-\phi'(u_k))}{d\alpha} &= \sum_{k=1}^m -\frac{1}{\alpha} - \frac{1}{\alpha^2} \log(-\log(u_k)) \\ &= -\frac{m}{\alpha} - \frac{1}{\alpha^2} \sum_{k=1}^m \log(-\log(u_k)) \end{aligned}$$

Now the only remaining term is f_4 , which is related to Ψ_{m-1} . Wu, Valdez, and Sherris (2007) show that $\Psi_k(x)$ satisfies a recursive equation

$$\Psi_k(x) = [\alpha(x-1) + k] \Psi_{k-1}(x) - \alpha x \Psi'_{k-1}(x)$$

with $\Psi_0(x) = 1$.

The preceding equation implies that $\Psi_{k-1}(x)$ is a polynomial of x and therefore can be represented as

$$\Psi_{k-1}(x) = \sum_{j=0}^{k-1} a_j(k-1, \alpha) x^j$$

In addition, its coefficient, denoted by $a_j(k-1, \alpha)$, is a polynomial of α . For simplicity, use the notation $a_j(\alpha) \equiv a_j(m-1, \alpha)$. Therefore,

$$f_4 = \Psi_{m-1}(x^\alpha) = \sum_{j=0}^{m-1} a_j(\alpha) x^{j\alpha}$$

$$\begin{aligned} \frac{df_4}{d\alpha} &= \frac{d\Psi_{m-1}(x^\alpha)}{d\alpha} \\ &= \sum_{j=0}^{m-1} \left[\frac{da_j(\alpha)}{d\alpha} x^{j\alpha} + a_j(\alpha) x^{j\alpha} \log(x) j + a_j(\alpha) (j\alpha) x^{j\alpha-1} \frac{dx}{d\alpha} \right] \end{aligned}$$

Fitting the Frank Copula

For the Frank copula,

$$\phi^{-1(k)}(u) = -\frac{1}{\theta} \Psi_{k-1} \left((1 + e^{-u}(e^{-\theta} - 1))^{-1} \right)$$

When $\theta > 0$, a Frank copula has a probability density function

$$\begin{aligned} c &= \varphi^{-1(m)}(x) \prod_k \varphi'(u_k) \\ &= \frac{-1}{\theta} \Psi_{m-1} \left(\frac{1}{1 + e^{-x}(e^{-\theta} - 1)} \right) \prod_k \varphi'(u_k) \end{aligned}$$

where $x = \sum_k \varphi(u_k)$.

The log likelihood is

$$\log c = -\log(\theta) + \log \left(\Psi_{m-1} \left(\frac{1}{1 + e^{-x}(e^{-\theta} - 1)} \right) \right) + \sum \log(\varphi'(u_k))$$

Denote

$$y = \frac{1}{1 + e^{-x}(e^{-\theta} - 1)}$$

Then the derivative of the log likelihood is

$$\frac{d \log c}{d\theta} = -\frac{1}{\theta} + \frac{1}{\Psi_{m-1}(y)} \frac{d\Psi_{m-1}}{d\theta} + \sum_k \frac{1}{\varphi'(u_k)} \frac{d\varphi'(u_k)}{d\theta}$$

The term in the last summation is

$$\frac{1}{\varphi'(u_k)} \frac{d\varphi'(u_k)}{d\theta} = \frac{1}{\theta(1 - e^{\theta u_k})} [1 - e^{\theta u_k} + \theta u_k e^{\theta u_k}]$$

The function Ψ_{m-1} satisfies a recursive relation

$$\Psi_k(x) = x(x-1)\Psi'_{k-1}(x)$$

with $\Psi_0(x) = x - 1$. Note that Ψ_{m-1} is a polynomial whose coefficients do not depend on θ ; therefore,

$$\begin{aligned} \frac{d\Psi_{m-1}}{d\theta} &= \frac{d\Psi_{m-1}}{dy} \frac{dy}{d\theta} \\ &= \frac{d\Psi_{m-1}}{dy} \left[\frac{dy}{d\theta} + \frac{dy}{dx} \frac{dx}{d\theta} \right] \\ &= \frac{d\Psi_{m-1}}{dy} \left[\frac{e^{-x}e^{-\theta}}{[1 + e^{-x}(e^{-\theta} - 1)]^2} + \frac{e^{-x}(e^{-\theta} - 1)}{[1 + e^{-x}(e^{-\theta} - 1)]^2} \frac{dx}{d\theta} \right] \end{aligned}$$

where

$$\begin{aligned} \frac{dx}{d\theta} &= \sum_k \frac{d\varphi(u_k)}{d\theta} = \sum_k \left[-\frac{u_k e^{-\theta u_k}}{1 - e^{-\theta u_k}} + \frac{e^{-\theta}}{1 - e^{-\theta}} \right] \\ &= \sum_k \left[-\frac{u_k}{e^{\theta u_k} - 1} + \frac{1}{e^{\theta} - 1} \right] \end{aligned}$$

For the case of $m = 2$ and $\theta < 0$, the bivariate density is

$$\log c = \log(\theta(1 - e^{-\theta})) - \theta(u_1 + u_2) - \log((1 - e^{-\theta} - (1 - e^{-\theta u_1})(1 - e^{-\theta u_2}))^2)$$

Canonical Maximum Likelihood Estimation (CMLE)

In the canonical maximum likelihood estimation (CMLE) method, it is assumed that the sample data $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{im})^\top$, where $i = 1, \dots, n$, have been transformed into uniform variates $\hat{\mathbf{u}}_i = (\hat{u}_{i1}, \dots, \hat{u}_{im})$, $i = 1, \dots, n$. One commonly used transformation is the nonparametric estimation of the CDF of the marginal distributions, which is closely related to empirical CDF,

$$\hat{u}_{i,j} = \hat{F}_{j,n}(x_{i,j})$$

where

$$\hat{F}_{j,n}(x) = \frac{1}{n+1} \sum_{i=1}^n I_{[x_{i,j} \leq x]}$$

The transformed data $\hat{u}_{i,j}$ are used as if they had uniform marginal distributions; hence, they are called pseudo-samples. The function $\hat{F}_{j,n}$ is different from the standard empirical CDF in the scalar $1/(n+1)$,

which is to ensure that the transformed data cannot be on the boundary of the unit interval $[0, 1]$. It is clear that

$$\hat{u}_{i,j} = \frac{1}{n+1} \text{rank}(x_{i,j})$$

where $\text{rank}(x_{i,j})$ is the rank among $i = 1, \dots, n$ in increasing order.

Let $c(u_1, u_2, \dots, u_m; \theta)$ be the density function of a copula $C(u_1, u_2, \dots, u_m; \theta)$, and let θ be the parameter vector to be estimated. The parameter θ is estimated by maximum likelihood:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log c(\hat{u}_{i1}, \dots, \hat{u}_{im}; \theta)$$

Exact Maximum Likelihood Estimation (MLE)

Suppose that the marginal distributions of vector elements $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{im})^\top$, where $i = 1, \dots, n$, are already known to be uniform. Then the parameter θ is estimated by exact maximum likelihood:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log c(x_{i1}, x_{i2}, \dots, x_{im}; \theta)$$

Calibration Estimation

Instead of fitting the whole distribution as in MLE methods, you can directly use empirical estimates of distribution parameters. The unknown parameter that you want to estimate can be obtained by calibration that uses Kendall's tau. There is a one-to-one map between the parameter of interest and Kendall's tau. Therefore, after you estimate Kendall's tau, you can use the map to compute the parameter value. For example, the parameter matrix Σ in a t copula and the parameter θ in Archimedean copulas can be estimated in this manner. The most frequently used estimator of Kendall's tau is the rank correlation coefficient:

$$\hat{\rho}_\tau(X_i, X_j) = \left(\frac{n}{2} \right)^{-1} \sum_{1 \leq t < s \leq n} \text{sign}((x_{t,i} - x_{s,i})(x_{t,j} - x_{s,j}))$$

The preceding formula is analogous to its population counterpart,

$$\rho_\tau(X_i, X_j) = E[\text{sign}((X_i - \tilde{X}_i)(X_j - \tilde{X}_j))]$$

where $(\tilde{X}_i, \tilde{X}_j)$ has the same distribution but is independent of (X_i, X_j) .

For Archimedean multivariate copulas, there is only one parameter to estimate, τ (or its function θ), although for m variables there are $m(m-1)/2$ unique pairwise correlation coefficients. Denote the map from ρ_τ to θ by $\theta = \hat{\theta}(\rho_\tau)$. To aggregate the map, take a simple arithmetic average:

$$\hat{\theta} = \frac{2}{m(m-1)} \sum_{1 \leq i < j \leq m} \hat{\theta}[\hat{\rho}_\tau(X_i, X_j)]$$

Nonlinear Optimization Options

PROC CCOPULA uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. In the PROC CCOPULA statement, you can specify nonlinear optimization options that are then passed to the NLO subsystem. For a list of all the nonlinear optimization options, see *SAS/ETS User's Guide*.

Displayed Output

PROC CCOPULA produces the displayed output that is described in the following sections.

Optimization Start and Resulting Parameter Estimates

If you specify the ITPRINT option in the PROC CCOPULA statement, PROC CCOPULA displays two tables, “Optimization Start Parameter Estimates” and “Optimization Results Parameter Estimates.” Each table contains the following information for each model parameter:

- parameter number
- parameter name
- parameter estimate
- gradient of the objective function at the initial parameter values

In addition to this information, the “Optimization Start Parameter Estimates” table contains the following columns:

- lower-bound constraint
- upper-bound constraint

The value of the objective function at the parameter values is displayed below each table.

Iteration History for Parameter Estimates

If you specify the ITPRINT option in the PROC CCOPULA statement, PROC CCOPULA displays a table that contains the following information for each iteration. Note that some information is specific to the model-fitting method that you choose (for example, Newton-Raphson, trust region, or quasi-Newton method).

- iteration number
- number of restarts since the fitting began
- number of function calls
- number of active constraints at the current solution

- value of the objective function (-1 times the log-likelihood value) at the current solution
- change in the objective function from the previous iteration
- value of the maximum absolute gradient element
- step size (for Newton-Raphson and quasi-Newton methods)
- slope of the current search direction (for Newton-Raphson and quasi-Newton methods)
- lambda (for trust region method)
- radius value at the current iteration (for trust region method)

Model Fit Summary

The “Model Fit Summary” table contains the following information:

- number of observations used
- number of missing values in the data table, if any
- data table name
- type of model that was fit
- log-likelihood value at the solution
- maximum absolute gradient at the solution
- number of iterations
- optimization method
- value of Akaike’s information criterion (AIC) at the solution (a smaller value indicates better fit)
- value of Schwarz Bayesian criterion (SBC) at the solution (a smaller value indicates better fit)

Below the “Model Fit Summary” table is a statement about whether the algorithm successfully converged.

Parameter Estimates

The “Parameter Estimates” table contains the estimates of the model parameters. For the normal copula, this table is not displayed because the only parameters are in the correlations matrix, which is displayed in the “Pearson Correlations Matrix” table. For the t copula, the parameter is the number of degrees of freedom; in the table it is called “DF.” For Archimedean copulas such as Clayton, Frank, and Gumbel, the parameter is called “theta.”

Pearson Correlations Matrix

The “Pearson Correlations Matrix” table contains the estimates of the model Pearson correlations matrix. This table is displayed only for the normal and t copulas. Row and column names come from the list of variables that are defined in the VAR statement.

Kendall Correlations Matrix

The “Kendall Correlations Matrix” table contains the estimates of the model Kendall correlations matrix. This table is displayed only for the normal and t copulas. Row and column names come from the list of variables that are defined in the VAR statement.

Spearman Correlations Matrix

The “Spearman Correlations Matrix” table contains the estimates of the model Spearman correlations matrix. This table is displayed only for the normal copula. Row and column names come from the list of variables that are defined in the VAR statement.

OUTPSEUDO= and OUTUNIFORM= Data Tables

The number of columns and the names of columns in the OUTPSEUDO= and OUTUNIFORM= data tables match the number and names of the variables in the VAR statement.

ODS Table Names

PROC CCOPULA assigns a name to each table that it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data tables. These table names are listed in [Table 6.3](#).

Table 6.3 ODS Tables Produced by PROC CCOPULA

Table Name	Description	Option
ODS Tables Created by the FIT Statement		
ConvergenceStatus	Convergence status	Default
PearsonCorrelations	Pearson correlations matrix estimates	Default with elliptical copulas
KendallCorrelations	Kendall correlations matrix estimates	Default with elliptical copulas
SpearmanCorrelations	Spearman correlations matrix estimates	Default with normal copula
FitSummary	Summary of nonlinear estimation	Default
ParameterEstimates	Parameter estimates	Default
ConvergenceStatus	Convergence status	ITPRINT
InputOptions	Input options	ITPRINT

Table 6.3 *continued*

Table Name	Description	Option
IterHist	Iteration history	ITPRINT
IterStart	Optimization start	ITPRINT
IterStop	Optimization results	ITPRINT
ParameterEstimatesResults	Parameter estimates	ITPRINT
ParameterEstimatesStart	Parameter estimates	ITPRINT
ODS Tables Created by the PRINTTIMING Option		
TimingDetails	Detailed summary of time used for all phases of execution	PRINTTIMING=(DETAILS)
TimingSummary	Summary of time used for main phases of execution	PRINTTIMING
ODS Table Created by the SIMULATE Statement		
SimulateModelSummary	Summary of simulation	Default

ODS Graph Names

PROC CCOPULA assigns a name to each graph that it creates by using ODS. You can use these names to selectively reference the graphs. The names are listed in [Table 6.4](#).

Table 6.4 ODS Graphics Produced by PROC CCOPULA

ODS Graph Name	Plot Description	PLOTS= Option
Fit Plots		
MatrixPlotOrig	Matrix of correlation plots	PLOTS=(DATATYPE=ORIGINAL)
MatrixPlotUnif	Matrix of transformed correlation plots	PLOTS=(DATATYPE=UNIFORM)
ScatterPlotOrig	Individual correlation plots	PLOTS(UNPACK)=(DATATYPE=ORIGINAL)
ScatterPlotUnif	Individual transformed correlation plots	PLOTS(UNPACK)=(DATATYPE=UNIFORM)
Simulation Plots		
MatrixPlotSUnif	Matrix of uniform correlation plots	PLOTS=(DATATYPE=UNIFORM)
ScatterPlotSUnif	Individual uniform correlation plots	PLOTS(UNPACK)=(DATATYPE=UNIFORM)

Examples: CCOPULA Procedure

Example 6.1: Simulating Default Times

Suppose the correlations structure required for a normal copula function is already given. For example, it can be estimated from the historic data on default times in some set of industries, but this stage is not in the scope of this example. The correlations structure is saved in a SAS data set called `lnparm`. The following statements and their output in [Output 6.1.1](#) show that the correlation parameter for Y1 and Y2 is set at 0.8:

```
proc print data = lnparm;
run;
```

Output 6.1.1 CCopula Pearson Correlations Matrix

Obs	Y1	Y2
1	1.0	0.8
2	0.8	1.0

Now you use PROC CCOPULA to simulate the data. The VAR statement specifies the list of variables to contain simulated data. The DEFINE statement assigns the name COP and specifies a normal copula that reads the correlations matrix from the `lnparm` data table.

The SIMULATE statement refers to the COP label defined in the VAR statement and specifies some options: the NDRAWS= option specifies a sample size, the SEED= option specifies 1234 as the random number generator seed, and the OUTUNIFORM=NORMAL_UNIFDATA option names the output data table for the result of simulation in uniforms.

```
/* simulate the data from bivariate normal copula */
proc ccopula ;
  var Y1-Y2;
  define cop normal (corr=mycas.lnparm);
  simulate cop /
    ndraws      = 1000000
    seed        = 1234
    outuniform  = mycas.normal_unifdata;
run;
```

The next DATA step transforms the variables from zero-one uniformly distributed to nonnegative exponentially distributed with parameter 0.5. Three indicator variables are added to the data table. SURVIVE1 and SURVIVE2 are equal to 1 if a respective company has remained in business for more than three years. SURVIVE is equal to 1 if both companies survived the same period together.

```
/* default time has exponential marginal distribution with parameter 0.5 */
data default;
  set mycas.normal_unifdata;
  array arr{2} Y1-Y2;
  array time{2} time1-time2;
  array surv{2} survive1-survive2;
  lambda = 0.5;
  do i=1 to 2;
```

```

time[i] = -log(1-arr[i])/lambda;
surv[i] = 0;
if (time[i] >3) then surv[i]=1;
end;
survive = 0;
if (time1 >3) && (time2 >3) then survive = 1;
run;

```

The first analysis step is to look at correlations between survival times of two companies. This step is performed with the following CORR procedure:

```

proc corr data = default pearson kendall;
var time1 time2;
run;

```

The output of this code is given in [Output 6.1.2](#).

[Output 6.1.2](#) shows some descriptive statistics and two measures of correlation: Pearson and Kendall. Both of these measures indicate high and statistically significant dependence between life spans of two companies.

Output 6.1.2 Default Time Descriptive Statistics and Correlations

The CORR Procedure

2 Variables: time1 time2

The second and final step is to empirically estimate the default probabilities of two companies. This is done using the FREQ procedure as follows:

```

proc freq data=default;
table survive survive1-survive2;
run;

```

The result is shown in [Output 6.1.3](#).

Output 6.1.3 Probabilities of Default

The FREQ Procedure

survive	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	852061	85.21	852061	85.21
1	147939	14.79	1000000	100.00

[Output 6.1.3](#) shows that the empirical default probabilities are 77.59% and 77.63%. Assuming that these companies are independent gives the probability estimate of both companies defaulting during the period of three years as: $0.776 \times 0.776 = 0.602$ (60.2%). Comparing this naive estimate with the much higher actual 85.2% joint default probability illustrates that neglecting the correlation between the two companies significantly underestimates the probability of default.

Example 6.2: Copula Analysis of Stock Returns

In this example, the correlations of daily returns on the large capital stocks Microsoft, Coca-Cola, IBM, Duke Energy, and British Petroleum are modeled using a t copula. To model the correlations, these companies' stock prices are first converted into returns by using the following DATA step:

```
data allreturns;
    format date MMDDYY10.;
    input date:MMDDYY10. price_ibm price_msft price_bp price_ko price_duk;
    ret_msft = price_msft/lag(price_msft)-1;
    ret_ko   = price_ko/lag(price_ko)-1;
    ret_ibm  = price_ibm/lag(price_ibm)-1;
    ret_duk  = price_duk/lag(price_duk)-1;
    ret_bp   = price_bp/lag(price_bp)-1;
    if ret_msft ^= .;
    keep date ret_ibm ret_msft ret_bp ret_ko ret_duk;
datalines;
01/02/2008    99.91    33.48    62.78    56.91    17.13
01/03/2008   100.11    33.62    63.98    57.50    17.19
01/04/2008    96.52    32.68    63.05    57.61    17.18
01/07/2008    95.49    32.90    64.18    58.97    17.62

... more lines ...
```

The stock return data can then be visualized and used to estimate the parameters of a t copula by using the following code:

```
/* Copula estimation and simulation of returns */
proc ccopula data = mycas.allreturns;
    var ret_ibm ret_msft ret_bp ret_ko ret_duk;
    * fit t-copula to stock returns;
    fit T / marginals = empirical
           method     = MLE
           plots = (datatype = both);
run;

proc ccopula;
    var x1-x5;
    define t t(df=5);
    simulate t / ndraws=603 seed=100 outuniform=mycas.copuniform plots;
run;

proc ccopula data=mycas.copuniform;
    var x1-x5;
    fit T / marginals = uniform
           method     = MLE
           plots = (datatype = uniform);
run;
```

The output from these statements is shown in [Output 6.2.1](#).

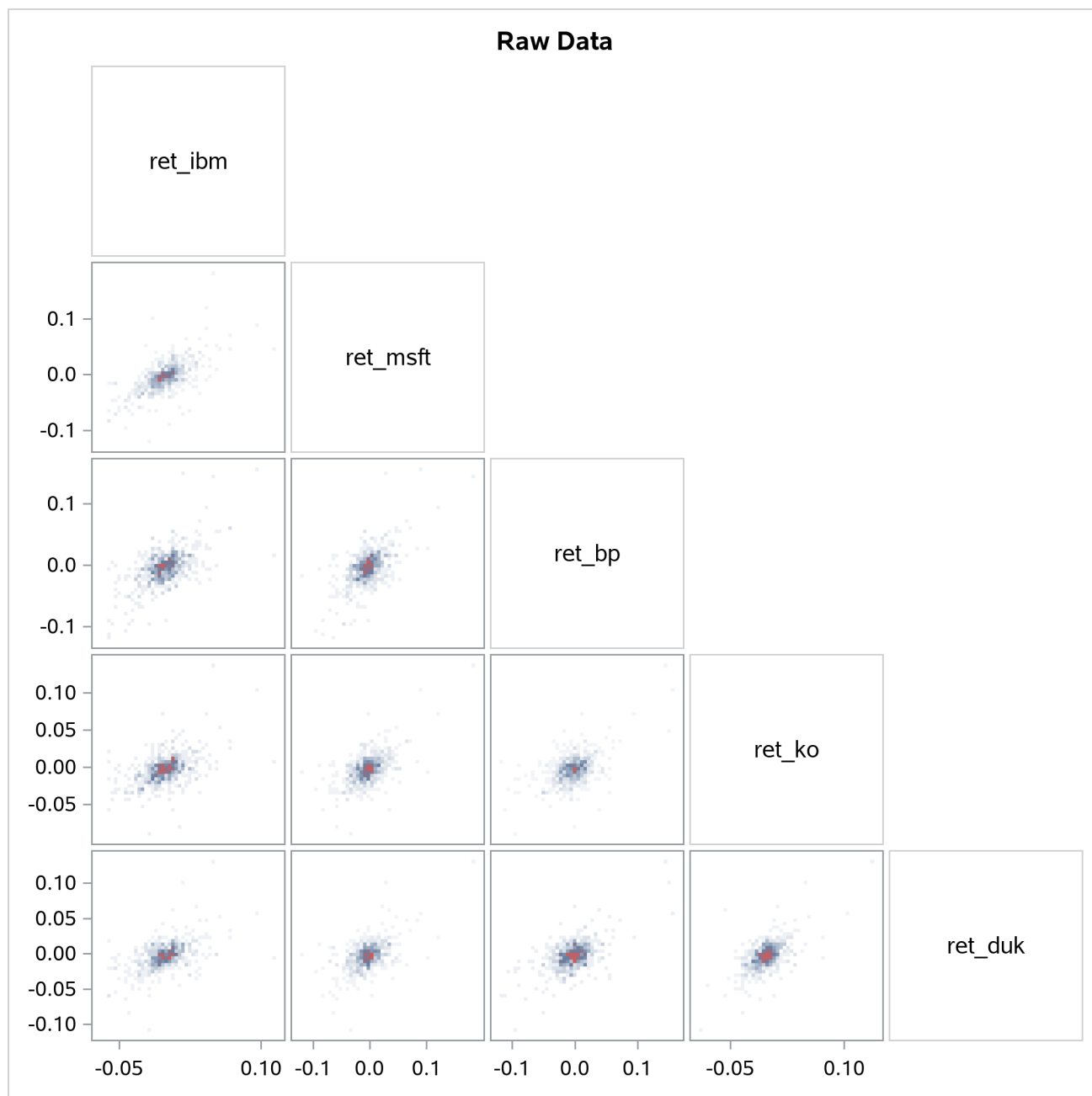
Output 6.2.1 Copula Estimation**The CCOPULA Procedure**

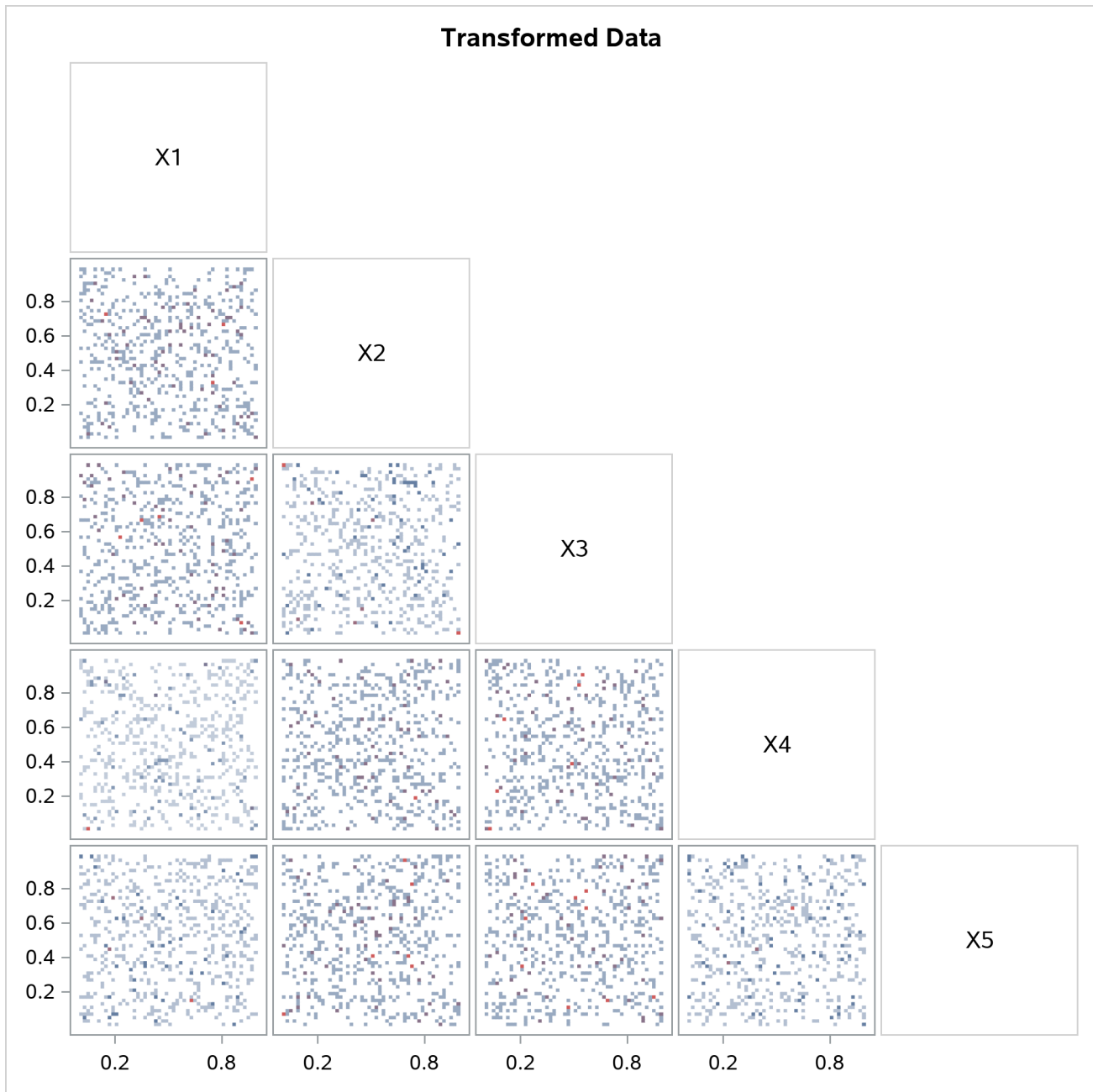
Model Fit Summary	
Number of Observations	603
Data Set	ALLRETURNS
Copula Type	T
Fit Method	MLE
Marginals	Empirical
Empirical Approximation	Adaptive Binning
Adaptive Refinement Resolution	1000
Adaptive Interpolation Method	Linear
Number of Adaptive Iterations	4
Log Likelihood	634.6931
Maximum Absolute Gradient	6.111E-8
Number of Iterations	7
Optimization Method	Newton-Raphson
AIC	-1247.39
SBC	-1198.97

The first table in [Output 6.2.1](#), “Model Fit Summary,” provides some general description of copula model estimation. The second table, “Parameter Estimates,” provides point estimates and inference about copula parameters. In this example, the only parameter in this table is the number of degrees of freedom in the multivariate t distribution. The last table, “Correlation Matrix,” contains estimates of copula model parameters.

The graphical output from the preceding statements is shown in [Output 6.2.2](#) and [Output 6.2.3](#).

Output 6.2.2 Original Data



Output 6.2.3 Original Data Transformed into Uniform Marginals

Note that in [Output 6.2.2](#), the most elliptical scatter plot, which is between IBM and Microsoft, indicates the strongest dependence. Similarly, in [Output 6.2.3](#), graphs that are denser along the diagonal indicate the same thing.

References

- Cherubini, U., Luciano, E., and Vecchiato, W. (2004). *Copula Methods in Finance*. Chichester, UK: John Wiley & Sons.
- Devroye, L. (1986). *Non-uniform Random Variate Generation*. New York: Springer-Verlag. <http://luc.devroye.org/rnbookindex.html>.
- Marshall, A. W., and Olkin, I. (1988). “Families of Multivariate Distributions.” *Journal of the American Statistical Association* 83:834–841.
- McNeil, A., Frey, R., and Embrechts, P. (2005). *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton, NJ: Princeton University Press.
- Nolan, J. P. (2010). *Stable Distributions: Models for Heavy Tailed Data*. Boston: Birkhäuser.
- Sklar, A. (1959). “Fonctions de répartition à n dimensions et leurs marges [Distribution functions with n dimensions and their margins].” *Publications de l’Institut de Statistique de L’Université de Paris* 8:229–231.
- Wu, F., Valdez, E., and Sherris, M. (2007). “Simulating from Exchangeable Archimedean Copulas.” *Communications in Statistics—Simulation and Computation* 36:1019–1034.

Chapter 7

The CESM Procedure

Contents

Overview: CESM Procedure	250
Using CAS Sessions and CAS Engine Librefs	250
Getting Started: CESM Procedure	251
Syntax: CESM Procedure	252
Functional Summary	252
PROC CESM Statement	253
BY Statement	255
FORECAST Statement	255
ID Statement	257
Details: CESM Procedure	261
Accumulation	261
Missing Value Interpretation	263
Transformations	263
Parameter Estimation	263
Missing Value Modeling Issues	264
Forecasting	264
Inverse Transformations	264
Data Table Output	264
Printed Output	269
Examples: CESM Procedure	269
Example 7.1: Forecasting of Time Series Data	269
Example 7.2: Forecasting of Transactional Data	277
Example 7.3: Specifying the Forecasting Model	281

Overview: CESM Procedure

The CESM procedure generates forecasts by using exponential smoothing models with optimized smoothing weights for one or more time series. Time series data have observations that are equally spaced by a specific time interval (for example, monthly or weekly).

PROC CESM can also generate these forecasts for transactional data, whose observations are not spaced with respect to any particular time interval. The procedure accumulates transactional data on the basis of a specified (or default) time interval to form a time series prior to modeling and forecasting. Typical examples of transactional data are internet, inventory, sales, and other similar data.

For typical time series, you can use the following smoothing models: simple, double, linear, damped trend, seasonal, and Winters method (additive or multiplicative). In addition, transformed versions of the following models are provided: log, square root, logistic, and Box-Cox.

The CESM procedure writes the following to output data tables: the time series that are extrapolated by the forecasts, the series summary statistics, the forecasts and confidence limits, and the parameter estimates.

PROC CESM requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the `CAS` statement as follows:

```
cas mysess terminate;
```

For more information about the `CAS` and `LIBNAME` statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CESM Procedure

The CESM procedure is simple to use and does not require in-depth knowledge of forecasting methods. It can provide results in output data tables. The examples in this section are more fully illustrated in “[Examples: CESM Procedure](#)” on page 269.

The following statements generate forecasts for every numeric variable in the time series in the input data table `mycas.sales`, which contains sales data that are recorded monthly:

```
proc cesm data=mycas.sales outfor=mycas.nextyear;  
  id date interval=month;  
  forecast _numeric_ / lead=12;  
run;
```

The `DATA=` option specifies `mycas.sales` as the input data table, and the `OUTFOR=` option requests that the forecast be output to the `mycas.nextYear` table. The `ID` statement specifies the variable that represents time (Date) and specifies that the time series data are spaced monthly. The `_NUMERIC_` specification in the `FORECAST` statement requests that every numeric variable in the input data table be forecast, and the `LEAD=12` options requests that the time series be forecast for the next 12 months.

The following statements show how you can forecast data that contain transactional variables that are not recorded at any specific frequency:

```
proc cesm data=mycas.websites outfor=mycas.nextweek;  
  id time interval=dtday accumulate=total;  
  forecast boats cars planes / lead=7;  
run;
```

The `DATA=` option specifies the input data table `mycas.websites`, which contains three variables (Boats, Cars, and Planes) in internet data that are recorded at no particular time interval. The `OUTFOR=` option requests that forecasts for the total daily values be recorded in the `mycas.nextWeek` data table. The `ID` statement specifies Time (which records the time of a website hit) as the variable that represents time and requests that the data be accumulated into a daily time series. The `FORECAST` statement requests that forecasts for the Boats, Cars, and Planes variables be generated. The `LEAD=7` option requests that forecasts be generated for the next seven days. Because the `METHOD=` option is not specified in the `FORECAST` statement, `PROC CESM` fits each series by selecting the best exponential smoothing model (which is the default model) according to the default criterion of root mean square error (RMSE).

Syntax: CESM Procedure

The following statements are available in the CESM procedure:

```
PROC CESM options ;
  BY variables ;
  ID variable INTERVAL= interval < options > ;
  FORECAST variable-list / < options > ;
```

Functional Summary

Table 7.1 summarizes the statements and options that are available in the CESM procedure.

Table 7.1 Functional Summary

Description	Statement	Option
Statements		
Specifies data tables and options	PROC CESM	
Specifies BY-group processing	BY	
Specifies variables to forecast	FORECAST	
Specifies the time ID variable	ID	
Data Set Options		
Specifies the input data table	PROC CESM	DATA=
Specifies the output data table	PROC CESM	OUT=
Specifies the parameter output data table	PROC CESM	OUTEST=
Specifies the forecast output data table	PROC CESM	OUTFOR=
Specify statistics output data table	PROC CESM	OUTSTAT=
Specifies the summary output data table	PROC CESM	OUTSUM=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of seasonal cycle	PROC CESM	SEASONALITY=
Specifies the interval alignment	ID	ALIGN=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, FORECAST	ACCUMULATE=
Specifies how to interpret missing values	ID, FORECAST	SETMISSING=
Forecasting Horizon, Holdback Options		
Specifies the data to hold back	FORECAST	BACK=
Specifies the forecast horizon or lead	FORECAST	LEAD=
Forecasting Model Options		
Specifies the confidence limit width	FORECAST	ALPHA=
Specifies the model selection criterion	FORECAST	CRITERION=

Table 7.1 *continued*

Description	Statement	Option
Specifies the forecast model	FORECAST	METHOD=
Specifies the median forecasts	FORECAST	MEDIAN
Specifies the model transformation	FORECAST	TRANSFORM=
Miscellaneous Options		
Specifies the time ID format	ID	FORMAT=
Specifies the method for trimming the data in the BY groups	ID	TRIMID=

The following sections describe the PROC CESM statement and then describe the other statements in alphabetical order.

PROC CESM Statement

PROC CESM *options* ;

You can specify the following *options*:

DATA=*CAS-libref.data-table*

names the input data table for PROC CESM to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

data-table specifies the name of the input data table.

OUT=*CAS-libref.data-table*

names the output data table to contain the forecasts of the variables that are specified in subsequent FORECAST statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

The *variable* that is specified in the ID statement is also included in the output data set. The values are accumulated based on the ACCUMULATE= option. The OUT= data table is particularly useful in extending the independent variables.

OUTEST=CAS-libref.data-table

names the output data table to contain the model parameter estimates and the associated test statistics and probability values. The OUTEST= data table is useful for evaluating the significance of the model parameters and for understanding the model dynamics.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

OUTFOR=CAS-libref.data-table

names the output data table to contain the forecast time series components (actual, predicted, lower confidence limit, upper confidence limit, prediction error, and prediction standard error). The OUTFOR= data table is useful for displaying the forecasts in tabular or graphical form.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

OUTSTAT=CAS-libref.data-table

names the output data table to contain the statistics of fit (or goodness-of-fit statistics). The OUTSTAT= data table is useful for evaluating how well the model fits the series. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

OUTSUM=CAS-libref.data-table

names the output data table to contain the summary statistics and the forecast summation. The summary statistics are based on the accumulated time series when the ACCUMULATE= or SETMISSING= option is specified in the FORECAST statement. The OUTSUM= data table is useful when large numbers of series are forecasted and a summary of the results is needed.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 250.

SEASONALITY=number

specifies the length of the seasonal cycle. For example, SEASONALITY=3 means that every group of three observations forms a seasonal cycle. This option is applicable only for seasonal forecasting models. By default, the length of the seasonal cycle is 1 (no seasonality) or the length that is implied by the INTERVAL= option in the ID statement. For example, INTERVAL=MONTH implies that the length of the seasonal cycle is 12.

BY Statement

BY *variables* ;

You can use a BY statement to obtain separate dummy variable definitions for groups of observations that are defined by the BY variables.

FORECAST Statement

FORECAST *variable-list* / < *options* > ;

The FORECAST statement lists the numeric variables in the input data table whose accumulated values represent time series to be modeled and forecast. (The input data table is specified in the DATA= option in the PROC CESM statement.)

You must specify a *variable-list* that contains one or more numeric variables. For more information about the *variable-list*, see the section “SAS Variable Lists” in *SAS Programmers Guide: Essentials*.

You can specify any number of FORECAST statements, but you cannot specify the same variable in more than one of them.

You can specify the following *options* to specify which forecast model to use:

ACCUMULATE=*option*

specifies how to accumulate the data table observations within each time period for the variables in the *variable-list*. If the ACCUMULATE= option is not specified in the FORECAST statement, accumulation is determined by the ACCUMULATE= option in the ID statement. Use the ACCUMULATE= option with multiple FORECAST statements when you want different accumulation specifications for different variables. For more information, see the [ACCUMULATE=](#) option in the ID statement.

ALPHA=*number*

specifies the significance level to use in computing the confidence limits of the forecast, where *number* must be between 0 and 1. By default, ALPHA=0.05, which produces 95% confidence intervals.

BACK=*n*

specifies the number of observations before the end of the data where the multistep forecasts are to begin. By default, BACK=0.

CRITERION=*criterion*

specifies the model selection criterion (statistic of fit) to use to select from several candidate models. For a list of valid values, see the CRITERION= option in the HPFDIAGNOSE procedure in *SAS Forecast Server Procedures: User's Guide*. By default, CRITERION=RMSE.

LEAD=*n*

specifies the number of periods ahead to forecast (the forecast lead or horizon).

The value *n* is not relative to the last nonmissing observation of a particular series, but is instead relative to the BACK= option specification to the last observation in the input data table or the accumulated series. Thus, if a series has missing values at the end, the actual number of forecasts computed for that series is greater than *n*.

By default, LEAD=0.

MEDIAN

estimates the median forecast values and uses those values for forecasting. (By default, PROC CESM uses mean values for forecasting.) If you do not specify the TRANSFORM= option, no transformation is applied to the time series, so the mean and median forecast values are identical.

METHOD=*model-name*

specifies the forecasting model to use to forecast the time series. You can specify the following forecasting *model-names*:

ADDWINTERS	requests the Winters additive method.
BEST	requests the best candidate smoothing model among the SIMPLE, LINEAR, DAMPTREND, SEASONAL, ADDWINTERS, or WINTERS methods.
BESTN	requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, or DAMPTREND methods.
BESTS	requests the best candidate seasonal smoothing model among the SEASONAL, ADDWINTERS, or WINTERS methods.
DAMPTREND	requests damped trend exponential smoothing.
DOUBLE	requests second-order exponential smoothing.
LINEAR	requests linear (Holt) exponential smoothing.
MULTSEASONAL	requests multiplicative seasonal exponential smoothing.
SEASONAL	requests additive seasonal exponential smoothing.
SIMPLE	requests simple (single) exponential smoothing.
WINTERS	requests Winters multiplicative method.

By default, METHOD=BEST.

SETMISSING=*option* | *number*

specifies how to assign missing values (either input or accumulated) in the accumulated time series for variables in the *variable-list*. If the SETMISSING= option is not specified in the FORECAST statement, missing values are set according to the value of the SETMISSING= option in the ID statement. For more information, see the SETMISSING= option in the ID statement.

TRANSFORM=*option*

specifies the time series transformation to be applied to the input or accumulated time series. You can specify the following values for *option*:

AUTO	automatically chooses between NONE and LOG on the basis of the model selection criteria.
BOXCOX (<i>n</i>)	performs Box-Cox transformation with parameter number (<i>n</i>), where <i>n</i> must be between -5 and 5.
LOG	performs logarithmic transformation.
LOGISTIC	performs logistic transformation.
NONE	performs no transformation.
SQRT	performs square-root transformation.

By default, TRANSFORM=NONE.

When the TRANSFORM= option is specified, the time series must be strictly positive. After the time series is transformed, the model parameters are estimated by using the transformed series. The forecasts of the transformed series are then computed, and finally the transformed series forecasts are inverse-transformed. The inverse transform produces either mean or median forecasts depending on whether the MEDIAN option is specified. For more information, see the sections “[Transformations](#)” on page 263 and “[Inverse Transformations](#)” on page 264.

ID Statement

ID *variable* **INTERVAL**=*interval* < *options* > ;

The ID statement names a numeric *variable* that identifies observations in the input and output data tables. The values of the ID *variable* are assumed to be SAS date or datetime values. In addition, the ID statement specifies (in the INTERVAL= option) the frequency that is associated with the time series. The *options* also specify how to accumulate the observations and how to align the time ID values to form the time series to be forecast. The specified information affects all variables that are specified in subsequent FORECAST statements. If an ID statement is not specified, the observation number (with respect to the BY group) is used as the time ID.

You must specify the following argument:

INTERVAL=*interval*

specifies the frequency of the input time series or of the time series to be accumulated from the input data. For example, if the input data table consists of quarterly observations, then INTERVAL=QTR should be used. If the SEASONALITY= option is not specified, the length of the seasonal cycle is implied by the INTERVAL= option. For example, INTERVAL=QTR implies a seasonal cycle of length 4. If the ACCUMULATE= option is also specified, the INTERVAL= option determines the time periods for the accumulation of observations.

The basic intervals are YEAR, SEMIYEAR, QTR, MONTH, SEMIMONTH, TENDAY, WEEK, WEEKDAY, DAY, HOUR, MINUTE, and SECOND. For more information about the intervals that can be specified, see the chapter “Date Intervals, Formats, and Functions” in [SAS/ETS User’s Guide](#).

You can also specify the following *options*:

ACCUMULATE=*option*

specifies how to accumulate the data table observations within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID *variable* contains the time ID values. Each value of the time ID *variable* value corresponds to a specific time period. The accumulated values form the time series, which is used in subsequent model fitting and forecasting.

The ACCUMULATE= option is particularly useful when there are gaps in the input data or when there are multiple input observations that coincide with a particular time period (for example, transactional data). The EXPAND procedure in [SAS/ETS User’s Guide](#) offers additional frequency conversions and transformations that can also be useful in creating a time series.

You can specify the following *options* to determine how to accumulate the observations within each time period based on the ID *variable* and the frequency that is specified in the INTERVAL= option:

TOTAL SUM	accumulates observations based on the total sum of their values.
AVERAGE AVG	accumulates observations based on the average of their values.
MINIMUM MIN	accumulates observations based on the minimum of their values.
MAXIMUM MAX	accumulates observations based on the maximum of their values.
N	accumulates observations based on the number of nonmissing observations.
NMISS	accumulates observations based on the number of missing observations.
STDDEV STD	accumulates observations based on the standard deviation of their values.
CSS	accumulates observations based on the corrected sum of squares of their values.
USS	accumulates observations based on the uncorrected sum of squares of their values.

By default, **ACCUMULATE**=**TOTAL**.

If the **ACCUMULATE**= option is specified, the **SETMISSING**= option is useful for specifying how to treat accumulated missing values. If missing values should be interpreted as 0, then you should specify **SETMISSING**=0. For more information about accumulation, see the section “[Accumulation](#)” on page 261.

ALIGN=*option*

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, this option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations that are supplied by the user, this option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often are not supplied by the user. You can specify the following *options*:

BEGINNING BEG B	represents each time period by using the beginning SAS date or datetime value of the time period.
ENDING END E	represents each time period by using the ending SAS date or datetime value of the time period.
MIDDLE MID M	represents each time period by using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, **ALIGN**=**BEGINNING**.

END=*date | datetime*

specifies a SAS date or datetime literal value that represents the end of the data. If the value of the last time ID variable is less than the **END**= value, the series is extended with missing values. If the value of the last time ID variable is greater than the **END**= value, the series is truncated. For example, **END**= '1jan2008'D requests that data for time periods after the first of January 2008 not be used. The option **END**= "&sysdate"D uses the automatic macro variable **SYSDATE** to extend or truncate the series to the current date. You can use this option and the **START**= option to ensure that data that are associated with each **BY** group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If the FORMAT= option is not specified, the default format is implied by the INTERVAL= option.

SETMISSING=*n* | *option*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can specify either a number (*n*) or an *option* to determine how to interpret missing values:

<i>n</i>	interprets a missing value as having the value <i>n</i> . You can specify any number for <i>n</i> , but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because no recorded data usually implies no activity.
MISSING	interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
AVERAGE AVG	interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
MINIMUM MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*date* | *datetime*

specifies a SAS date or datetime literal value that represents the beginning of the data. If the value of the first time ID variable is greater than the START= value, the series is prefixed with missing values. If the value of the first time ID variable is less than the START= value, the series is truncated. You can use this option and the END= option to ensure that data that are associated with each BY group contain the same number of observations.

TRIMID=method

specifies the method for trimming the data in the BY groups. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable is recalculated according to the value of the **LEAD=** option in the FORECAST statement (if the LEAD= option is specified).

You can specify one of the following *methods*:

- NONE** uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
- LEFT** uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.
- RIGHT** uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.
- BOTH** uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

Details: CESM Procedure

You can use the CESM procedure to forecast both time series data and transactional data. If the data are transactional, then the procedure must first accumulate the data into a time series before it can forecast them. [Table 7.2](#) shows the sequential steps that the procedure uses to produce forecasts; the options that control each step and the statements where those options can be specified are shown in the third and fourth columns, respectively.

Table 7.2 CESM Processing Steps and Control Options

Step	Operation	Option	Statements
1	Accumulation	ACCUMULATE=	ID, FORECAST
2	Missing value interpretation	SETMISSING=	ID, FORECAST
3	Transformations	TRANSFORM=	FORECAST
4	Parameter estimation	METHOD=	FORECAST
5	Forecasting	METHOD=, LEAD=	FORECAST
6	Inverse transformation	TRANSFORM, MEDIAN	FORECAST

Each of the steps shown in [Table 7.2](#) is described in the following sections.

Accumulation

If the ACCUMULATE= option is specified in the ID statement, data table observations are accumulated within each time period. The frequency (width of each time interval) is specified in the INTERVAL= option in the ID statement, and the variable that is specified in the ID statement contains the time ID values. Each time ID value corresponds to a specific time period. Accumulation is particularly useful when the input data table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series that is used in subsequent analyses by the CESM procedure.

For example, suppose a data table contains the following observations:

```

19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
23MAY1999    20

```

If the INTERVAL=MONTH option is specified in the ID statement, all the preceding observations fall within three time periods: March 1999, April 1999, and May 1999. The observations are accumulated within each time period as follows.

- If the ACCUMULATE=NONE option is specified, an error is generated because the ID variable values are not equally spaced with respect to the specified frequency (MONTH).
- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

```
O1MAR1999    40
O1APR1999    .
O1MAY1999    90
```

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    30
```

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MEDIAN option is specified, the resulting time series is

```
O1MAR1999    20
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=FIRST option is specified, the resulting time series is

```
O1MAR1999    10
O1APR1999    .
O1MAY1999    50
```

- If the ACCUMULATE=LAST option is specified, the resulting time series is

```
O1MAR1999    30
O1APR1999    .
O1MAY1999    20
```

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

O1MAR1999	14.14
O1APR1999	.
O1MAY1999	17.32

The preceding examples demonstrate that even when the data observations contain no missing values, the accumulated time series can have missing values.

Missing Value Interpretation

Sometimes missing values should be interpreted as truly unknown values and should be retained as missing values in the data table. The forecasting models used by the CESM procedure can effectively handle missing values; see the section “[Missing Value Modeling Issues](#)” on page 264. However, sometimes missing values are known, such as when missing values are created from accumulation and represent no observed values for the variable. In this case, the value for the period should be interpreted as 0 (no values), and the SETMISSING=0 option should be used to cause PROC CESM to recode missing values as 0. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and missing-value-recoded time series is used in subsequent analyses in PROC CESM.

Transformations

If the TRANSFORM= option is specified in the FORECAST statement, the time series is transformed prior to model parameter estimation and forecasting. Only strictly positive series can be transformed. An error is generated when the TRANSFORM= option is used with a nonpositive series. For more information about forecasting transformed time series, see the chapter “Forecasting Process Details” in *SAS/ETS User’s Guide*.

Parameter Estimation

All the parameters (smoothing weights) that are associated with the exponential smoothing model that is used to forecast the time series (as specified by the METHOD= option in the PROC CESM statement) are optimized based on the data, with the default parameter restrictions imposed. If the TRANSFORM= option is specified in the FORECAST statement, the transformed time series data are used to estimate the model parameters.

The techniques that the CESM procedure uses are identical to the techniques that are used for exponential smoothing models in the Time Series Forecasting System of SAS/ETS software. For more information, see the chapter “Overview of the Time Series Forecasting System” in *SAS/ETS User’s Guide*.

Missing Value Modeling Issues

The treatment of missing values varies with the forecasting model. Missing values after the start of the series are replaced with one-step-ahead predicted values, and the predicted values are used in the smoothing equations.

The treatment of missing values can also be specified in the SETMISSING= option, which changes the missing values prior to modeling.

NOTE: Even if all the observed data are nonmissing, the ACCUMULATE= option can create missing values in the accumulated series (when the data contain no observations for some of the time periods that are specified in the INTERVAL= option in the ID statement).

Forecasting

After the model parameters are estimated, one-step-ahead forecasts are generated for the full range of the accumulated and optionally transformed time series data, and multistep forecasts are generated from the end of the time series to the future time period that is specified in the LEAD= option in the PROC CESM statement. If there are missing values at the end of the time series, the forecast horizon will be greater than that specified by the LEAD= option.

Inverse Transformations

If the TRANSFORM= option is specified in the FORECAST statement, the forecasts of the transformed time series are inverse-transformed. By default, forecasts of the mean (expected value) are generated. If the MEDIAN option is specified in the PROC CESM statement, median forecasts are generated. For more information about forecasting transformed time series, see the chapter “Forecasting Process Details” in *SAS/ETS User’s Guide*.

Data Table Output

The CESM procedure can create the OUT=, OUTEST=, OUTFOR=, OUTSTAT=, and OUTSUM= data tables. These data tables contain the variables that are specified in the BY statement and statistics that are related to the variables that are listed in the FORECAST statement. In general, if a forecasting step that is related to an output data table fails, the values of this step are not recorded or are set to missing in the related output data table and appropriate error or warning (or both) messages are recorded in the log. For more information about how the variables in the output data tables are computed, see the “Smoothing Models” section in the chapter “Forecasting Process Details” in *SAS/ETS User’s Guide*.

OUT= Data Table

The OUT= data table contains the variables that are specified in the BY, ID, and FORECAST statements. If the ID statement is specified, the ID variable values are aligned and extended based on the ALIGN= and INTERVAL= options in the ID statement. The values of the variables that are specified in the FORECAST statements are accumulated based on the ACCUMULATE= option in the FORECAST statement, and missing values are interpreted based on the SETMISSING= option in the FORECAST statement.

If any of the forecasting steps fail for a particular variable, the variable is extended by missing values.

OUTEST= Data Table

The OUTEST= data table contains observations that are related to the parameter estimation step. The column names of the table are the BY variables and the following names:

NAME	variable name
MODEL	forecasting model
MODELVAR	model variable
DSVAR	data variable
VARTYPE	variable type
TRANSFORM	transformation
COMPONENT	component
COMPMODEL	component model
PARM	parameter name
FACTOR	factor
LAG	lag
SHIFT	shift
EST	value of parameter estimate
STDERR	standard error of parameter estimate
TVALUE	<i>t</i> values of parameter estimate
PVALUE	<i>p</i> -values of parameter estimate

If the parameter estimation step fails for a particular variable, no observations are output to the OUTEST= data table for that variable.

OUTFOR= Data Table

The OUTFOR= data table contains observations that are related to the forecasting step. The column names of the table are the BY variables and the following names:

NAME	variable name
ID Variable	values of the variable specified in the ID statement
ACTUAL	actual values

PREDICT	predicted values
ERROR	prediction errors
STD	prediction standard errors
UPPER	upper confidence limits
LOWER	lower confidence limits

If the forecasting step fails for a particular variable, no observations are recorded in the OUTFOR= data table for that variable. If the TRANSFORM= option is specified in the FORECAST statement, the values in the preceding variables are the inverse-transform forecasts. If the MEDIAN option is specified in the FORECAST statement, the median forecasts are stored; otherwise, the mean forecasts are stored.

OUTSTAT= Data Set

The OUTSTAT= data table contains observations that are related to the statistics of fit. The column names of the table are the BY variables and the following names:

NAME	variable name
REGION	region in which the statistics are calculated. Statistics that are calculated in the fit region are indicated by FIT. Statistics that are calculated in the forecast region (which happens only if the value of the BACK= option is greater than zero) are indicated by FORECAST.
MODEL	model name
DFE	degrees of freedom error
N	number of observations
NOBS	number of observations used
NMISSA	number of missing actuals
NMISSP	number of missing predicted values
NPARMS	number of parameters
TSS	total sum of squares
SST	corrected total sum of squares
SSE	sum of square error
MSE	mean square error
RMSE	root mean square error
UMSE	unbiased mean square error
URMSE	unbiased root mean square error
MAPE	mean absolute percentage error
MAE	mean absolute error
RSQUARE	R-square
ADJRSQ	adjusted R-square
AADJRSQ	Amemiya's adjusted R-square

RWRSQ	random walk R-square
AIC	Akaike's information criterion
AICC	finite sample corrected AIC
SBC	Schwarz Bayesian information criterion
APC	Amemiya's prediction criterion
MAXERR	maximum error
MINERR	minimum error
MAXPE	maximum percentage error
MINPE	minimum percentage error
ME	mean error
MPE	mean percentage error
MDAPE	median absolute percentage error
GMAPE	geometric mean absolute percentage error
MINPPE	minimum predictive percentage error
MAXPPE	maximum predictive percentage error
MPPE	mean predictive percentage error
MAPPE	symmetric mean absolute predictive percentage error
MDAPPE	median absolute predictive percentage error
GMAPPE	geometric mean absolute predictive percentage error
MINSPE	minimum symmetric percentage error
MAXSPE	maximum symmetric percentage error
MSPE	mean symmetric percentage error
SMAPE	symmetric mean absolute percentage error
MDASPE	median absolute symmetric percentage error
GMASPE	geometric mean absolute symmetric percentage error
MINRE	minimum relative error
MAXRE	maximum relative error
MRE	mean relative error
MRAE	mean relative absolute error
MDRAE	median relative absolute error
GMRAE	geometric mean relative absolute error
MASE	mean absolute scaled error
MINAPES	minimum absolute error percentage of standard deviation
MAXAPES	maximum absolute error percentage of standard deviation
MAPES	mean absolute error percentage of standard deviation

MDAPES	median absolute error percentage of standard deviation
GMAPES	geometric mean absolute error percentage of standard deviation

If the statistics of fit cannot be computed for a particular variable, no observations are recorded in the OUTSTAT= data table for that variable. If the TRANSFORM= option is specified, the values in the preceding variables are computed on the basis of the inverse transform forecasts. If the MEDIAN option is specified, the median forecasts are the basis; otherwise, the mean forecasts are the basis.

For more information about the calculation of forecasting statistics of fit, see the chapter “Forecasting Process Details” in *SAS/ETS User’s Guide*.

OUTSUM= Data Table

The OUTSUM= data table contains the variables that are specified in the BY statement in addition to the variables that are listed in this section. The OUTSUM= data table records the summary statistics for each variable that is specified in a FORECAST statement.

The following variables, which are related to summary statistics, are based on the ACCUMULATE= and SETMISSING= options:

NAME	variable name
NOBS	number of observations
N	number of nonmissing observations
NMISS	number of missing observations
MIN	minimum value
MAX	maximum value
MEAN	mean value
STDDEV	standard deviation
STATUS	forecasting status; nonzero values imply that no forecast was generated for the series
START	the starting date of each series
END	the ending date of each series
STARTOBS	the beginning observation number of each series
ENDOBS	the ending observation number of each series

Printed Output

The CESM procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC CESM. Printing of other results is best accomplished by the use of targeted data queries to subset and display the information from the tables that are produced by the CESM procedure. For example, you might want to print the `OUTFOR=` results only for BY groups that have a particular `_STATUS_` value in the `OUTSUM=` table.

Examples: CESM Procedure

Example 7.1: Forecasting of Time Series Data

This example uses retail sales data to illustrate how you can use the CESM procedure to forecast time series data.

The statements in this example assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

The following DATA step creates a data table from data that were recorded monthly at numerous points of sale. The data table, `mycas.sales`, contains a variable `Date` (which represents time) and a variable for each sales item. Each value of the `Date` variable is recorded in ascending order, and the values of each of the other variables represent a single time series.

```
data mycas.sales;
    format date date9.;
    input date : date9. shoes socks laces dresses
          coats shirts ties belts hats blouses;
    datalines;
01JAN1994 3557 3718 6368.80 575 987 10.8200 15.0000 102.600 12410 15013
01FEB1994 5128 4174 8123.20 565 1000 12.1200 15.1000 99.900 13556 12413
01MAR1994 5222 4482 7807.20 406 1005 11.7800 15.3000 102.000 11063 12752

    ... more lines ...

01DEC1998 5399 4795 6075.30 614 1239 21.2000 1.9000 79.700 23004 22044
01JAN1999 6405 4981 6812.10 607 1196 20.7000 14.9000 99.900 20583 26093
;
```

The following CESM procedure statements forecast each of the monthly time series by using the simple exponential smoothing model:

```
proc cesm data=mycas.sales outfor=mycas.nextyear;
    id date interval=month;
    forecast _numeric_ / lead=12 method=simple;
run;
```

The preceding statements generate forecasts for every numeric variable in the input data table `mycas.sales` for the next 12 months and store these forecasts in the output data table `mycas.nextYear`. The summary of time series processing is output in [Output 7.1.1](#).

Output 7.1.1 Summary of Time Series Processing
The CESM Procedure

Summary of time series processing for SALES	
Number of analysis variables	10
Number of rows read	122
Number of groups read	1
Memory for group packages (KB)	33
Time to load groups (seconds)	0.0168399811
Minimum time ID	JAN1994
Maximum time ID	JAN1999
Minimum time periods	61
Maximum time periods	61
Number of nodes run	4
Number of nodes with data	3
Number of nodes with groups	1
Number of threads budgeted	1
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	1
Maximum threads active	1
Number of groups processed by submitted code	1
Number of groups failing	0
Elapsed time to process groups (seconds)	0.086097002

The following statements plot the forecasts for the sale of belts, blouses, and laces:

```
%macro plotActualPredict(ds,timeId,var,refValue,
                        xStart,xEnd,xBy,xLabel,yStart,yEnd,yBy,yLabel);
  proc sgplot data=&ds.(where=(_name_="&var."));
    series x=&timeId. y=actual / markers lineattrs=(color=red)
      markerattrs=(symbol=circlefilled color=red);
    series x=&timeId. y=predict / markers lineattrs=(color=blue)
      markerattrs=(symbol=asterisk color=blue);
    reffline &refValue. / axis=x;
    xaxis values=(&xStart. to &xEnd. by &xBy.) label=&xLabel;
    yaxis values=(&yStart. to &yEnd. by &yBy.) label=&yLabel minor;
  run;
%mend;

%plotActualPredict(mycas.nextyear, date, belts, '01JAN1999'd,
  '01JAN1994'd, '01DEC2000'd, year, 'Date',
  20, 140, 20, 'Sales');
```

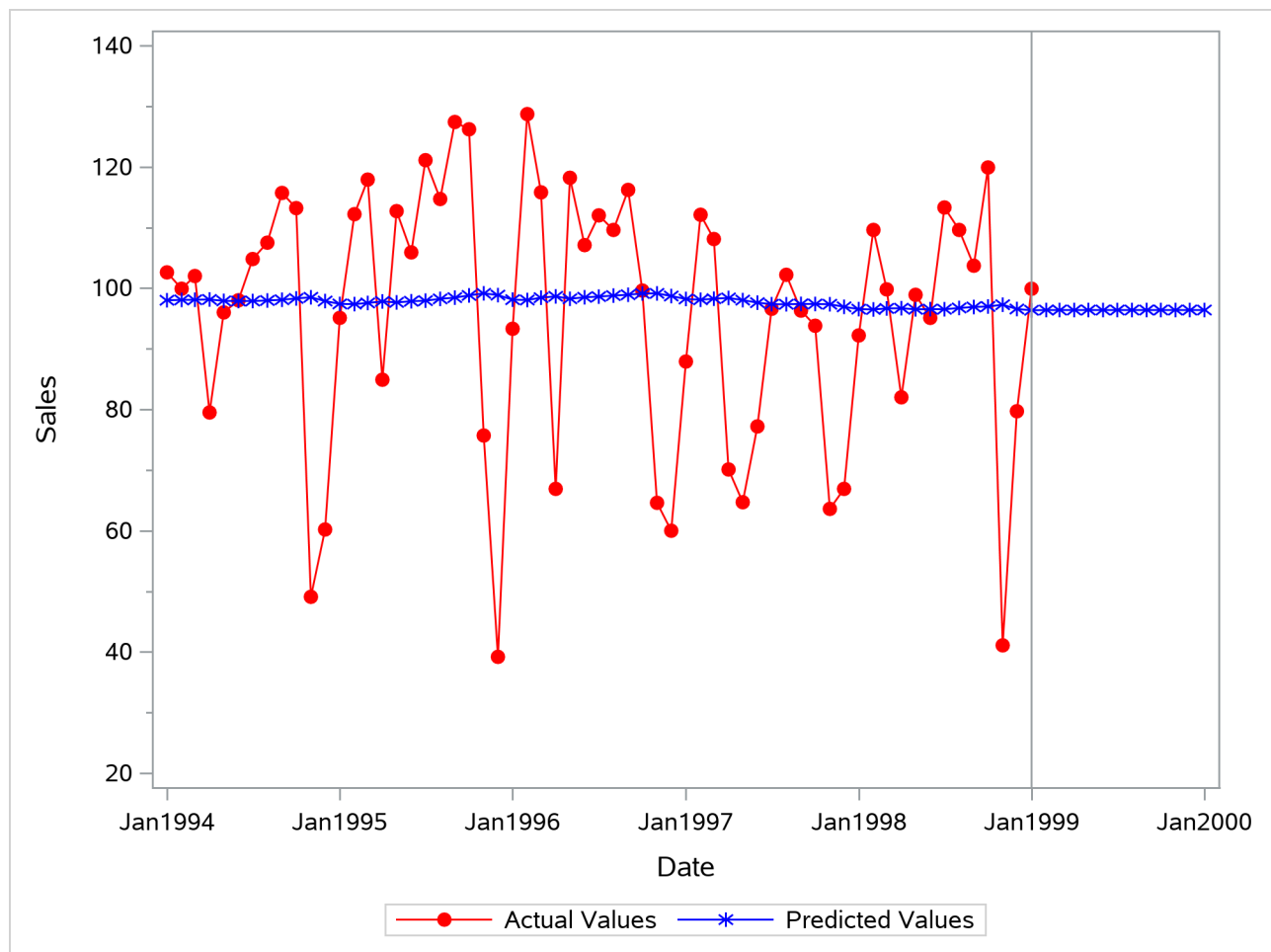


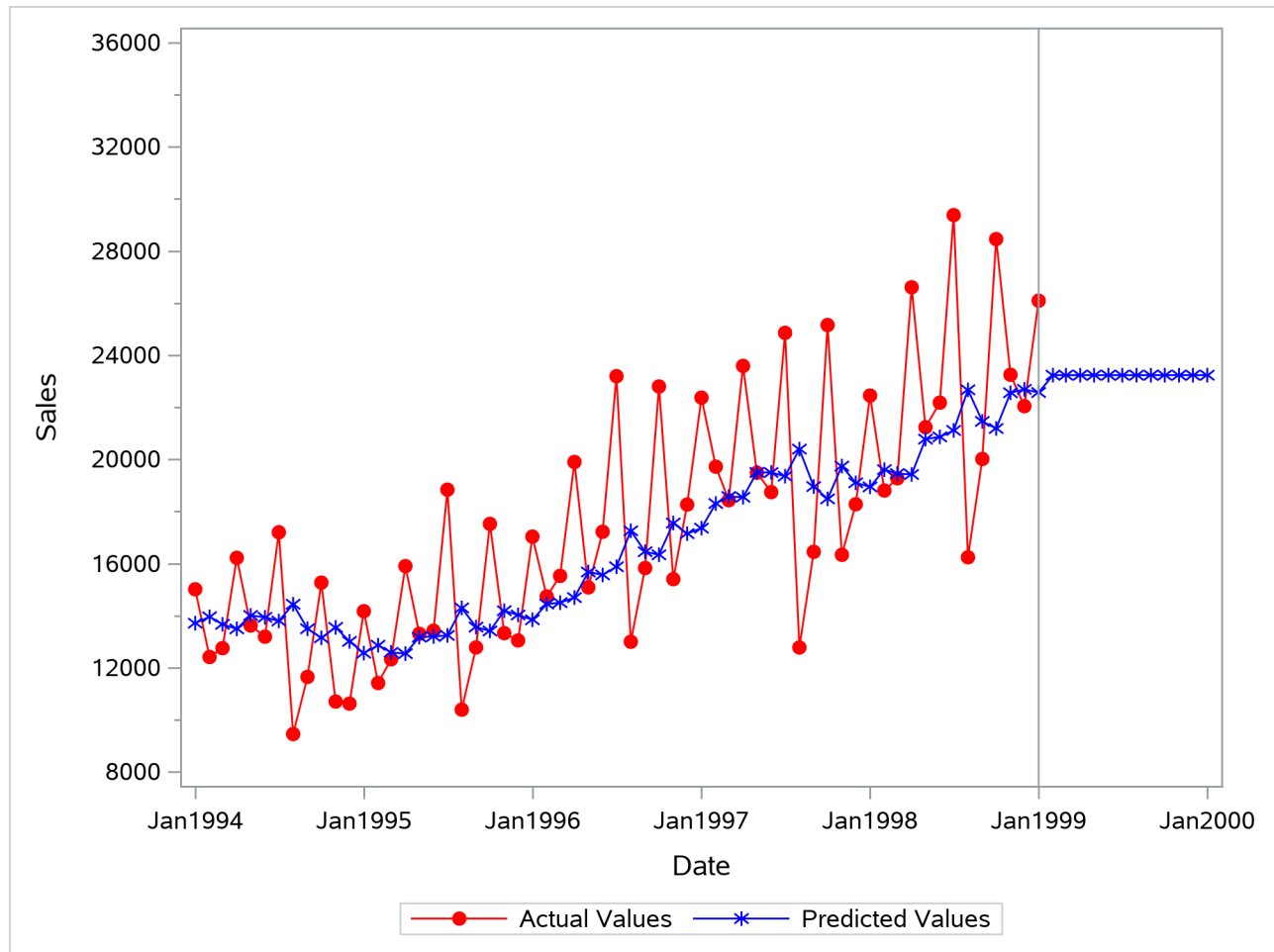
```
%plotActualPredict(mycas.nextyear, date, blouses, '01JAN1999'd,
'01JAN1994'd, '01DEC2000'd, year, 'Date',
8000, 36000, 4000, 'Sales');

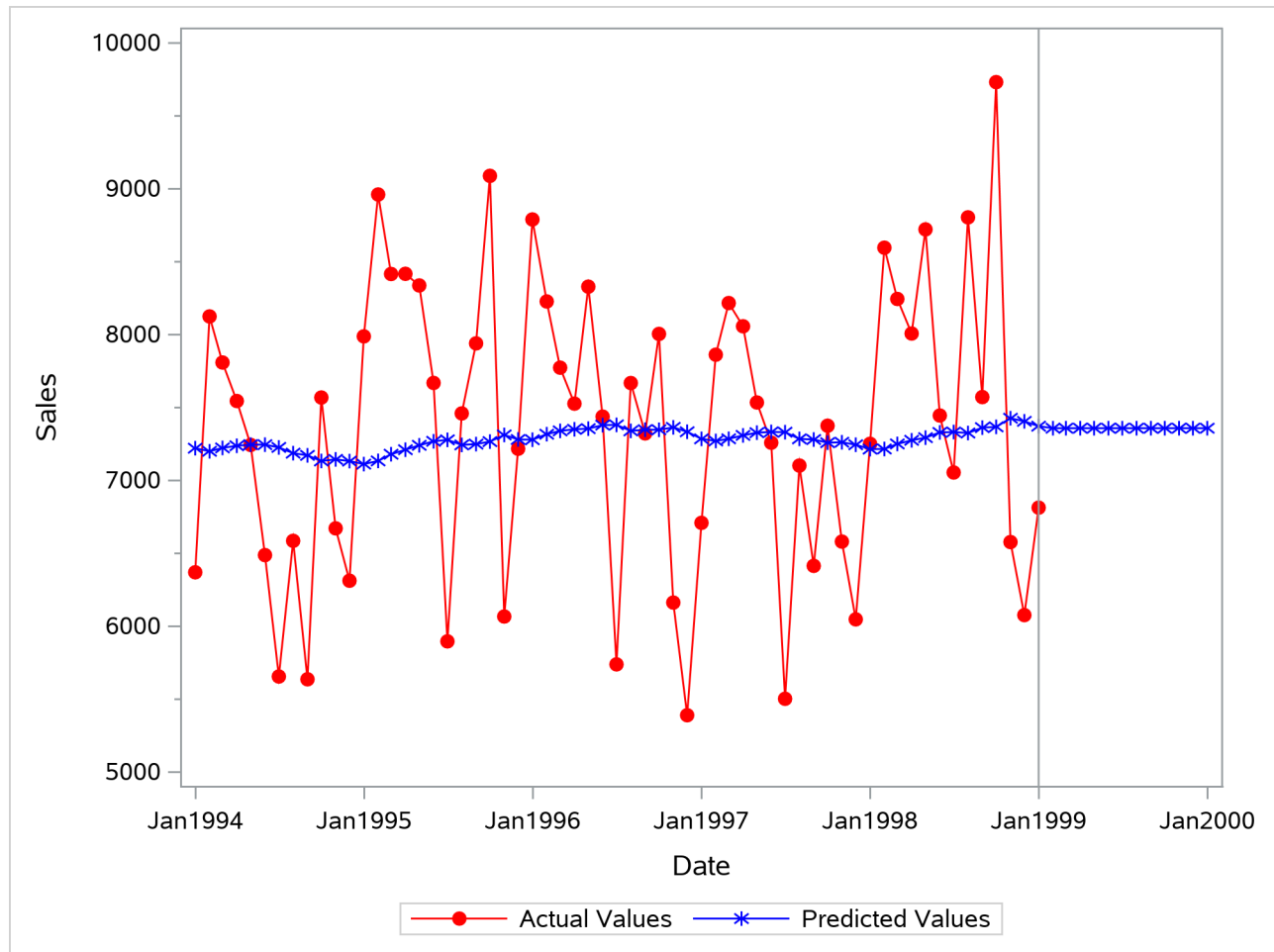
%plotActualPredict(mycas.nextyear, date, laces, '01JAN1999'd,
'01JAN1994'd, '01DEC2000'd, year, 'Date',
5000, 10000, 1000, 'Sales');
```

The plots are shown in [Output 7.1.2](#), [Output 7.1.3](#), and [Output 7.1.4](#). The historical actual values and their predicted values are shown to the left of the reference line, and the forecasts for the next 12 monthly periods are shown to the right. For simple exponential smoothing models, the forecasts are constant. The plots seem to indicate that the fit of the predicted values to the actual values is not good.

Output 7.1.2 Retail Sales Forecast Plots (Belts)



Output 7.1.3 Retail Sales Forecast Plots (Blouses)

Output 7.1.4 Retail Sales Forecast Plots (Laces)

The following statements forecast each of the monthly time series by using the default METHOD=BEST option:

```
proc cesm data=mycas.sales outfor=mycas.nextyear outest=mycas.salesEst;
  id date interval=month;
  forecast _numeric_ / lead=12;
run;
```

The best method for each variable is selected according to the criterion (RMSE, the default value, in this case). The following statements plot the forecasts for the sale of belts, blouses, and laces:

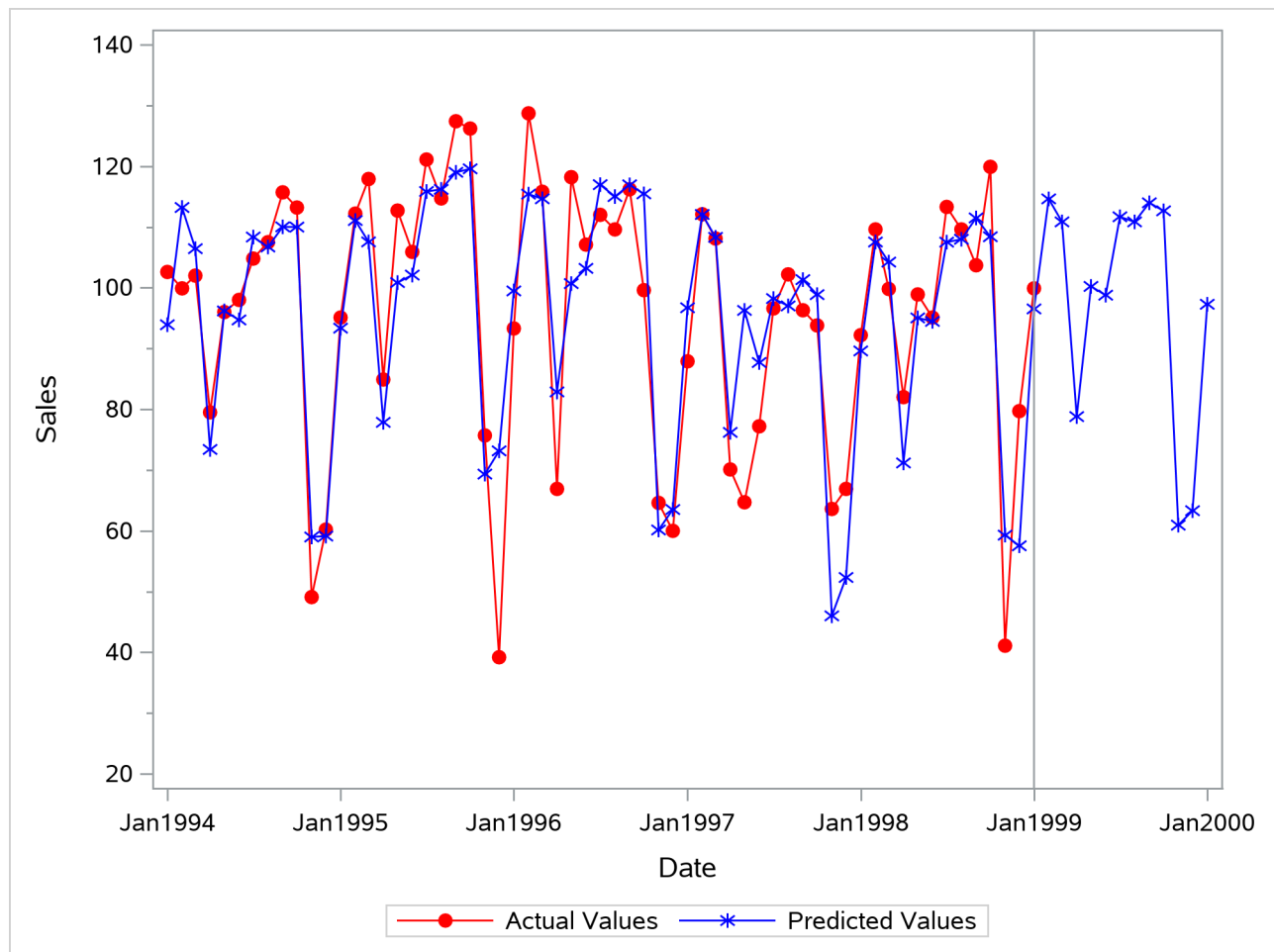
```
%plotActualPredict(mycas.nextyear, date, belts, '01JAN1999'd,
  '01JAN1994'd, '01DEC2000'd, year, 'Date',
  20, 140, 20, 'Sales');

%plotActualPredict(mycas.nextyear, date, blouses, '01JAN1999'd,
  '01JAN1994'd, '01DEC2000'd, year, 'Date',
  8000, 36000, 4000, 'Sales');
```

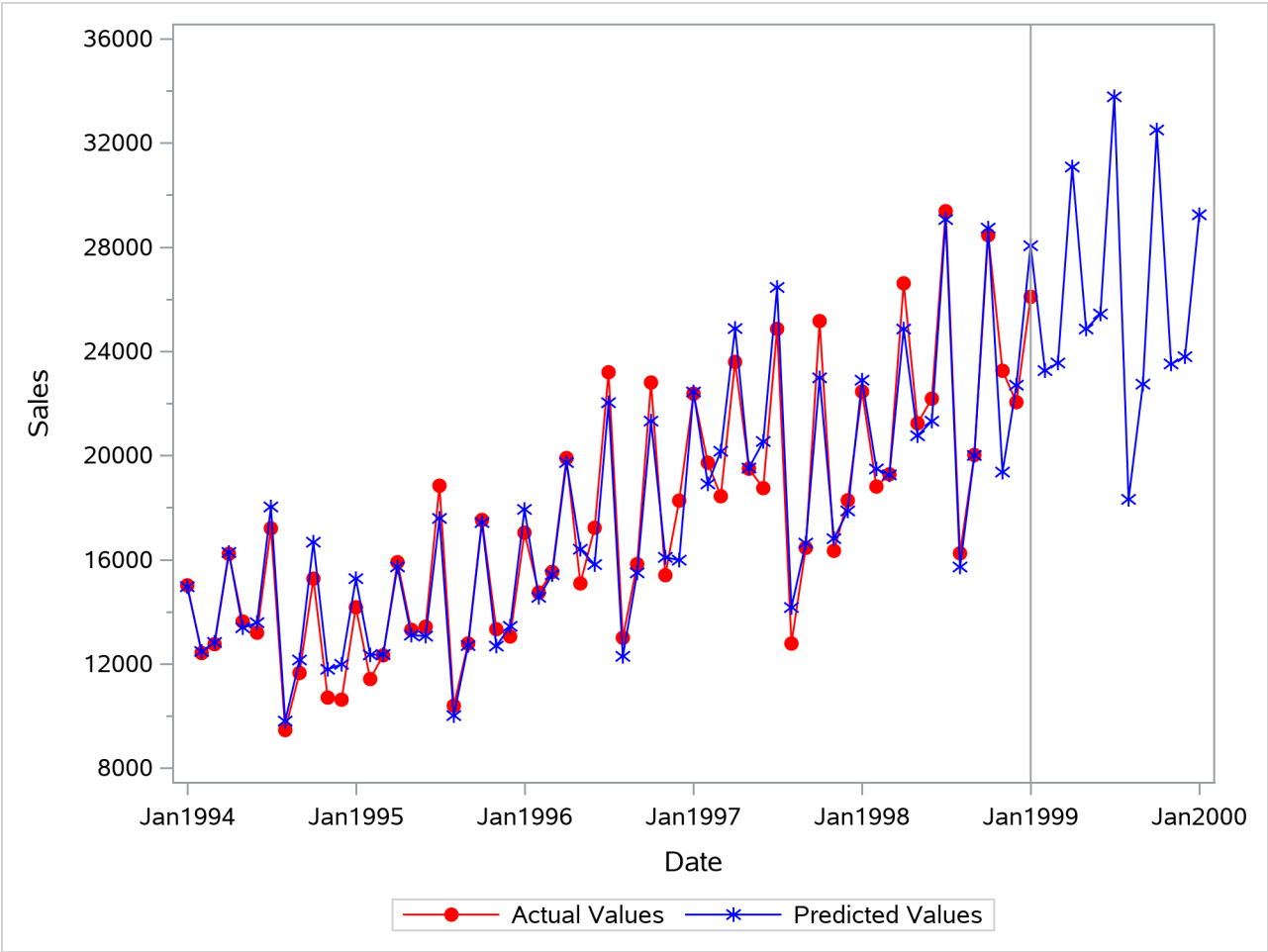
```
%plotActualPredict(mycas.nextyear, date, laces, '01JAN1999'd,
'01JAN1994'd, '01DEC2000'd, year, 'Date',
5000, 10000, 1000, 'Sales');
```

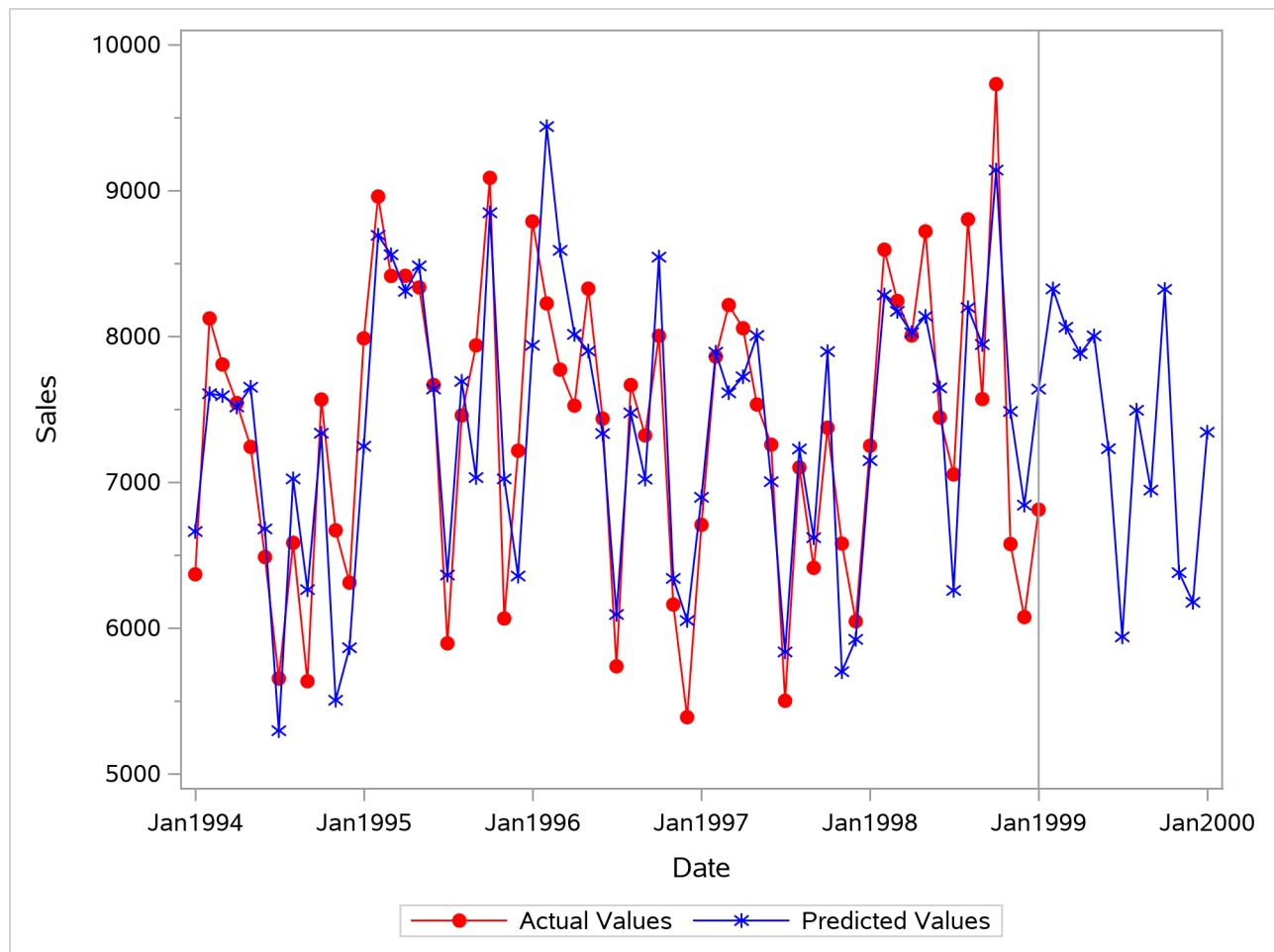
The plots are shown in [Output 7.1.5](#), [Output 7.1.6](#), and [Output 7.1.7](#). The fit of the predicted values to the actual values seems much better, so you might have more confidence in these forecasts.

Output 7.1.5 Retail Sales Forecast Plots (Belts)



Output 7.1.6 Retail Sales Forecast Plots (Blouses)



Output 7.1.7 Retail Sales Forecast Plots (Laces)

The following statements print the methods that are selected for each variable, as shown in [Output 7.1.8](#):

```
data salesEst; set mycas.salesEst; run;
proc sort data=salesEst nodupkey; by _name_ _compmodel_; run;
proc print data=salesEst label noobs;
  var _name_ _compmodel_;
  label _name_='Variable' _compmodel_='Method';
run;
```

Output 7.1.8 Best Method for Each Variable

Variable	Method
belts	SEASONAL
blouses	WINTERS
coats	ADDWINTERS
dressses	SEASONAL
hats	ADDWINTERS
laces	ADDWINTERS
shirts	ADDWINTERS
shoes	ADDWINTERS
socks	ADDWINTERS
ties	SEASONAL

Example 7.2: Forecasting of Transactional Data

This example illustrates how you can use the CESM procedure to forecast transactional data.

The statements in this example assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The following DATA step creates a data table from data that were recorded at several internet websites. The data table mycas.websites contains a variable Time (which represents time) and the variables Engine, Boats, Cars, and Planes, which represent internet website data. Each value of the Time variable is recorded in ascending order, and the values of each of the other variables represent a transactional data series.

```
data mycas.websites(label="Transactional internet data");

  /*- time variable definition -*/
  keep time;
  format time datetime.;
  label time="Time of Web Hit";
  starttime = '12mar2000:00:00:00'dt; /*- Sunday -*/
  seedtime = 1234321;

  /*- cars variable definition -*/
  keep cars;
  format cars best12.;
  label cars="Number of Car Web Hits";
  seedcar = 1234321;

  /*- boats variable definition -*/
  keep boats;
  format boats best12.;
  label boats="Number of Boat Web Hits";
  seedboat = 1234321;

  /*- planes variable definition -*/
  keep planes;
  format planes best12.;
  label planes="Number of Planes Web Hits";
```

```

seedplane = 1234321;

/*- engines variable definition -*/
keep engines;
format engines best12.;
label engines="Number of Engine Web Hits";
seedengine = 1234321;

/*- simulate the data -*/
do day = 1 to 30;
    season = abs(4 - mod(day,7));
    nhits = ceil(10*ranuni(seedtime));
    intv = 24*3600*ranuni(seedtime)/nhits;
    do hits = 1 to nhits;

        /*- randomly generate the next time -*/
        intv = intv + 24*3600*ranuni(seedtime)/nhits;
        intv = int(intv);
        time = intnx( 'DTDAY', starttime, day );
        time = intnx( 'DTSECOND', time, intv );

        /*- randomly generate car data -*/
        cars = 1000 + 600*day + 1000*season
              + 10*rannor(seedcar);
        cars = int(cars);

        /*- randomly generate boats data -*/
        boats = 1000 + 1000*season +
              + 10*rannor(seedboat);
        boats = int(boats);

        /*- randomly generate planes data -*/
        planes = 1000 - 10*day +
              + 10*rannor(seedplane);
        planes = int(planes);

        /*- randomly generate engines data -*/
        engines = 1000 + 1*cars - 2*boats + 4*planes
              + 10*rannor(seedengine);
        engines = int(engines);

        output;
    end;
end;

run;

```

The following CESM procedure statements forecast each of the transactional data series:

```

proc cesm data=mycas.websites outfor=mycas.nextweek;
    id time interval=dtday accumulate=total;
    forecast boats cars planes / lead=7 method=simple;
run;

```

The preceding statements accumulate the data into a daily time series, generate forecasts for the Boats, Cars,

and Planes variables in the input data table mycas.websites for the next week, and store the forecasts in the mycas.nextWeek data table, which is specified in the OUTFOR= option.

The following statements plot the forecasts that are related to the internet data:

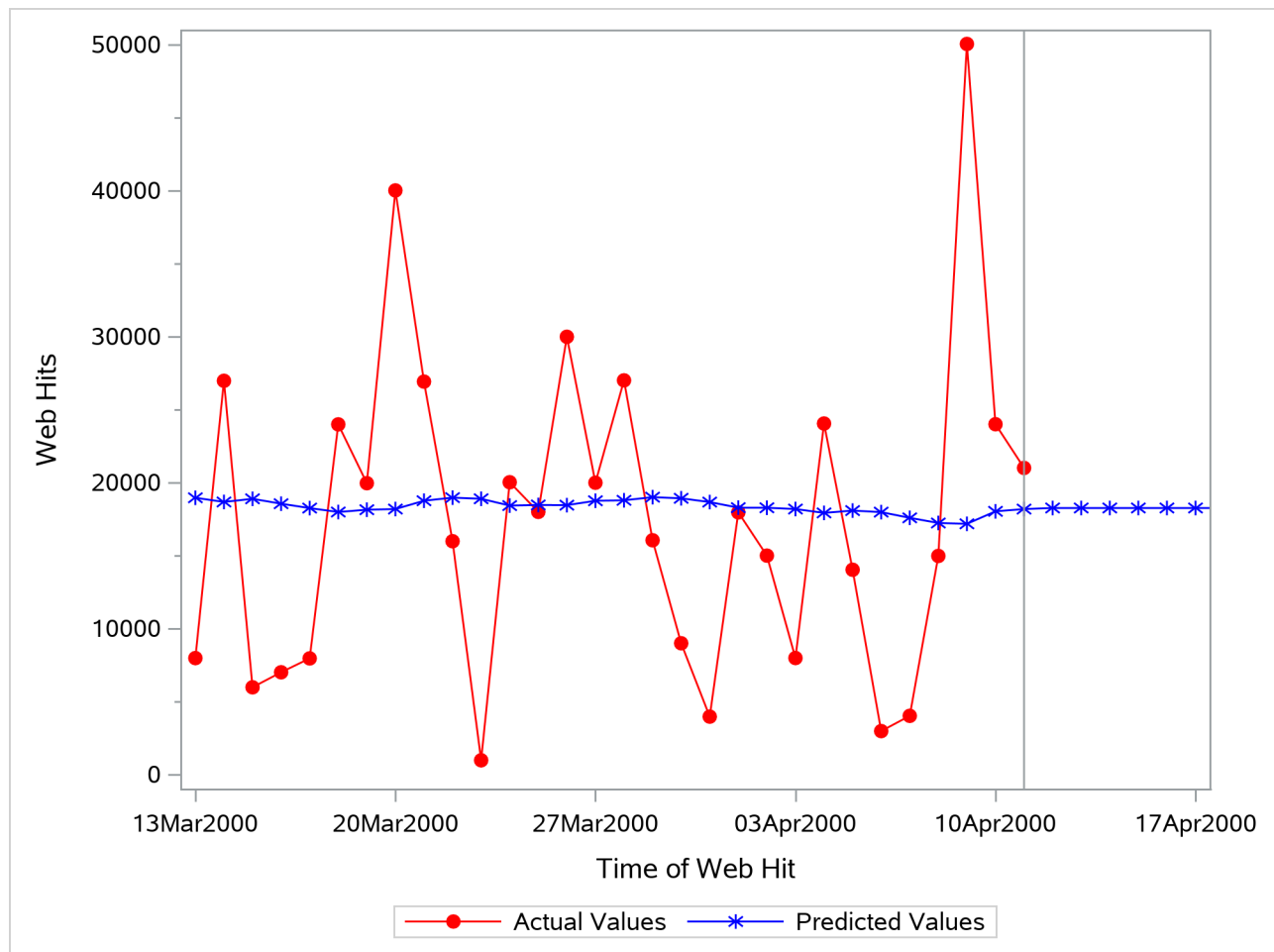
```
%plotActualPredict(mycas.nextweek, time, boats, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 50000, 10000, 'Web Hits');

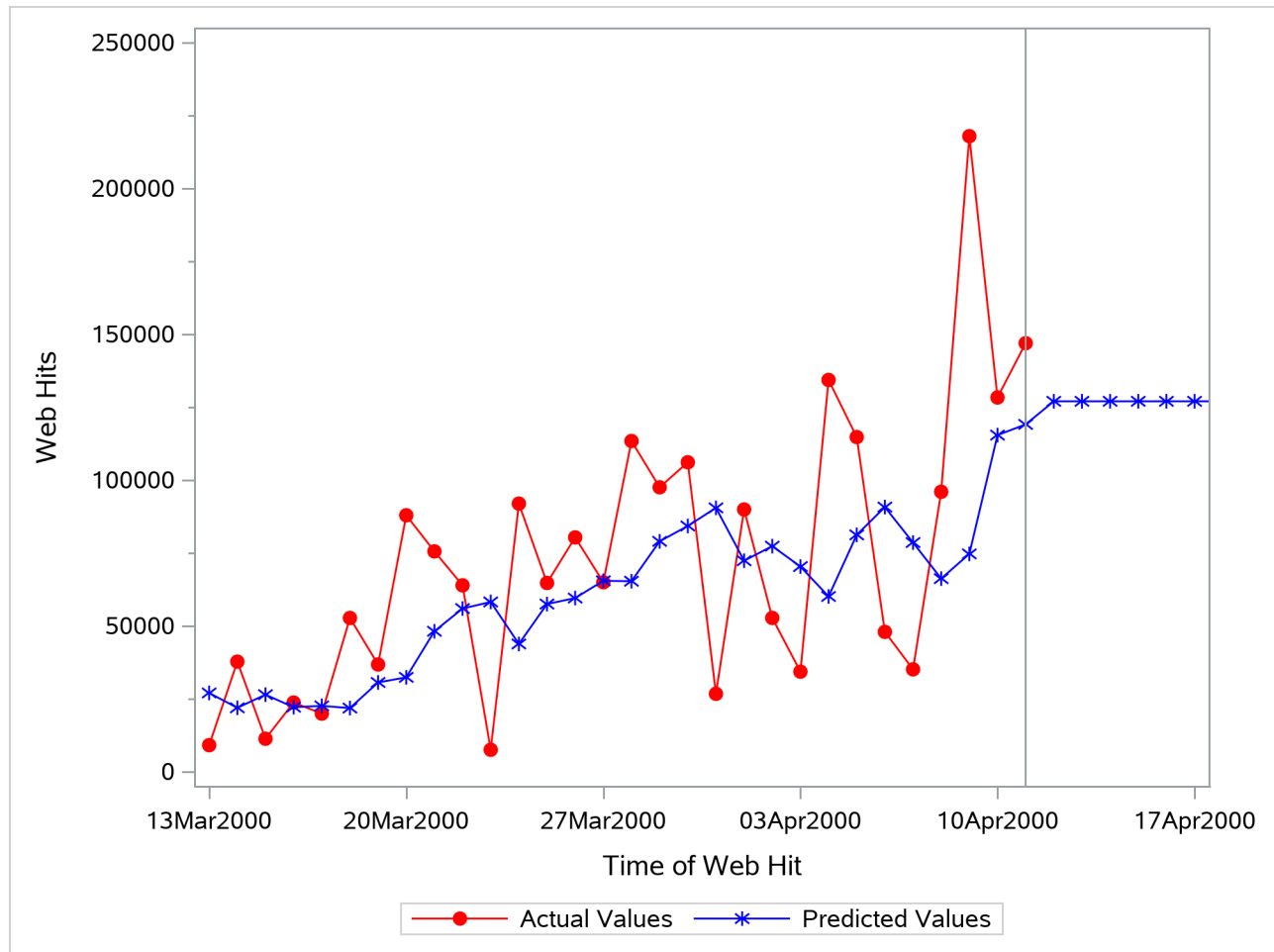
%plotActualPredict(mycas.nextweek, time, cars, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 250000, 50000, 'Web Hits');

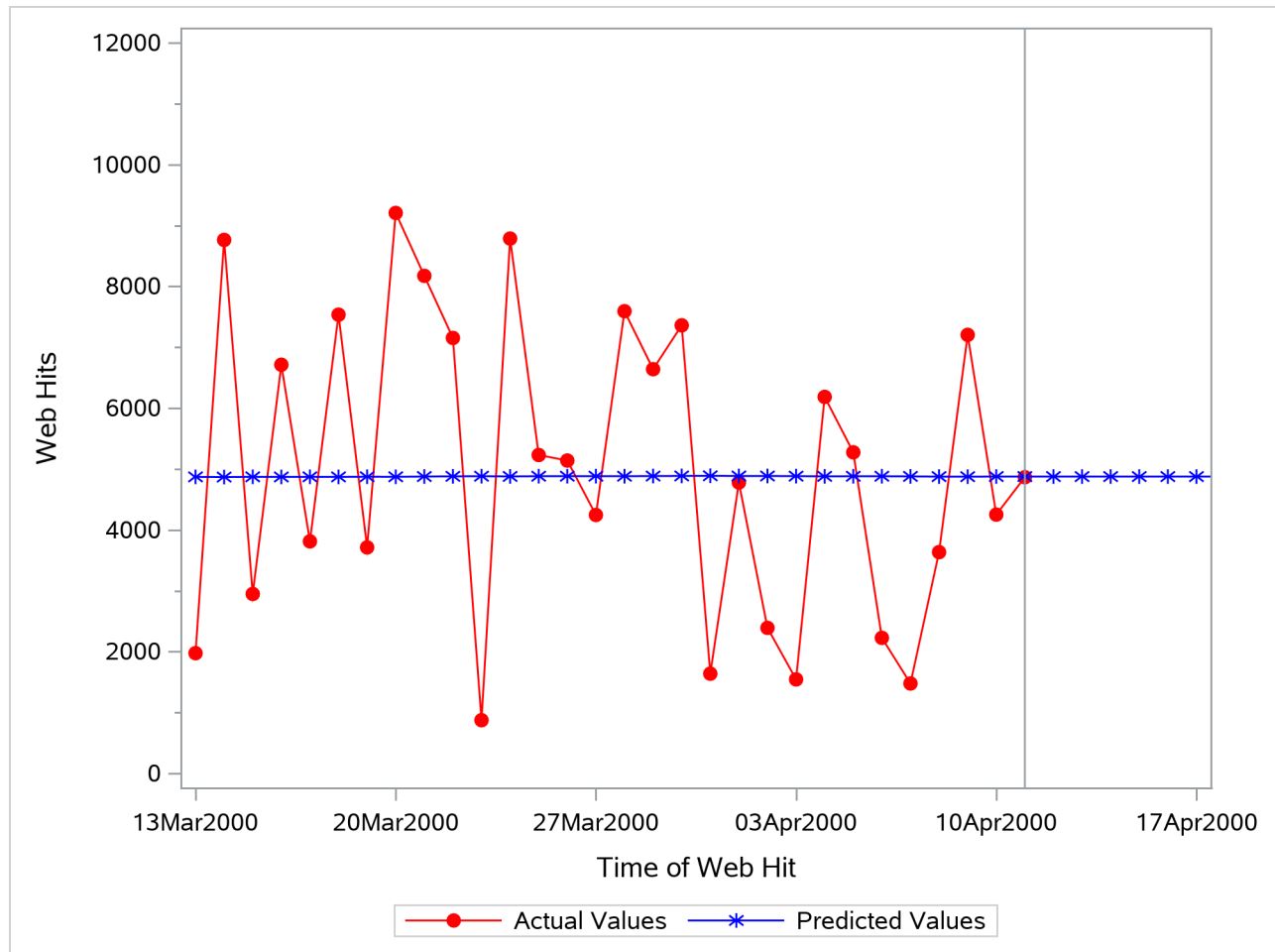
%plotActualPredict(mycas.nextweek, time, planes, '11APR2000:00:00:00'dt,
    '13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
    0, 12000, 2000, 'Web Hits');
```

The plots are shown in [Output 7.2.1](#), [Output 7.2.2](#), and [Output 7.2.3](#). The historical data and their fits are shown to the left of the reference line, and the forecasts for the next seven days are shown to the right.

Output 7.2.1 Internet Data Forecast Plots (Boats)



Output 7.2.2 Internet Data Forecast Plots (Cars)

Output 7.2.3 Internet Data Forecast Plots (Planes)

Example 7.3: Specifying the Forecasting Model

This example illustrates how you can use the CESM procedure to specify different models for different series. Internet data from the previous example are used for this illustration.

The statements in this example assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

This example forecasts the Boats variable by using the seasonal exponential smoothing model (SEASONAL), the Cars variable by using the Winters (multiplicative) model (WINTERS), and the Planes variable by using the log Winters (additive) model (ADDWINTERS). The following CESM procedure statements forecast each of the transactional data series based on these requirements:

```
proc cesm data=mycas.websites outfor=mycas.nextweek;
  id time interval=dtday accumulate=total;
  forecast boats / lead=7 method=seasonal;
  forecast cars / lead=7 method=winters;
  forecast planes / lead=7 method=addwinters transform=log;
run;
```

The following statements plot the forecasts that are related to the internet data:

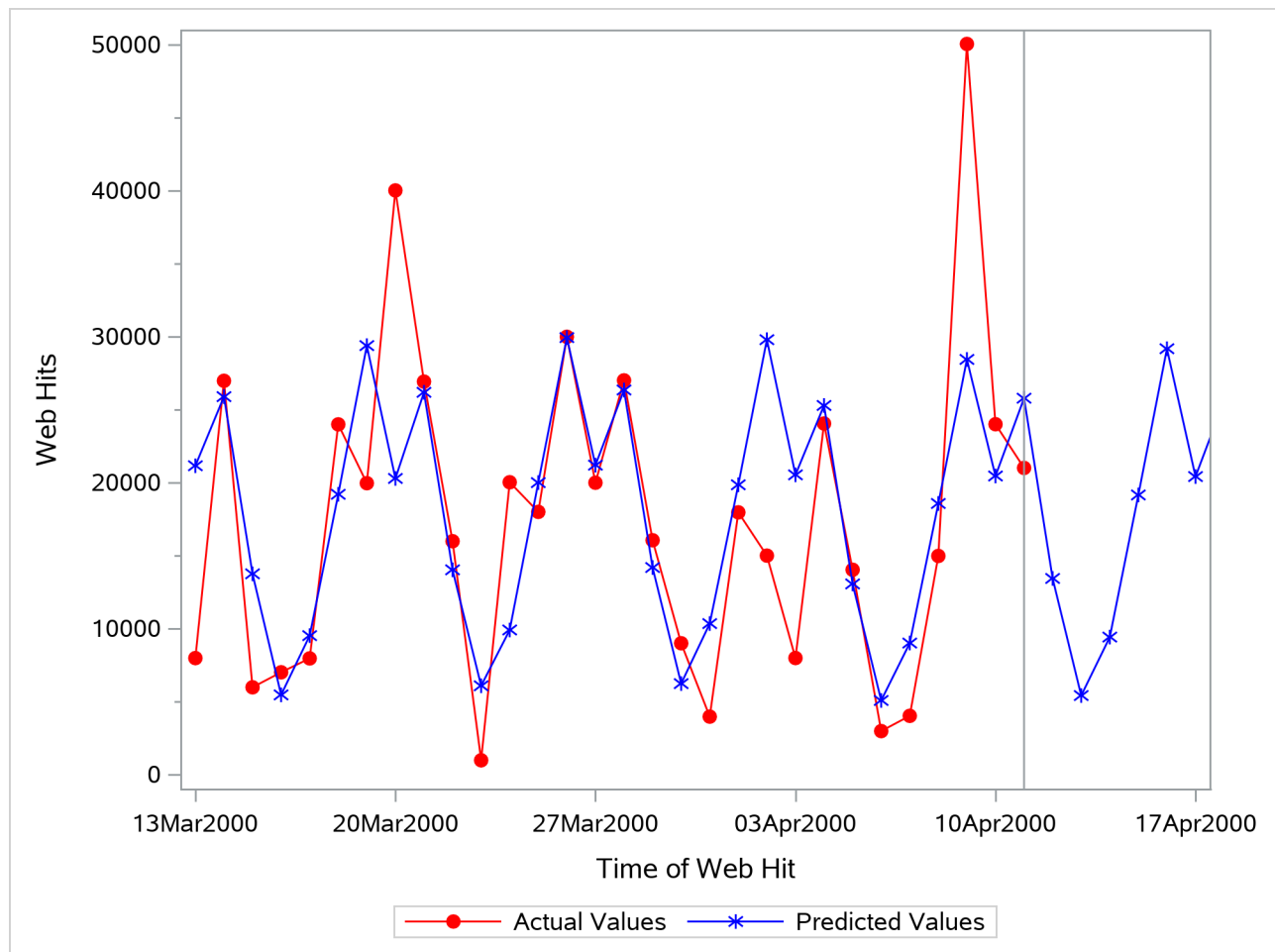
```
%plotActualPredict(mycas.nextweek, time, boats, '11APR2000:00:00:00'dt,
'13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
0, 50000, 10000, 'Web Hits');
```

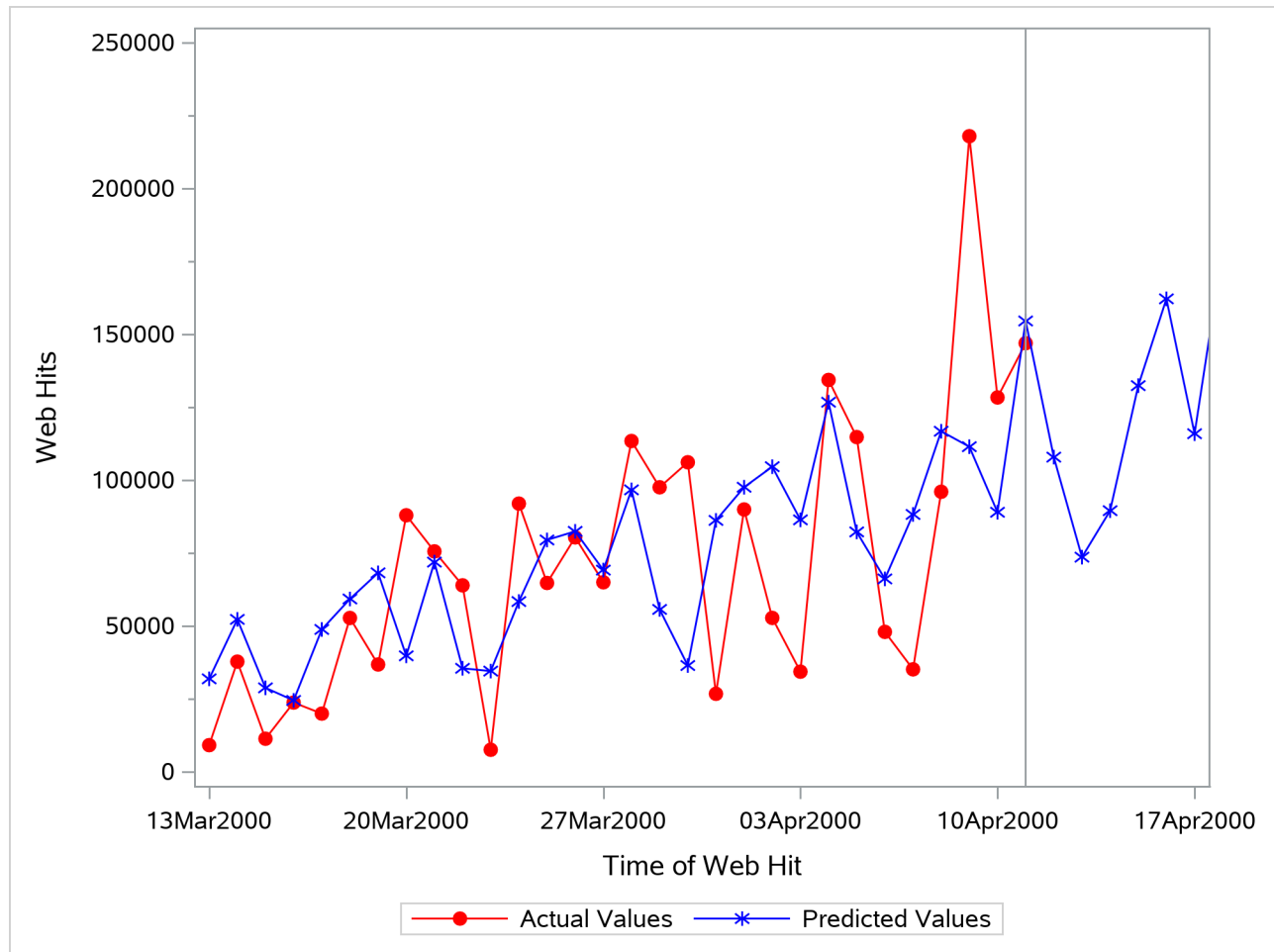
```
%plotActualPredict(mycas.nextweek, time, cars, '11APR2000:00:00:00'dt,
'13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
0, 250000, 50000, 'Web Hits');
```

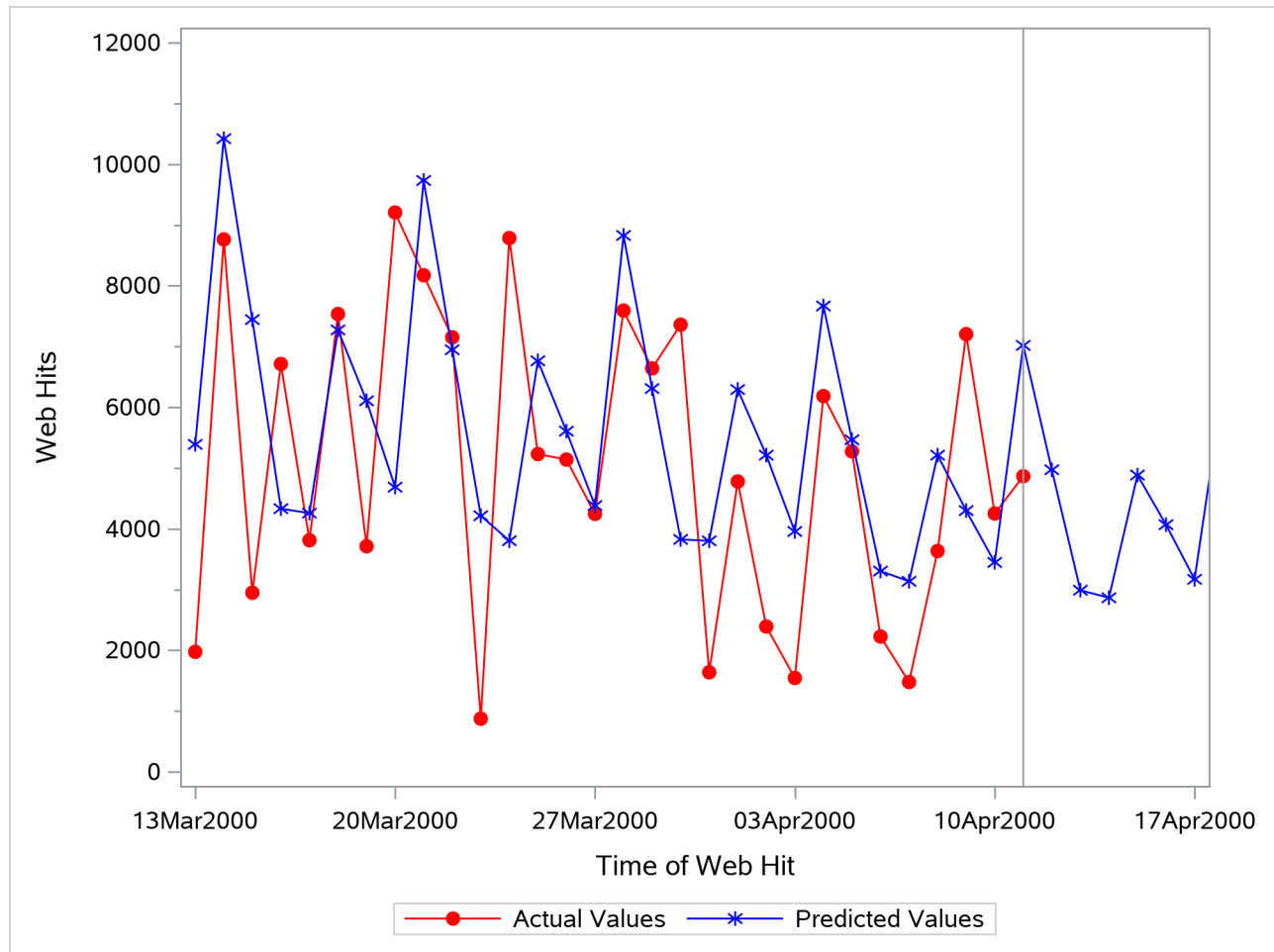
```
%plotActualPredict(mycas.nextweek, time, planes, '11APR2000:00:00:00'dt,
'13MAR2000:00:00:00'dt, '18APR2000:00:00:00'dt, dtweek, 'Time of Web Hit',
0, 12000, 2000, 'Web Hits');
```

The plots are shown in [Output 7.3.1](#), [Output 7.3.2](#), and [Output 7.3.3](#). The historical data and their fits are shown to the left of the reference line, and the forecasts for the next seven days are shown to the right.

Output 7.3.1 Internet Data Forecast Plots (Boats)



Output 7.3.2 Internet Data Forecast Plots (Cars)

Output 7.3.3 Internet Data Forecast Plots (Planes)

Chapter 8

The CNTSELECT Procedure

Contents

Overview: CNTSELECT Procedure	286
PROC CNTSELECT Features	287
PROC CNTSELECT Compared with Other SAS Procedures	288
Using CAS Sessions and CAS Engine Librefs	288
Getting Started: CNTSELECT Procedure	289
Syntax: CNTSELECT Procedure	292
Functional Summary	292
PROC CNTSELECT Statement	295
BOUNDS Statement	297
BY Statement	298
CLASS Statement	298
DISPLAY Statement	298
DISPLAYOUT Statement	299
DISPMODEL Statement	300
EFFECT Statement	301
FREQ Statement	302
INIT Statement	303
MODEL Statement	303
OUTPUT Statement	305
RESTRICT Statement	306
SELECTION Statement	308
TEST Statement	310
VIEWSTORE Statement	311
WEIGHT Statement	312
ZEROMODEL Statement	312
Details: CNTSELECT Procedure	313
Missing Values	313
Poisson Regression	314
Conway-Maxwell-Poisson Regression	315
Negative Binomial Regression	319
Zero-Inflated Count Regression Overview	321
Zero-Inflated Poisson Regression	322
Zero-Inflated Conway-Maxwell-Poisson Regression	323
Zero-Inflated Negative Binomial Regression	325
Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT State- ments	326

Computational Resources	330
Covariance Matrix Types	330
Displayed Output	331
OUTPUT OUT= Data Table	332
ODS Table Names	333
Example: The CNTSELECT Procedure	334
Example 8.1: Zero-Inflated Poisson Model with CLASS Statement	334
References	336

Overview: CNTSELECT Procedure

The CNTSELECT (count regression) procedure analyzes regression models in which the dependent variable takes nonnegative integer or count values. The dependent variable is usually an *event count*, which refers to the number of times an event occurs. For example, an event count might represent the number of ship accidents per year for a given fleet. In count regression, the conditional mean $E(y_i | \mathbf{x}_i)$ of the dependent variable y_i is assumed to be a function of a vector of covariates $\mathbf{x}_i$.

The models that PROC CNTSELECT supports can contain main effects that consist of both continuous and classification variables and interaction effects of these variables. The models can also include constructed effects such as splines. The procedure offers effect-selection methods along with capabilities for customizing the model selection process by using various selection and stopping criteria.

The Poisson (log-linear) regression model is the most basic model that explicitly takes into account the nonnegative integer-valued aspect of the outcome. For this model, the probability of an event count is determined by a Poisson distribution, where the conditional mean of the distribution is a function of a vector of covariates. However, the basic Poisson regression model is limited because it forces the conditional mean of the outcome to equal the conditional variance. This assumption is often violated in real-life data. Negative binomial regression is an extension of Poisson regression in which the conditional variance can exceed the conditional mean. Also, a common characteristic of count data is that the number of zeros in the sample exceeds the number of zeros that are predicted by either the Poisson or negative binomial model. Zero-inflated Poisson (ZIP) and zero-inflated negative binomial (ZINB) models explicitly model the production of zero counts to account for excess zeros and also enable the conditional variance of the outcome to differ from the conditional mean.

In zero-inflated models, additional zeros occur with probability φ_i , which is determined by a separate model, $\varphi_i = F(\mathbf{z}_i' \boldsymbol{\gamma})$, where F is the normal or logistic distribution function that results in a probit or logistic model and $\mathbf{z}_i$ is a set of covariates.

PROC CNTSELECT Features

The CNTSELECT procedure estimates the parameters of a count regression model by using maximum likelihood techniques.

The CNTSELECT procedure supports the following models for count data:

- Poisson regression
- Conway-Maxwell-Poisson regression
- negative binomial regression with quadratic and linear variance functions (Cameron and Trivedi 1986)
- zero-inflated Poisson (ZIP) model (Lambert 1992)
- zero-inflated Conway-Maxwell-Poisson (ZICMP) model
- zero-inflated negative binomial (ZINB) model
- fixed-effects and random-effects Poisson models for panel data
- fixed-effects and random-effects negative binomial models for panel data

The following list summarizes some basic features of the CNTSELECT procedure:

- has model-building syntax that uses **CLASS** and effect-based **MODEL** statements familiar from SAS/Econometrics analytic procedures
- performs maximum likelihood estimation
- supports multiple link functions
- uses the **WEIGHT** statement for weighted analysis
- uses the **FREQ** statement for grouped analysis
- uses the **OUTPUT** statement to produce a data table that contains predicted probabilities and other observationwise statistics
- supports effect-selection methods

PROC CNTSELECT Compared with Other SAS Procedures

The CNTSELECT procedure provides count-data modeling functionality comparable to that of the HPCOUNTREG and COUNTREG procedures in SAS/ETS software.

PROC CNTSELECT Compared with the HPCOUNTREG Procedure

The functionality of the CNTSELECT procedure closely resembles that of the HPCOUNTREG procedure, which is a high-performance procedure. The CNTSELECT procedure is the next generation of the HPCOUNTREG procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CNTSELECT procedure provides all the functionality of the HPCOUNTREG procedure.

PROC CNTSELECT Compared with the COUNTREG Procedure

The CNTSELECT procedure is specifically designed to operate on SAS Viya and performs computations in multiple threads on multiple machines. Although the COUNTREG procedure runs in multiple threads, it executes only on a single machine.

This release of the CNTSELECT procedure contains some, but not all, of the functionality present in the COUNTREG procedure. The COUNTREG procedure provides the following features, which are not present in the CNTSELECT procedure:

- spatial effect models
- Bayesian analysis
- scoring by using an item store

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CNTSELECT Procedure

Except for its ability to operate in the high-performance distributed environment, the CNTSELECT procedure is similar in use to other regression model procedures in the SAS System. For example, the following statements are used to estimate a Poisson regression model:

```
proc cntselect data=mycas.one ;
  model y = x / dist=poisson ;
run;
```

The response variable `y` is numeric and has nonnegative integer values.

This section illustrates two simple examples that use PROC CNTSELECT. The data are taken from Long (1997). This study examines how factors such as gender (`fem`), marital status (`mar`), number of young children (`kid5`), prestige of the graduate program (`phd`), and number of articles published by a scientist’s mentor (`ment`) affect the number of articles (`art`) published by the scientist.

The first 10 observations are shown in [Figure 8.1](#).

Figure 8.1 Article Count Data

Obs	art	fem	mar	kid5	phd	ment
1	3	0	1	2	1.38000	8.0000
2	1	0	1	1	3.59000	19.0000
3	0	0	1	1	2.12000	10.0000
4	3	0	1	1	1.80000	4.0000
5	1	0	1	1	3.41000	10.0000
6	2	0	1	0	2.10000	2.0000
7	0	0	1	1	2.26000	5.0000
8	6	0	1	1	3.85000	16.0000
9	2	0	0	0	2.26000	2.0000
10	1	0	1	2	4.29000	10.0000

The following SAS statements estimate the Poisson regression model.

```
/*-- Poisson Regression --*/
proc cntselect data=mycas.long97data;
    model art = fem mar kid5 phd ment / dist=poisson method=quanew;
run;
```

The “Model Fit Summary” table that is shown in [Figure 8.2](#) lists several details about the model. By default, the CNTSELECT procedure uses the Newton-Raphson optimization technique. The maximum log-likelihood value is shown, in addition to two information measures—Akaike’s information criterion (AIC) and Schwarz’s Bayesian information criterion (SBC)—which can be used to compare competing Poisson models. Smaller values of these criteria indicate better models.

Figure 8.2 Estimation Summary Table for a Poisson Regression

The CNTSELECT Procedure	
Model Fit Summary	
Dependent Variable	art
Number of Observations	915
Data Set	LONG97DATA
Model	Poisson
Log Likelihood	-1651.06
Maximum Absolute Gradient	0.001295
Number of Iterations	11
Optimization Method	Quasi-Newton
AIC	3314.113
SBC	3343.026
Covariance Estimation	Hessian

[Figure 8.3](#) shows the parameter estimates of the model and their standard errors. All covariates are significant predictors of the number of articles, except for the prestige of the program (phd), which has a *p*-value of 0.6271.

Figure 8.3 Parameter Estimates of Poisson Regression

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.304617	0.102982	2.96	0.0031
fem	1	-0.224595	0.054614	-4.11	<.0001
mar	1	0.155243	0.061375	2.53	0.0114
kid5	1	-0.184882	0.040127	-4.61	<.0001
phd	1	0.012823	0.026397	0.49	0.6271
ment	1	0.025543	0.002006	12.73	<.0001

To allow for variance greater than the mean, you can fit the negative binomial model instead of the Poisson model by specifying the DIST=NEGBIN option, as shown in the following statements. Whereas the Poisson model requires that the conditional mean and conditional variance be equal, the negative binomial model allows for overdispersion, in which the conditional variance can exceed the conditional mean.

```

/*-- Negative Binomial Regression --*/
proc cntselect data=mycas.long97data;
  model art = fem mar kid5 phd ment / dist=negbin(p=2) method=quanew;
run;

```

Figure 8.4 shows the fit summary and Figure 8.5 shows the parameter estimates.

Figure 8.4 Estimation Summary Table for a Negative Binomial Regression

The CNTSELECT Procedure

Model Fit Summary	
Dependent Variable	art
Number of Observations	915
Data Set	LONG97DATA
Model	NegBin(p=2)
Log Likelihood	-1560.96
Maximum Absolute Gradient	0.070336
Number of Iterations	12
Optimization Method	Quasi-Newton
AIC	3135.917
SBC	3169.649
Covariance Estimation	Hessian

Figure 8.5 Parameter Estimates of Negative Binomial Regression

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.256154	0.138560	1.85	0.0645
fem	1	-0.216404	0.072672	-2.98	0.0029
mar	1	0.150485	0.082106	1.83	0.0668
kid5	1	-0.176423	0.053060	-3.32	0.0009
phd	1	0.015266	0.036040	0.42	0.6719
ment	1	0.029083	0.003470	8.38	<.0001
_Alpha	1	0.441618	0.052967	8.34	<.0001

The parameter estimate for _Alpha of 0.4416 is an estimate of the dispersion parameter in the negative binomial distribution. A t test for the hypothesis $H_0 : \alpha = 0$ is provided. It is highly significant, indicating overdispersion ($p < 0.0001$).

The null hypothesis $H_0 : \alpha = 0$ can be also tested against the alternative $\alpha > 0$ by using the likelihood ratio test, as described by Cameron and Trivedi (1998, pp. 45, 77–78). The likelihood ratio test statistic is equal to $-2(\mathcal{L}_P - \mathcal{L}_{NB}) = -2(-1651 + 1561) = 180$, which is highly significant, providing strong evidence of overdispersion.

Syntax: CNTSELECT Procedure

The following statements are available in the CNTSELECT procedure. Items within angle brackets (< >) or square brackets ([]) are optional.

```

PROC CNTSELECT <options> ;
  BOUNDS bound1 [ , bound2 ... ] ;
  BY variables ;
  CLASS variable <(options)> ... <variable <(options)>> </ global-options> ;
  DISPLAY <table-list> </ options> ;
  DISPLAYOUT table-spec-list </ options> ;
  DISPMODEL dependent variable ~ <dispersion-related regressors> ;
  EFFECT name=effect-type(variables </ options>) ;
  FREQ freq-variable ;
  INIT initialization1 < , initialization2 ...> ;
  MODEL dependent-variable = regressors </ options> ;
  OUTPUT <output-options> ;
  RESTRICT restriction1 [ , restriction2 ... ] ;
  SELECTION < METHOD=method <(method-options)>> <options> ;
  TEST equation1 < , equation2 ...> </ test-options> ;
  VIEWSTORE / options ;
  WEIGHT variable </ option> ;
  ZEROMODEL dependent-variable ~ zero-inflated-regressors </ options> ;

```

There can be only one MODEL statement. Any ZEROMODEL, DISPMODEL, OUTPUT, INIT, BOUNDS, RESTRICT, or TEST statement, if used, must appear after the MODEL statement. There can be no more than one ZEROMODEL statement and no more than one DISPMODEL statement. Multiple INIT, BOUNDS, RESTRICT, and TEST statements are allowed. If a FREQ or WEIGHT statement is specified more than once, the variable specified in the first instance is used.

Functional Summary

Table 8.1 summarizes the statements and options used with the CNTSELECT procedure.

Table 8.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CNTSELECT	DATA=
Specifies the identification variable for panel data analysis	PROC CNTSELECT	GROUPID=
Writes estimates to an output data table	OUTPUT	OUT=
Specifies BY-group processing	BY	
Specifies the classification variables	CLASS	
Specifies an optional frequency variable	FREQ	
Specifies an optional weight variable	WEIGHT	

Table 8.1 *continued*

Description	Statement	Option
Printing Control Options		
Prints the correlation matrix of the estimates	PROC CNTSELECT	CORRB
Prints the covariance matrix of the estimates	PROC CNTSELECT	COVB
Suppresses the normal printed output	PROC CNTSELECT	NOPRINT
Requests all printing options	PROC CNTSELECT	PRINTALL
Prints timing information	PROC CNTSELECT	PRINTTIMING
Prints the names used internally for the parameters	PROC CNTSELECT	PRINTINTERNALNAMES
Options to Control the Optimization Process		
Selects the iterative minimization method to use	PROC CNTSELECT	METHOD=
Specifies maximum number of iterations allowed	PROC CNTSELECT	MAXITER=
Specifies maximum number of function calls	PROC CNTSELECT	MAXFUNC=
Specifies the upper limit of CPU time in seconds	PROC CNTSELECT	MAXTIME=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the dispersion variables	DISPMODEL	
Specifies the type of model	PROC CNTSELECT	DIST=
Specifies the type of covariance matrix	PROC CNTSELECT	COVEST=
Specifies the type of error components model for panel data	MODEL	ERRORCOMP=
Suppresses the intercept parameter	MODEL	NOINT
Specifies the offset variable	MODEL	OFFSET=
Specifies the parameterization for the Conway-Maxwell-Poisson (CMP) model	MODEL	PARAMETER=
Specifies the zero-inflated offset variable	ZEROMODEL	OFFSET=
Specifies the zero-inflated link function	ZEROMODEL	LINK=
Specifies the constructed regression effects	EFFECT	
Regression Effect-Selection Options		
Specifies the selection method	SELECTION	METHOD=
Specifies how to apply the STOP= criterion	SELECTION	STOPHORIZON=
Regression Effect-Selection Method Options		
Specifies a criterion for choosing the best model at each step	SELECTION	CHOOSE=
Specifies that the competitive form of the stepwise selection method be used	SELECTION	COMPETITIVE
Specifies the maximum number of effects in the model	SELECTION	MAXEFFECTS=
Specifies the maximum number of selection steps	SELECTION	MAXSTEPS=

Table 8.1 *continued*

Description	Statement	Option
Specifies the minimum number of effects in the model	SELECTION	MINEFFECTS=
Specifies a criterion to decide which effects enter or leave the model at each step	SELECTION	SELECT=
Specifies a criterion for stopping the selection process	SELECTION	STOP=
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	
Specifies the output item store to preserve the properties of the model and the results of fitting the model	PROC CNTSELECT	STORE=
Outputs SAS variables to the output data table	OUTPUT	COPYVAR=
Outputs the estimates of dispersion for the CMP model	OUTPUT	DISPERSION
Outputs the estimates of $G\Delta = \mathbf{g}'_i \boldsymbol{\delta}$ for the CMP model	OUTPUT	GDELTA=
Outputs the estimates of λ for the CMP model	OUTPUT	LAMBDA=
Outputs the estimates of ν for the CMP model	OUTPUT	NU=
Outputs the estimates of μ for the CMP model	OUTPUT	MU=
Outputs the estimates of mode for the CMP model	OUTPUT	MODE=
Outputs the probability that the response variable will take the current value	OUTPUT	PROB=
Outputs probabilities for particular response values	OUTPUT	PROBCOUNT()
Outputs expected value of response variable	OUTPUT	PRED=
Outputs the estimates of variance for the CMP model	OUTPUT	VARIANCE=
Outputs estimates of $\mathbf{XBeta} = \mathbf{x}'_i \boldsymbol{\beta}$	OUTPUT	XBETA=
Outputs estimates of $\mathbf{ZGamma} = \mathbf{z}'_i \boldsymbol{\gamma}$	OUTPUT	ZGAMMA=
Outputs probability of a zero value as a result of the zero-generating process	OUTPUT	PROBZERO=

PROC CNTSELECT Statement

PROC CNTSELECT <options> ;

The following *options* can be used in the PROC CNTSELECT statement.

Input Data Table Options

DATA=*CAS-libref.data-table*

names the input data table for PROC CNTSELECT to use. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 288.

data-table specifies the name of the input data table.

GROUPID=*variable*

specifies an identification variable when a panel data model is estimated. The identification variable is used as a cross-sectional ID variable.

Item Store Option

STORE=*SAS-item-store*

specifies the item store that preserves the properties of the model and the fit results. No information about the model is preserved unless you specify this option.

Printing Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

PRINTTIMING

prints a timing report.

PRINTINTERNALNAMES

prints internal names assigned to parameters.

Estimation Control Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

COVEST=HESSIAN | OP | QML

specifies the type of covariance matrix for the parameter estimates.

The default is COVEST=HESSIAN. You can specify the following values:

HESSIAN	specifies the covariance from the Hessian matrix.
OP	specifies the covariance from the outer product matrix.
QML	specifies the covariance from the outer product and Hessian matrices.

Optimization Control Options

PROC CNTSELECT uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options* in either the PROC CNTSELECT statement or the MODEL statement.

MAXFUNC=*i***MAXFU=*i***

specifies the maximum number of function calls in the optimization process. The default is 1,000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number of calls that are specified by this option.

MAXITER=*i***MAXIT=*i***

specifies the maximum number of iterations in the optimization process. The default is 200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation of your computer. The time that is specified by this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time needed to finish the iteration and the time needed to generate the output of the results.

METHOD=*value*

specifies the iterative minimization method to use. The default is METHOD=NEWRAP. You can specify the following *values*:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.

NEWRAP	specifies the Newton-Raphson method (this is the default).
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NRRIDG	specifies the Newton-Raphson Ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

BOUNDS Statement

BOUNDS *bound1* [, *bound2* ...] ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item [*operator item* [*operator item* ...]]

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as '*', '-', '(', or ')'), then you must use the internal name of the parameter in order to refer to that parameter in the BOUNDS statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 326.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. For more information, see the section “[RESTRICT Statement](#)” on page 306.

The following BOUNDS statement illustrates the use of parameter lists to specify boundary constraints. It constrains the estimates of the parameter for z to be negative, the parameters for x1 through x10 to be between 0 and 1, and the parameter for x1 in the zero-inflation model to be less than 1.

```
bounds z < 0,
       0 < x1-x10 < 1,
       Inf_x1 < 1;
```

BY Statement

BY *variables* ;

A BY statement can be used with PROC CNTSELECT to obtain separate analyses on observations in groups that are defined by the BY *variables*.

CLASS Statement

CLASS *variable* <(options)>... <*variable* <(options)>> </global-options> ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. Table 8.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 14 in Chapter 3, “Shared Concepts.”

Table 8.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY <*table-list*> </options> ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 333. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* </ *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the CAS output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 333. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

DISPMODEL Statement

DISPMODEL *dependent-variable* ~ < dispersion-related-regressors > ;

The DISPMODEL statement specifies the *dispersion-related-regressors* that are used to model dispersion. This statement is ignored unless you specify either DIST=CMPOISSON or DIST=ZICMPOISSON in the MODEL statement. The *dependent-variable* in the DISPMODEL statement must be the same as the *dependent-variable* in the MODEL statement.

The *dependent-variable* that appears in the DISPMODEL statement is directly used to model dispersion. Each of the q variables to the right of the tilde (~) has a parameter to be estimated in the regression. For example, let $\mathbf{g}_i'$ be the i th observation's $1 \times (q + 1)$ vector of values of the q dispersion explanatory variables (q_0 is set to 1 for the intercept term). Then the dispersion is a function of $\mathbf{g}_i' \boldsymbol{\delta}$, where $\boldsymbol{\delta}$ is the $(q + 1) \times 1$ vector of parameters to be estimated, the dispersion model intercept is δ_0 , and the coefficients for the q dispersion covariates are $\delta_1, \dots, \delta_q$. If you specify DISP=CMPOISSON in the MODEL statement but do not include a DISPMODEL statement, then only the intercept term δ_0 is estimated. The “Parameter Estimates” table in the displayed output shows the estimates for the dispersion intercept and dispersion explanatory variables; they are labeled with the prefix “Disp_”. For example, the dispersion intercept is labeled “Disp_Intercept”. If you specify Age (a variable in your data table) as a dispersion explanatory variable, then the “Parameter Estimates” table labels the corresponding parameter estimate “Disp_Age”. The following statements fit a Conway-Maxwell-Poisson model by using the regressors SEX, ILLNESS, and INCOME and by using AGE as a dispersion-related regressor:

```
proc cntselect data=mycas.docvisit;
  model doctorvisits=sex illness income / dist=cm poisson;
  dispmodel doctorvisits ~ age;
run;
```

EFFECT Statement

EFFECT *name=effect-type (variables < / options>)* ;

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are referred to as *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 52 in Chapter 3, “Shared Concepts.”

You can specify the following *effect-types*:

COLLECTION	specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference.
MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables.

Table 8.3 summarizes the *options* available in the EFFECT statement.

Table 8.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 8.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

For more information about the syntax of these *effect-types* and how columns of constructed effects are computed, see the section “**EFFECT Statement**” on page 20 in Chapter 3, “**Shared Concepts**.”

FREQ Statement

FREQ *freq-variable* ;

The FREQ statement identifies a variable (*freq-variable*) that contains the frequency of occurrence of each observation. PROC CNTSELECT treats each observation as if it appears n times, where n is the value of *freq-variable* for the observation. If the value for the observation is not an integer, it is truncated to an integer. If the value is less than 1 or missing, the observation is not used in the model fitting. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

INIT Statement

INIT *initialization1* < , *initialization2* ... > ;

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter <=> *number*

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as '*', '-', '(', or ')'), then you must use the internal name of the parameter in order to refer to that parameter in the INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 326.

MODEL Statement

MODEL *dependent-variable* = *regressors* </ *options*> ;

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model. The dependent count variable should take only nonnegative integer values from the input data table. PROC CNTSELECT rounds any positive noninteger count value to the nearest integer. PROC CNTSELECT discards any observation that has a negative count.

Only one MODEL statement can be specified. You can specify the following *options* in the MODEL statement after a slash (/):

DIST=*value*

specifies a type of model to be analyzed. You can specify the following *values*:

POISSON P	specifies the Poisson regression model.
CMPOISSON C CMP	specifies a Conway-Maxwell-Poisson regression model.
NEGBIN(P=1)	specifies the negative binomial regression model that uses a linear variance function.
NEGBIN(P=2) NEGBIN	specifies the negative binomial regression model that uses a quadratic variance function.
ZIPOISSON ZIP	specifies zero-inflated Poisson regression.
ZICMPOISSON ZICMP	specifies a zero-inflated Conway-Maxwell-Poisson regression. The ZERO-MODEL statement must be specified when this model type is specified.
ZINEGBIN ZINB	specifies zero-inflated negative binomial regression.

You can also specify the DIST option in the CNTSELECT statement.

ERRORCOMP=FIXED | RANDOM

specifies a type of conditional panel model to be analyzed. You can specify the following model types:

FIXED	specifies a fixed-effect error component regression model.
RANDOM	specifies a random-effect error component regression model.

NOINT

suppresses the intercept parameter.

OFFSET=offset-variable

specifies a variable in the input data table to be used as an offset variable. The *offset-variable* is used to allow the observational units to vary across observations. For example, when the number of shipping accidents could be measured across different time periods or the number of students who participate in an activity could be reported across different class sizes, the observational units need to be adjusted to a common denominator by using the offset variable. The offset variable appears as a covariate in the model with its parameter restricted to 1. The offset variable cannot be the response variable, the zero-inflation offset variable (if any), or any of the explanatory variables. The “Model Fit Summary” table gives the name of the data table variable that is used as the offset variable; it is labeled “Offset.”

PARAMETER=MU | LAMBDA

specifies the parameterization for the Conway-Maxwell-Poisson model. The following parameterizations are supported:

LAMBDA	estimates the original Conway-Maxwell-Poisson model (Shmueli et al. 2005).
MU	reparameterizes λ as documented by Guikema and Coffelt (2008), where $\mu = \lambda^{1/\nu}$ and the integral part of μ represents the mode, which can be considered a measure of central tendency (mean).

By default, PARAMETER=MU.

Printing Options

You can specify the following options in either the PROC CNTSELECT statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

OUTPUT Statement

OUTPUT <output-options> ;

The OUTPUT statement creates a new data table that includes variables created by the *output-options*. These variables include the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability of the response variable taking on the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$ and the probability that the response is zero as a result of the zero-generating process. For the Conway-Maxwell-Poisson model, the estimates of $\mathbf{g}_i' \boldsymbol{\delta}$, λ , ν , μ , mode, variance, and dispersion are also available. Except for the probability of the current value, these statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations that have missing response values to the input data table, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit.

You can specify only one OUTPUT statement. You can specify the following *output-options*:

OUT=CAS-libref.data-table

names the output data table for PROC LOGSELECT to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 288.

data-table specifies the name of the output data table.

COPYVAR=SAS-variable-names

COPYVARS=SAS-variable-names

adds SAS variables to the output data table.

PRED=name

MEAN=name

names the variable to contain the predicted value of the response variable.

PROB=name

names the variable to contain the probability that the response variable will take the actual value, $\Pr(Y = y_i)$.

PROBCOUNT(value1 < value2 ... >)

outputs the probability that the response variable will take particular values. Each *value* should be a nonnegative integer. If you specify a noninteger, it is rounded to the nearest integer. The *value* can also be a list of the form X TO Y BY Z. For example, PROBCOUNT(0 1 2 TO 10 BY 2 15) requests predicted probabilities for counts 0, 1, 2, 4, 5, 6, 8, 10, and 15. This option is not available for the fixed-effects and random-effects panel models.

PROBZERO=name

names the variable to contain the value of φ_i , which is the probability that the response variable will take the value of 0 as a result of the zero-generating process. This variable is written to the output file only if the model is zero-inflated.

VARIANCE=name

assigns a name to the variable that contains the estimate of variance.

XBETA=name

names the variable to contain estimates of $\mathbf{x}_i' \boldsymbol{\beta}$.

ZGAMMA=name

names the variable to contain estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$.

You can specify the following additional *output-options* if you specify a Conway-Maxwell-Poisson regression in the MODEL statement:

DISPERSION=name

assigns a name to the variable that contains the value of dispersion.

GDELTA=name

assigns a name to the variable that contains estimates of $\mathbf{g}_i' \boldsymbol{\delta}$.

LAMBDA=name

assigns a name to the variable that contains the estimate of λ .

MODE=name

assigns a name to the variable that contains the integral part of μ (mode).

MU=name

assigns a name to the variable that contains the estimate of μ .

NU=name

assigns a name to the variable that contains the estimate of ν .

RESTRICT Statement

RESTRICT *restriction1* [, *restriction2* ...] ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written either as a single linear equation or as a comma-separated list of two or more linear equations. A restriction equation consists of an *expression*, followed by an equality *operator* (=) or an inequality *operator* (<, >, <=, >=), followed by a second *expression*:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

RESTRICT Statement Expressions

A restriction expression is composed of parameter names, constants, and the operators times (*), plus (+), and minus (−). Each restriction expression must be a linear function of the parameters in the model. In addition, no grouping symbols (such as parentheses) are allowed, and the constant factor in any product can appear only on the left-hand side of the times (*) operator.

In the following example, the assumption is that you have a data table in which *y* is the count variable and *x1*–*x3* are continuous variables. The PROC CNTSELECT program uses a RESTRICT statement to impose a restriction on the estimate for the parameter that is associated with the variable *x2*. Thus, in any solution that the optimizer finds, the solution must satisfy the condition that the parameter that is associated with the variable *x2* is equal to 1.5:

```
proc cntselect data=mycas.exrestrict;
  model y = x1-x3;
  restrict x2=1.5;
run;
```

It is important to keep in mind that the parameters that are associated with the variables, not the variables themselves, are restricted. Thus, in the preceding RESTRICT statement, the variable name “*x2*” refers to the parameter that is associated with the variable *x2*, not to the variable itself.

Parameter names are shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter in order to refer to that parameter in the RESTRICT statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, which shows the internal name of each parameter.

For more information about how parameters are named, see the section “[Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 326.

Restrictions should be consistent and not redundant. All restriction equations in all RESTRICT statements are applied jointly.

RESTRICT Statement Examples

Examples of valid RESTRICT statements include the following:

```
restrict x1=0.1;
restrict a+b=1;
restrict a-b=0, b+c=1.5;
restrict 2*f=g+h, intercept+f=0;
```

Examples of invalid RESTRICT statements include the following:

```
restrict x1^2=4;
restrict x1*x3=4;
restrict x1/x3=2;
restrict sin(a)=0;
restrict a*0.5=1;
restrict 2*(f+h)=1;
```

In the first four examples of invalid RESTRICT statements, the equation is nonlinear. The fifth example is invalid because the constant factor (0.5) cannot appear on the right-hand side of the times (*) operator. The last example is invalid because grouping symbols are not allowed.

The set of restrictions must be consistent. For example, you cannot specify the following because the three restrictions are not consistent:

```
restrict f-g=0,
        f-intercept=0,
        g-intercept=1;
```

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The probabilities of these Lagrange multipliers are computed using a beta distribution (LaMotte 1994). Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

The following RESTRICT statement constrains the negative binomial dispersion parameter α to 1, which restricts the conditional variance to be $\mu + \mu^2$:

```
restrict _Alpha = 1;
```

SELECTION Statement

SELECTION < **METHOD**=*method* < (*method-options*) > > < *options* > ;

The SELECTION statement performs model selection by examining whether effects should be added to or removed from the model according to rules that are defined by the selection methods. The statement is fully documented in the section “[SELECTION Statement](#)” on page 36 in Chapter 3, “[Shared Concepts](#).” The CNTSELECT procedure supports only a subset of the *method* values and *method-options* that are listed in that section.

You can use the following *option* to specify the selection method:

METHOD=BACKWARD | FORWARD | FORWARDSWAP | STEPWISE

specifies the selection method to use. You can specify one of the following methods:

BACKWARD	specifies the backward elimination method.
FORWARD	specifies the forward selection method.
FORWARDSWAP	specifies forward-swap selection, which is an extension of the forward selection method.
STEPWISE	specifies the stepwise regression method.

By default, METHOD=STEPWISE. For more information about each of the selection methods, see the section “[Model Selection Methods](#)” on page 59

PROC CNTSELECT supports a specific set of values for the following *method-options*:

CHOOSE=*criterion*

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. PROC CNTSELECT supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the value of the CHOOSE= option is the same as the value that you assign to the SELECT= option. For more information, see the detailed description of the [CHOOSE=](#) option.

SELECT=*criterion*

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. PROC CNTSELECT supports the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, SELECT=SBC. For more information, see the detailed description of the [SELECT=](#) option.

STOP=*criterion*

specifies a criterion to use to stop the selection process. PROC CNTSELECT supports only the following values for *criterion*:

- AIC** specifies Akaike's information criterion.
- SBC** specifies the Schwarz Bayesian information criterion.

By default, the value of the STOP= option is the same as the value that you assign to the SELECT= option. For more information, see the detailed description of the [STOP=](#) option.

PROC CNTSELECT supports the following *method-options*, which are described in the section “[SELECTION Statement](#)” on page 36 in Chapter 3, “[Shared Concepts](#)”:

- [COMPETITIVE](#)
- [MAXEFFECTS=](#)
- [MAXSTEPS=](#)
- [MINEFFECTS=](#)

PROC CNTSELECT supports the following statement *options*, which are described in the section “[SELECTION Statement](#)” on page 36 in Chapter 3, “[Shared Concepts](#)”:

- [DETAILS=](#)
- [STOPHORIZON=](#)

TEST Statement

`<label> TEST <'string'> equation1 <, equation2... > /< test-options> ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters that are specified in the preceding MODEL statement.

Each *test* is written either as a single linear *equation* or as a comma-separated list of two or more linear *equations*. A test *equation* specifies a linear hypothesis to be tested and consists of an *expression*, followed by the equality operator (=), followed by a second expression:

expression = *expression*

The rules governing valid test expressions are the same as those for restriction expressions. For more information see the section “[RESTRICT Statement Expressions](#)” on page 307.

All hypotheses in one TEST statement are tested jointly.

You can specify the following *test-options* after a slash (/):

ALL

requests Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

By default, the Wald test is performed.

You can add a label (which is printed in the output) to a TEST statement in two ways: add an unquoted *label* followed by a colon before the TEST keyword, or add a quoted *string* after the TEST keyword. The unquoted *label* cannot contain any spaces. If you include both an unquoted *label* and a quoted *string*, PROC CNTSELECT uses the unquoted *label*. If you specify neither an unquoted *label* nor a quoted *string*, PROC CNTSELECT automatically labels the tests.

The following illustrates the use of the TEST statement:

```
proc cntselect;
  model y = x1 x2 x3;
  test x1 = 0, 1.5 * x2 + 2 * x3 = 0;
  test_int: test intercept = 0, x3 = 0.75;
run;
```

In the example, two separate tests are performed. The first test investigates the joint hypothesis that

$$\beta_1 = 0$$

and

$$1.5\beta_2 + 2\beta_3 = 0$$

The second test is labeled “test_int” and investigates the joint hypothesis that

$$\beta_{\text{Intercept}} = 0$$

and

$$\beta_3 = 0.75$$

VIEWSTORE Statement

VIEWSTORE / *options* ;

The VIEWSTORE statement enables you to print various reports that describe the model fitting information that is preserved in an item store. You must have created the item store by using the STORE= option in the PROC CNTSELECT statement. You can specify the following *options*:

ALL

prints the entire contents of the item store.

BYVARVALUES

prints the values of the BY-group variables that were used to fit the model.

CLASSVARLEVELS

prints the classification variables and levels that were used in the model.

COVARIANCES

prints the covariances matrix.

CORRELATIONS

prints the correlations matrix.

FINALESTIMATES

prints the final parameter estimates.

FITMODELSUMMARY

prints the fit model summary.

INITIALESTIMATES

prints the initial parameter estimates.

INSTORE=SAS-item-store

specifies the item store that you want to view.

MINIMAL

prints only the fit model summary and final parameter estimates.

MODELDEFINITION

prints the statements that defined the model.

NONE

suppresses printing.

OPTIMIZERSETTINGS

prints the optimizer settings that were used to fit the model.

WEIGHT Statement

WEIGHT *variable* *</option>* ;

The WEIGHT statement specifies a variable to supply weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

The following *option* can be added to the WEIGHT statement after a slash (/):

NONNORMALIZE

does not normalize the weights. (By default, the weights are normalized so that they add up to the actual sample size. The weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^n w_i}$, where n is the sample size.) If the weights are required to be used as they are, then specify the NONNORMALIZE option.

ZEROMODEL Statement

ZEROMODEL *dependent-variable* *~ zero-inflated-regressors* *</options>* ;

The ZEROMODEL statement is required if either ZIP or ZINB is specified in the DIST= option in the MODEL statement. If ZIP or ZINB is specified, then the ZEROMODEL statement must follow the MODEL statement. The dependent variable in the ZEROMODEL statement must be the same as the dependent variable in the MODEL statement.

The zero-inflated (ZI) regressors appear in the equation that determines the probability (φ_i) of a zero count. Each of these q variables has a parameter to be estimated in the regression. For example, let $\mathbf{z}'_i$ be the i th observation's $1 \times (q + 1)$ vector of values of the q ZI explanatory variables (w_0 is set to 1 for the intercept term). Then φ_i is a function of $\mathbf{z}'_i \boldsymbol{\gamma}$, where $\boldsymbol{\gamma}$ is the $(q + 1) \times 1$ vector of parameters to be estimated. (The zero-inflated intercept is γ_0 ; the coefficients for the q zero-inflated covariates are $\gamma_1, \dots, \gamma_q$.) If q is equal to 0 (no ZI explanatory variables are provided), then only the intercept term γ_0 is estimated. The “Parameter Estimates” table in the displayed output shows the estimates for the ZI intercept and ZI explanatory variables; they are labeled with the prefix “Inf_”. For example, the ZI intercept is labeled “Inf_intercept”. If you specify Age (a variable in your data table) as a ZI explanatory variable, then the “Parameter Estimates” table labels the corresponding parameter estimate “Inf_Age”.

You can specify the following *options* in the ZEROMODEL statement after a slash (/):

LINK=LOGISTIC | NORMAL

specifies the distribution function to use to compute probability of zeros. You can specify the following functions:

LOGISTIC specifies logistic distribution.

NORMAL specifies standard normal distribution.

If this option is omitted, then the default ZI link function is logistic.

OFFSET=zero-inflated-offset-variable

specifies a variable in the input data table to be used as a zero-inflated (ZI) offset variable. The ZI offset variable *zero-inflated-offset-variable* is included as a term, with coefficient restricted to 1, in the equation that determines the probability (φ_i) of a zero count and represents an adjustment to a common observational unit. The ZI offset variable cannot be the response variable, the offset variable (if any), or any of the explanatory variables. The name of the data table variable that is used as the ZI offset variable is displayed in the “Model Fit Summary” table, where it is labeled as “Inf_offset”.

Details: CNTSELECT Procedure

Missing Values

Any observations in the input data table that have a missing value for one or more of the regressors are ignored by PROC CNTSELECT and not used in the model fit. PROC CNTSELECT rounds any positive noninteger count values to the nearest integer and ignores any observations that have a negative count.

If the input data table contains any observations that have missing response values but nonmissing regressors, PROC CNTSELECT can compute several statistics and store them in an output data table by using the OUTPUT statement. For example, you can request that the output data table contain the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability that the response variable will take the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$, and the probability that the response is 0 as a result of the zero-generating process. Note that the presence of such observations (that have missing response values) does not affect the model fit.

Poisson Regression

The most widely used model for count data analysis is Poisson regression. Poisson regression assumes that y_i , given the vector of covariates $\mathbf{x}_i$, is independently Poisson distributed with

$$P(Y_i = y_i | \mathbf{x}_i) = \frac{e^{-\mu_i} \mu_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \dots$$

and the mean parameter—that is, the mean number of events per period—is given by

$$\mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

where $\boldsymbol{\beta}$ is a $(k + 1) \times 1$ parameter vector. (The intercept is β_0 ; the coefficients for the k regressors are $\beta_1, \dots, \beta_k$.) Taking the exponential of $\mathbf{x}_i' \boldsymbol{\beta}$ ensures that the mean parameter μ_i is nonnegative. It can be shown that the conditional mean is given by

$$E(y_i | \mathbf{x}_i) = \mu_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

Note that the conditional variance of the count random variable is equal to the conditional mean in the Poisson regression model:

$$V(y_i | \mathbf{x}_i) = E(y_i | \mathbf{x}_i) = \mu_i$$

The equality of the conditional mean and variance of y_i is known as *equidispersion*.

The standard estimator for the Poisson model is the maximum likelihood estimator (MLE). Because the observations are independent, the log-likelihood function is written as

$$\mathcal{L} = \sum_{i=1}^N (-\mu_i + y_i \ln \mu_i - \ln y_i!) = \sum_{i=1}^N (-e^{\mathbf{x}_i' \boldsymbol{\beta}} + y_i \mathbf{x}_i' \boldsymbol{\beta} - \ln y_i!)$$

For more information about the Poisson regression model, see [SAS/ETS User's Guide](#).

The Poisson model has been criticized for its restrictive property that the conditional variance equals the conditional mean. Real-life data are often characterized by *overdispersion*—that is, the variance exceeds the mean. Allowing for overdispersion can improve model predictions because the Poisson restriction of equal mean and variance results in the underprediction of zeros when overdispersion exists. The most commonly used model that accounts for overdispersion is the negative binomial model. Conway-Maxwell-Poisson regression enables you to model both overdispersion and underdispersion.

Conway-Maxwell-Poisson Regression

The Conway-Maxwell-Poisson (CMP) distribution is a generalization of the Poisson distribution that enables you to model both underdispersed and overdispersed data. It was originally proposed by Conway and Maxwell (1962), but its implementation to model under- and overdispersed count data is attributed to Shmueli et al. (2005).

Recall that y_i , given the vector of covariates $\mathbf{x}_i$, is independently Poisson-distributed as

$$P(Y_i = y_i | \mathbf{x}_i) = \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \dots$$

The Conway-Maxwell-Poisson distribution is defined as

$$P(Y_i = y_i | \mathbf{x}_i, \mathbf{z}_i) = \frac{1}{Z(\lambda_i, \nu_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{\nu_i}}, \quad y_i = 0, 1, 2, \dots$$

where the normalization factor is

$$Z(\lambda_i, \nu_i) = \sum_{n=0}^{\infty} \frac{\lambda_i^n}{(n!)^{\nu_i}}$$

and

$$\lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

$$\nu_i = -\exp(\mathbf{g}_i' \boldsymbol{\delta})$$

The $\boldsymbol{\beta}$ vector is a $(k + 1) \times 1$ parameter vector. (The intercept is β_0 , and the coefficients for the k regressors are $\beta_1, \dots, \beta_k$.) The $\boldsymbol{\delta}$ vector is an $(m + 1) \times 1$ parameter vector. (The intercept is represented by δ_0 , and the coefficients for the m regressors are $\delta_1, \dots, \delta_m$.) The covariates are represented by $\mathbf{x}_i$ and $\mathbf{g}_i$ vectors.

One of the restrictive properties of the Poisson model is that the conditional mean and variance must be equal:

$$E(y_i | \mathbf{x}_i) = V(y_i | \mathbf{x}_i) = \lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

The CMP distribution overcomes this restriction by defining an additional parameter, ν , which governs the rate of decay of successive ratios of probabilities such that

$$P(Y_i = y_i - 1) / P(Y_i = y_i) = \frac{(y_i)^{\nu_i}}{\lambda_i}$$

The introduction of the additional parameter, ν , allows for flexibility in modeling the tail behavior of the distribution. If $\nu = 1$, the ratio is equal to the rate of decay of the Poisson distribution. If $\nu < 1$, the rate of decay decreases, enabling you to model processes that have longer tails than the Poisson distribution (overdispersed data). If $\nu > 1$, the rate of decay increases in a nonlinear fashion, thus shortening the tail of the distribution (underdispersed data).

There are several special cases of the Conway-Maxwell-Poisson distribution. If $\lambda < 1$ and $\nu \rightarrow \infty$, the Conway-Maxwell-Poisson results in the Bernoulli distribution. In this case, the data can take only the values 0 and 1, which represents an extreme underdispersion. If $\nu = 1$, the Poisson distribution is recovered with

its equidispersion property. When $\nu = 0$ and $\lambda < 1$, the normalization factor is convergent and forms a geometric series,

$$Z(\lambda_i, 0) = \frac{1}{1 - \lambda_i}$$

and the probability density function becomes

$$P(Y = y_i; \lambda_i, \nu_i = 0) = (1 - \lambda_i) \lambda_i^{y_i}$$

The geometric distribution represents a case of severe overdispersion.

Mean, Variance, and Dispersion for the Conway-Maxwell-Poisson Model

The mean and the variance of the Conway-Maxwell-Poisson distribution are defined as

$$E[Y] = \frac{\partial \ln Z}{\partial \ln \lambda}$$

$$V[Y] = \frac{\partial^2 \ln Z}{\partial^2 \ln \lambda}$$

The Conway-Maxwell-Poisson distribution does not have closed-form expressions for its moments in terms of its parameters λ and ν . However, the moments can be approximated. Shmueli et al. (2005) use asymptotic expressions for Z to derive $E(Y)$ and $V(Y)$ as

$$E[Y] \approx \lambda^{1/\nu} + \frac{1}{2\nu} - \frac{1}{2}$$

$$V[Y] \approx \frac{1}{\nu} \lambda^{1/\nu}$$

In the Conway-Maxwell-Poisson model, the summation of infinite series is evaluated using a logarithmic expansion. The mean and variance are calculated as follows for the Shmueli et al. (2005) model:

$$E(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j \lambda^j}{(j!)^\nu}$$

$$V(Y) = \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j^2 \lambda^j}{(j!)^\nu} - E(Y)^2$$

The dispersion is defined as

$$D(Y) = \frac{V(Y)}{E(Y)}$$

Likelihood Function for the Conway-Maxwell-Poisson Model

The likelihood for a set of n independently and identically distributed variables $y_1, y_2, \dots, y_n$ is written as

$$\begin{aligned} L(y_1, y_2, \dots, y_n | \lambda, \nu) &= \frac{\prod_{i=1}^n \lambda^{y_i}}{(\prod_{i=1}^n y_i!)^\nu} Z(\lambda, \nu)^{-n} \\ &= \lambda^{\sum_{i=1}^n y_i} \exp\left(-\nu \sum_{i=1}^n \ln(y_i!)\right) Z(\lambda, \nu)^{-n} \\ &= \lambda^{S_1} \exp(-\nu S_2) Z(\lambda, \nu)^{-n} \end{aligned}$$

where S_1 and S_2 are sufficient statistics for $y_1, y_2, \dots, y_n$. You can see from the preceding equation that the Conway-Maxwell-Poisson distribution is a member of the exponential family. The log-likelihood function can be written as

$$\mathcal{L} = -n \ln(Z(\lambda, \nu)) + \sum_{i=1}^n (y_i \ln(\lambda) - \nu \ln(y_i!))$$

The gradients can be written as

$$\begin{aligned} \mathcal{L}_\beta &= \left(\sum_{k=1}^N y_k - n \frac{\lambda Z(\lambda, \nu)_\lambda}{Z(\lambda, \nu)} \right) \mathbf{x} \\ \mathcal{L}_\delta &= \left(\sum_{k=1}^N \ln(y_k!) - n \frac{Z(\lambda, \nu)_\nu}{Z(\lambda, \nu)} \right) \nu \mathbf{z} \end{aligned}$$

Conway-Maxwell-Poisson Regression: Guikema and Coffelt (2008) Reparameterization

Guikema and Coffelt (2008) propose a reparameterization of the Shmueli et al. (2005) Conway-Maxwell-Poisson model to provide a measure of central tendency that can be interpreted in the context of the generalized linear model. By substituting $\lambda = \mu^\nu$, the Guikema and Coffelt (2008) formulation is written as

$$P(Y = y_i; \mu, \nu) = \frac{1}{S(\mu, \nu)} \left(\frac{\mu^{y_i}}{y_i!} \right)^\nu$$

where the new normalization factor is defined as

$$S(\mu, \nu) = \sum_{j=0}^{\infty} \left(\frac{\mu^j}{j!} \right)^\nu$$

In terms of their new formulations, the mean and variance of Y are given as

$$\begin{aligned} E[Y] &= \frac{1}{\nu} \frac{\partial \ln S}{\partial \ln \mu} \\ V[Y] &= \frac{1}{\nu^2} \frac{\partial^2 \ln S}{\partial^2 \ln \mu} \end{aligned}$$

They can be approximated as

$$E[Y] \approx \mu + \frac{1}{2}\nu - \frac{1}{2}$$

$$V[Y] \approx \frac{\mu}{\nu}$$

In the CNTSELECT procedure, the mean and variance are calculated according to the following formulas, respectively, for the Guikema and Coffelt (2008) model:

$$E(Y) = \frac{1}{Z(\lambda, \mu)} \sum_{j=0}^{\infty} \frac{j\mu^{\nu j}}{(j!)^{\nu}}$$

$$V(Y) = \frac{1}{Z(\lambda, \mu)} \sum_{j=0}^{\infty} \frac{j^2\mu^{\nu j}}{(j!)^{\nu}} - E(Y)^2$$

In terms of the new parameter μ , the log-likelihood function is specified as

$$\mathcal{L} = \ln(S(\mu, \nu)) + \nu \sum_{i=1}^N (y_i \ln(\mu) - \ln(y_i!))$$

and the gradients are calculated as

$$\mathcal{L}_{\beta} = \left(\nu \sum_{i=1}^N y_i - \frac{\mu S(\mu, \nu)_{\mu}}{S(\mu, \nu)} \right) \mathbf{x}$$

$$\mathcal{L}_{\delta} = \left(\sum_{i=1}^N (y_i \ln(\mu) - \ln(y_i!)) - \frac{S(\mu, \nu)_{\nu}}{S(\mu, \nu)} \right) \nu \mathbf{g}$$

By default, the CNTSELECT procedure uses the Guikema and Coffelt (2008) specification. The Shmueli et al. (2005) model can be estimated by specifying the `PARAMETER=LAMBDA` option. If you specify `DISP=CMPOISSON` in the `MODEL` statement and you omit the `DISPMODEL` statement, the model is estimated according to the Lord, Guikema, and Geedipally (2008) specification, where ν represents a single parameter that does not depend on any covariates. The Lord, Guikema, and Geedipally (2008) specification makes the model comparable to the negative binomial model because it has only one parameter.

The dispersion is defined as

$$D(Y) = \frac{V(Y)}{E(Y)}$$

Using the Guikema and Coffelt (2008) specification results in the integral part of μ representing the mode, which is a reasonable approximation for the mean. The dispersion can be written as

$$D(Y) = \frac{V(Y)}{E(Y)} \approx \frac{\frac{\mu}{\nu}}{\mu + \frac{1}{2}\nu - \frac{1}{2}} \approx \frac{1}{\nu}$$

When $\nu < 1$, the variance can be shown to be greater than the mean and the dispersion greater than 1. This is a result of overdispersed data. When $\nu = 1$ and the mean and variance are equal, the dispersion is equal to 1 (Poisson model). When $\nu > 1$, the variance is smaller than the mean and the dispersion is less than 1. This is a result of underdispersed data.

All Conway-Maxwell-Poisson models in the CNTSELECT procedure are parameterized in terms of dispersion, where

$$-\ln(\nu) = \delta_0 + \sum_{n=1}^q \delta_n g_n$$

Negative values of $\ln(\nu)$ indicate that the data are approximately overdispersed, and positive values of $\ln(\nu)$ indicate that the data are approximately underdispersed.

Negative Binomial Regression

The Poisson regression model can be generalized by introducing an unobserved heterogeneity term for observation i . Thus, the individuals are assumed to differ randomly in a manner that is not fully accounted for by the observed covariates. This is formulated as

$$E(y_i | \mathbf{x}_i, \tau_i) = \mu_i \tau_i = e^{\mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i}$$

where the unobserved heterogeneity term $\tau_i = e^{\epsilon_i}$ is independent of the vector of regressors $\mathbf{x}_i$. Then the distribution of y_i conditional on $\mathbf{x}_i$ and τ_i is Poisson with conditional mean and conditional variance $\mu_i \tau_i$:

$$f(y_i | \mathbf{x}_i, \tau_i) = \frac{\exp(-\mu_i \tau_i) (\mu_i \tau_i)^{y_i}}{y_i!}$$

Let $g(\tau_i)$ be the probability density function of τ_i . Then, the distribution $f(y_i | \mathbf{x}_i)$ (no longer conditional on τ_i) is obtained by integrating $f(y_i | \mathbf{x}_i, \tau_i)$ with respect to τ_i :

$$f(y_i | \mathbf{x}_i) = \int_0^\infty f(y_i | \mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i$$

An analytical solution to this integral exists when τ_i is assumed to follow a gamma distribution. This solution is the negative binomial distribution. If the model contains a constant term, then in order to identify the mean of the distribution, it is necessary to assume that $E(e^{\epsilon_i}) = E(\tau_i) = 1$. Thus, it is assumed that τ_i follows a gamma(θ, θ) distribution with $E(\tau_i) = 1$ and $V(\tau_i) = 1/\theta$,

$$g(\tau_i) = \frac{\theta^\theta}{\Gamma(\theta)} \tau_i^{\theta-1} \exp(-\theta \tau_i)$$

where $\Gamma(x) = \int_0^\infty z^{x-1} \exp(-z) dz$ is the gamma function and θ is a positive parameter. Then, the density of y_i given $\mathbf{x}_i$ is derived as

$$\begin{aligned} f(y_i|\mathbf{x}_i) &= \int_0^\infty f(y_i|\mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i \\ &= \frac{\theta^\theta \mu_i^{y_i}}{y_i! \Gamma(\theta)} \int_0^\infty e^{-(\mu_i + \theta)\tau_i} \tau_i^{\theta + y_i - 1} d\tau_i \\ &= \frac{\theta^\theta \mu_i^{y_i} \Gamma(y_i + \theta)}{y_i! \Gamma(\theta) (\theta + \mu_i)^{\theta + y_i}} \\ &= \frac{\Gamma(y_i + \theta)}{y_i! \Gamma(\theta)} \left(\frac{\theta}{\theta + \mu_i} \right)^\theta \left(\frac{\mu_i}{\theta + \mu_i} \right)^{y_i} \end{aligned}$$

If you make the substitution $\alpha = \frac{1}{\theta}$ ($\alpha > 0$), the negative binomial distribution can then be rewritten as

$$f(y_i|\mathbf{x}_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i = 0, 1, 2, \dots$$

Thus, the negative binomial distribution is derived as a gamma mixture of Poisson random variables. It has the conditional mean

$$E(y_i|\mathbf{x}_i) = \mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}}$$

and the conditional variance

$$V(y_i|\mathbf{x}_i) = \mu_i \left[1 + \frac{1}{\theta} \mu_i \right] = \mu_i [1 + \alpha \mu_i] > E(y_i|\mathbf{x}_i)$$

The conditional variance of the negative binomial distribution exceeds the conditional mean. Overdispersion results from neglected unobserved heterogeneity. The negative binomial model with variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha \mu_i^2$, which is quadratic in the mean, is referred to as the NEGBIN2 model Cameron and Trivedi (1986). To estimate this model, specify DIST=NEGBIN(P=2) in the MODEL statement. The Poisson distribution is a special case of the negative binomial distribution where $\alpha = 0$. A test of the Poisson distribution can be carried out by testing the hypothesis that $\alpha = \frac{1}{\theta_i} = 0$. A Wald test of this hypothesis is provided (it is the reported t statistic for the estimated α in the negative binomial model).

The log-likelihood function of the negative binomial regression model (NEGBIN2) is given by

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^N \left\{ \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) - \ln(y_i!) \right. \\ &\quad \left. - (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta})) + y_i \ln(\alpha) + y_i \mathbf{x}_i' \boldsymbol{\beta} \right\} \end{aligned}$$

where use of the following fact is made if y is an integer:

$$\Gamma(y + a) / \Gamma(a) = \prod_{j=0}^{y-1} (j + a)$$

Cameron and Trivedi (1986) consider a general class of negative binomial models that have mean μ_i and variance function $\mu_i + \alpha\mu_i^p$. The NEGBIN2 model, with $p = 2$, is the standard formulation of the negative binomial model. Models that have other values of p , $-\infty < p < \infty$, have the same density $f(y_i|\mathbf{x}_i)$, except that α^{-1} is replaced everywhere by $\alpha^{-1}\mu_i^{2-p}$. The negative binomial model NEGBIN1, which sets $p = 1$, has the variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha\mu_i$, which is linear in the mean. To estimate this model, specify `DIST=NEGBIN(P=1)` in the `MODEL` statement.

The log-likelihood function of the NEGBIN1 regression model is given by

$$\mathcal{L} = \sum_{i=1}^N \left\{ \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1} \exp(\mathbf{x}_i' \boldsymbol{\beta})) - \ln(y_i!) - (y_i + \alpha^{-1} \exp(\mathbf{x}_i' \boldsymbol{\beta})) \ln(1 + \alpha) + y_i \ln(\alpha) \right\}$$

For more information about the negative binomial regression model, see *SAS/ETS User's Guide*.

Zero-Inflated Count Regression Overview

The main motivation for using zero-inflated count models is that real-life data frequently display overdispersion and excess zeros. Zero-inflated count models provide a way to both model the excess zeros and allow for overdispersion. In particular, there are two possible data generation processes for each observation. The result of a Bernoulli trial is used to determine which of the two processes to use. For observation i , Process 1 is chosen with probability φ_i and Process 2 with probability $1 - \varphi_i$. Process 1 generates only zero counts. Process 2 generates counts from either a Poisson or a negative binomial model. In general,

$$y_i \sim \begin{cases} 0 & \text{with probability } \varphi_i \\ g(y_i) & \text{with probability } 1 - \varphi_i \end{cases}$$

Therefore, the probability of $\{Y_i = y_i\}$ can be described as

$$\begin{aligned} P(y_i = 0|\mathbf{x}_i) &= \varphi_i + (1 - \varphi_i)g(0) \\ P(y_i|\mathbf{x}_i) &= (1 - \varphi_i)g(y_i), \quad y_i > 0 \end{aligned}$$

where $g(y_i)$ follows either the Poisson or the negative binomial distribution.

If the probability φ_i depends on the characteristics of observation i , then φ_i is written as a function of $\mathbf{z}_i' \boldsymbol{\gamma}$, where $\mathbf{z}_i'$ is the $1 \times (q + 1)$ vector of zero-inflated covariates and $\boldsymbol{\gamma}$ is the $(q + 1) \times 1$ vector of zero-inflated coefficients to be estimated. (The zero-inflated intercept is γ_0 ; the coefficients for the q zero-inflated covariates are $\gamma_1, \dots, \gamma_q$.) The function F that relates the product $\mathbf{z}_i' \boldsymbol{\gamma}$ (which is a scalar) to the probability φ_i is called the zero-inflated link function,

$$\varphi_i = F_i = F(\mathbf{z}_i' \boldsymbol{\gamma})$$

In the `CNTSELECT` procedure, the zero-inflated covariates are indicated in the `ZEROMODEL` statement. Furthermore, the zero-inflated link function F can be specified as either the logistic function,

$$F(\mathbf{z}_i' \boldsymbol{\gamma}) = \Lambda(\mathbf{z}_i' \boldsymbol{\gamma}) = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

or the standard normal cumulative distribution function (also called the probit function),

$$F(\mathbf{z}'_i \boldsymbol{\gamma}) = \Phi(\mathbf{z}'_i \boldsymbol{\gamma}) = \int_0^{\mathbf{z}'_i \boldsymbol{\gamma}} \frac{1}{\sqrt{2\pi}} \exp(-u^2/2) du$$

The zero-inflated link function is indicated by using the LINK= option in the ZEROMODEL statement. The default ZI link function is the logistic function.

Zero-Inflated Poisson Regression

In the zero-inflated Poisson (ZIP) regression model, the data generation process that is referred to earlier as Process 2 is

$$g(y_i) = \frac{\exp(-\mu_i) \mu_i^{y_i}}{y_i!}$$

where $\mu_i = e^{\mathbf{x}'_i \boldsymbol{\beta}}$. Thus the ZIP model is defined as

$$\begin{aligned} P(y_i = 0 | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) \exp(-\mu_i) \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{\exp(-\mu_i) \mu_i^{y_i}}{y_i!}, \quad y_i > 0 \end{aligned}$$

The conditional expectation and conditional variance of y_i are given by

$$\begin{aligned} E(y_i | \mathbf{x}_i, \mathbf{z}_i) &= \mu_i (1 - F_i) \\ V(y_i | \mathbf{x}_i, \mathbf{z}_i) &= E(y_i | \mathbf{x}_i, \mathbf{z}_i) (1 + \mu_i F_i) \end{aligned}$$

Note that the ZIP model (in addition to the ZINB model) exhibits overdispersion because $V(y_i | \mathbf{x}_i, \mathbf{z}_i) > E(y_i | \mathbf{x}_i, \mathbf{z}_i)$.

In general, the log-likelihood function of the ZIP model is

$$\mathcal{L} = \sum_{i=1}^N \ln [P(y_i | \mathbf{x}_i, \mathbf{z}_i)]$$

After a specific link function (either logistic or standard normal) for the probability φ_i is chosen, it is possible to write the exact expressions for the log-likelihood function and the gradient.

ZIP Model with Logistic Link Function

First, consider the ZIP model in which the probability φ_i is expressed by a logistic link function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}'_i \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}'_i \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned}\mathcal{L} = & \sum_{\{i:y_i=0\}} \ln [\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + \exp(-\exp(\mathbf{x}_i' \boldsymbol{\beta}))] \\ & + \sum_{\{i:y_i>0\}} \left[y_i \mathbf{x}_i' \boldsymbol{\beta} - \exp(\mathbf{x}_i' \boldsymbol{\beta}) - \sum_{k=2}^{y_i} \ln(k) \right] \\ & - \sum_{i=1}^N \ln [1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})]\end{aligned}$$

ZIP Model with Standard Normal Link Function

Next, consider the ZIP model in which the probability φ_i is expressed by a standard normal link function: $\varphi_i = \Phi(\mathbf{z}_i' \boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned}\mathcal{L} = & \sum_{\{i:y_i=0\}} \ln \{ \Phi(\mathbf{z}_i' \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma})] \exp(-\exp(\mathbf{x}_i' \boldsymbol{\beta})) \} \\ & + \sum_{\{i:y_i>0\}} \left\{ \ln [(1 - \Phi(\mathbf{z}_i' \boldsymbol{\gamma}))] - \exp(\mathbf{x}_i' \boldsymbol{\beta}) + y_i \mathbf{x}_i' \boldsymbol{\beta} - \sum_{k=2}^{y_i} \ln(k) \right\}\end{aligned}$$

For more information about the zero-inflated Poisson regression model, see *SAS/ETS User's Guide*.

Zero-Inflated Conway-Maxwell-Poisson Regression

In the Conway-Maxwell-Poisson regression model, the data generation process is defined as

$$P(Y_i = y_i | \mathbf{x}_i, \mathbf{z}_i) = \frac{1}{Z(\lambda_i, v_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}}, \quad y_i = 0, 1, 2, \dots$$

where the normalization factor is

$$Z(\lambda_i, v_i) = \sum_{n=0}^{\infty} \frac{\lambda_i^n}{(n!)^{v_i}}$$

and

$$\lambda_i = \exp(\mathbf{x}_i' \boldsymbol{\beta})$$

$$v_i = -\exp(\mathbf{g}_i' \boldsymbol{\delta})$$

The zero-inflated Conway-Maxwell-Poisson model can be written as

$$\begin{aligned}P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) \frac{1}{Z(\lambda_i, v_i)}, \quad y_i = 0 \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{1}{Z(\lambda_i, v_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}}, \quad y_i > 0\end{aligned}$$

The conditional expectation and conditional variance of y_i are given respectively by

$$E(y_i | \mathbf{x}_i, \mathbf{z}_i) = (1 - F_i) \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j \lambda^j}{(j!)^\nu}$$

$$V(y_i | \mathbf{x}_i, \mathbf{z}_i) = (1 - F_i) \frac{1}{Z(\lambda, \nu)} \sum_{j=0}^{\infty} \frac{j^2 \lambda^j}{(j!)^\nu} - E(y_i | \mathbf{x}_i, \mathbf{z}_i)^2$$

The general form of the log-likelihood function for the Conway-Maxwell-Poisson zero-inflated model is

$$\mathcal{L} = \sum_{i=1}^N w_i \ln [P(y_i | \mathbf{x}_i, \mathbf{z}_i)]$$

Zero-Inflated Conway-Maxwell-Poisson Model with Logistic Link Function

For this model, the probability φ_i is expressed by using a logistic link function as

$$\varphi_i = \Lambda(\mathbf{z}'_i \boldsymbol{\gamma}) = \frac{\exp(\mathbf{z}'_i \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}'_i \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} w_i \ln \left\{ \Lambda(\mathbf{z}'_i \boldsymbol{\gamma}) + [1 - \Lambda(\mathbf{z}'_i \boldsymbol{\gamma})] \frac{1}{Z(\lambda_i, \nu_i)} \right\} \\ & + \sum_{\{i: y_i>0\}} w_i \{ \ln [(1 - \Lambda(\mathbf{z}'_i \boldsymbol{\gamma}))] - \ln(Z(\lambda, \nu)) + (y_i \ln(\lambda) - \nu \ln(y_i!)) \} \end{aligned}$$

Zero-Inflated Conway-Maxwell-Poisson Model with Normal Link Function

For this model, the probability φ_i is specified by using the standard normal distribution function (probit function): $\varphi_i = \Phi(\mathbf{z}'_i \boldsymbol{\gamma})$.

The log-likelihood function is written as

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} w_i \ln \left\{ \Phi(\mathbf{z}'_i \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma})] \frac{1}{Z(\lambda_i, \nu_i)} \right\} \\ & + \sum_{\{i: y_i>0\}} w_i \{ \ln [(1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma}))] - \ln(Z(\lambda, \nu)) + (y_i \ln(\lambda) - \nu \ln(y_i!)) \} \end{aligned}$$

Zero-Inflated Negative Binomial Regression

The zero-inflated negative binomial (ZINB) model in PROC CNTSELECT is based on the negative binomial model that has a quadratic variance function (when DIST=NEGBIN in the MODEL or PROC CNTSELECT statement). The ZINB model is obtained by specifying a negative binomial distribution for the data generation process referred to earlier as Process 2:

$$g(y_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}$$

Thus the ZINB model is defined to be

$$\begin{aligned} P(y_i = 0 | \mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i) (1 + \alpha \mu_i)^{-\alpha^{-1}} \\ P(y_i | \mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i) \frac{\Gamma(y_i + \alpha^{-1})}{y_i! \Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \\ &\quad \times \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i > 0 \end{aligned}$$

In this case, the conditional expectation (E) and conditional variance (V) of y_i are

$$E(y_i | \mathbf{x}_i, \mathbf{z}_i) = \mu_i (1 - F_i)$$

$$V(y_i | \mathbf{x}_i, \mathbf{z}_i) = E(y_i | \mathbf{x}_i, \mathbf{z}_i) [1 + \mu_i (F_i + \alpha)]$$

Like the ZIP model, the ZINB model exhibits overdispersion because the conditional variance exceeds the conditional mean.

ZINB Model with Logistic Link Function

In this model, the probability φ_i is given by the logistic function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned} \mathcal{L} &= \sum_{\{i: y_i=0\}} \ln \left[\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + (1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta}))^{-\alpha^{-1}} \right] \\ &+ \sum_{\{i: y_i>0\}} \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) \\ &+ \sum_{\{i: y_i>0\}} \{ -\ln(y_i!) - (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}_i' \boldsymbol{\beta})) + y_i \ln(\alpha) + y_i \mathbf{x}_i' \boldsymbol{\beta} \} \\ &- \sum_{i=1}^N \ln [1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})] \end{aligned}$$

ZINB Model with Standard Normal Link Function

For this model, the probability φ_i is expressed by the standard normal distribution function (probit function): $\varphi_i = \Phi(\mathbf{z}'_i \boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i=0\}} \ln \left\{ \Phi(\mathbf{z}'_i \boldsymbol{\gamma}) + [1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma})] (1 + \alpha \exp(\mathbf{x}'_i \boldsymbol{\beta}))^{-\alpha^{-1}} \right\} \\ & + \sum_{\{i: y_i>0\}} \ln [1 - \Phi(\mathbf{z}'_i \boldsymbol{\gamma})] \\ & + \sum_{\{i: y_i>0\}} \sum_{j=0}^{y_i-1} \{ \ln(j + \alpha^{-1}) \} \\ & - \sum_{\{i: y_i>0\}} \ln(y_i!) \\ & - \sum_{\{i: y_i>0\}} (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(\mathbf{x}'_i \boldsymbol{\beta})) \\ & + \sum_{\{i: y_i>0\}} y_i \ln(\alpha) \\ & + \sum_{\{i: y_i>0\}} y_i \mathbf{x}'_i \boldsymbol{\beta} \end{aligned}$$

For more information about the zero-inflated negative binomial regression model, see *SAS/ETS User's Guide*.

Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements

This section describes how you can refer to the parameters in the MODEL, ZEROMODEL, and DISPMODEL statements when you use the RESTRICT, TEST, BOUNDS, or INIT statement. The following examples use the RESTRICT statement, but the same remarks apply to naming parameters when you use the TEST, BOUNDS, or INIT statement. The names of the parameters can be seen in the “Parameter Estimates” table if you specify the option. If a parameter name contains a blank or some other special character (such as '*', '-', '(', or ')'), then you must use the internal name of the parameter in order to refer to that parameter in the RESTRICT, TEST, BOUNDS, or INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CNTSELECT statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, an extra column is added to the “Parameter Estimates” table, showing the internal name of each parameter.

To impose a restriction on a parameter that is related to a regressor in the MODEL statement, you simply use the name of the regressor itself to refer to its associated parameter. Suppose your model is

```
model y = x1 x2 x5;
```

where x1 through x5 are continuous variables. If you want to restrict the parameter associated with the regressor x5 to be greater than 1.7, then you should use the following statement:


```
RESTRICT x5 > 1.7;
```

To impose a restriction on a parameter associated with a regressor in the *ZEROMODEL* statement, you can form the name of the parameter by prefixing *Inf_* to the name of the regressor. Suppose your *MODEL* and *ZEROMODEL* statements are as follows:

```
model y = x1 x2 x5;
zeromodel y ~ x3 x5;
```

If you want to restrict the parameter related to the *x5* regressor in the *ZEROMODEL* statement to be less than 1.0, then you refer to the parameter as *Inf_x5* and provide the following statement:

```
RESTRICT Inf_x5 < 1.0;
```

Even though the regressor *x5* appears in both the *MODEL* and *ZEROMODEL* statements, the parameter associated with *x5* in the *MODEL* statement is, of course, different from the parameter associated with *x5* in the *ZEROMODEL* statement. Thus, when the name of a regressor is used in a *RESTRICT* statement without any prefix, it refers to the parameter associated with that regressor in the *MODEL* statement. Meanwhile, when the name of a regressor is used in a *RESTRICT* statement with the prefix *Inf_*, it refers to the parameter associated with that regressor in the *ZEROMODEL* statement. The parameter associated with the intercept in the *ZEROMODEL* is named *Inf_Intercept*.

In a similar way, you can form the name of a parameter associated with a regressor in the *DISPMODEL* statement by prefixing *Dsp_* to the name of the regressor. The parameter associated with the intercept in the *DISPMODEL* is named *Dsp_Intercept*.

Referring to Class-Level Parameters

When your *MODEL* includes a classification variable, you can impose restrictions on the parameters associated with each of the levels that are related to the classification variable as follows.

Suppose your classification variable is named *C* and it has three levels: 0, 1, 2. Suppose your model is the following:

```
class C;
model y = x1 x2 C;
```

Adding a classification variable as a regressor to your model introduces additional parameters into your model, each of which is associated with one of the levels of the classification variable. You can form the name of the parameter associated with a particular level of your class variable by inserting the underscore character between the name of the classification variable and the value of the level. Thus, to restrict the parameter associated with level 0 of the classification variable *C* to always be greater than 0.7, you refer to the parameter as *C_0* and provide the following statement:

```
RESTRICT C_0 > 0.7;
```

Referring to Parameters Associated with Interactions between Regressors

When a regressor in your model involves an interaction between other regressors, you can impose restrictions on the parameters associated with the interaction.

Suppose you have the following model:

```
model y = x1 x2 x3*x4;
```

You can form the name of the parameter associated with the interaction regressor $x3*x4$ by replacing the multiplication sign with an underscore. Thus, $x3_x4$ refers to the parameter that is associated with the interaction regressor $x3*x4$.

Referring to interactions between regressors and classification variables is handled in the same way. Suppose you have a classification variable that is named C and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C*x3;
```

The interaction between the continuous variable $x3$ and the classification variable C introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note how, although the order of the terms in the interaction is C followed by $x3$, the name of the parameter associated with the interaction is formed by placing the name of the continuous variable $x3$ first, followed by an underscore, followed by the name of the classification variable C , followed by an underscore, and then followed by the level value. Once again, depending on the parameterization you specify in your CLASS statement, for each interaction in your model that involves a classification variable, one of the parameters associated with that interaction might be dropped from your model prior to optimization.

The name of a parameter associated with a nested interaction is formed in a slightly different way. Suppose you have a classification variable that is named C and has three levels: 0, 1, 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 x3(C);
```

The nested interaction between the continuous variable $x3$ and the classification variable C introduces three additional parameters, which are named $x3_C_0$, $x3_C_1$, and $x3_C_2$. Note how the name in each case is formed from the name of the regressor by replacing the left and right parentheses with underscores and then appending another underscore followed by the level value.

Referring to Class Level Parameters with Negative Values

When the value of a level is a negative number, you must replace the minus sign with an underscore when you form the name of the parameter that is associated with that particular level of the classification variable. For example, suppose your classification variable is named D and has four levels: -1 , 0, 1, 2. Suppose your model is the following:

```
class D;
model y = x1 x2 D;
```

To restrict the parameter that is associated with level -1 of the classification variable D to always be less than 0.4, you refer to the parameter as D_1 (note that there are two underscores in this parameter name: one to

connect the name of the classification variable to its value and the other to replace the minus sign in the value itself) and provide the following statement:

```
RESTRICT D__1 < 0.4;
```

Dropping a Class Level Parameter to Avoid Collinearity

Depending on the parameterization you impose on your classification variable, one of the parameters associated with its levels might be dropped from your model prior to optimization in order to avoid collinearity. For example, when the default parameterization GLM is imposed, the parameter that is associated with the last level of your classification variable is dropped prior to optimization. If you attempt to impose a restriction on a dropped parameter by using the *RESTRICT* statement, PROC CNTSELECT issues an error message in the log.

For example, suppose again that your classification variable is named *C* and that it has three levels: 0, 1, 2. Suppose your model is the following:

```
class C;  
model y = x1 x2 C;
```

Because no additional options are specified in the *CLASS* statement, GLM parameterization is assumed. This means that the parameter named *C_2* (which is the parameter associated with the last level of your classification variable) will be dropped from your model before the optimizer is invoked. Therefore, an error will be issued if you attempt to restrict the *C_2* parameter in any way by referring to it in a *RESTRICT* statement. For example, the following *RESTRICT* statement will generate an error:

```
RESTRICT C_2 < 0.3;
```

Referring to Implicit Parameters

Unless you specified the *NOINT* option, you can impose a restriction on the parameter associated with the Intercept in your model simply by referring to it by name: *Intercept*.

For certain model types, one or more implicit parameters will be added to your model prior to optimization. You can impose restrictions on these implicit parameters.

For the Poisson model for which *ERRORCOMP=*RANDOM is specified, PROC CNTSELECT automatically adds the *_Alpha* parameter to your model.

If no *ERRORCOMP=* option is specified, for zero-inflated binomial and negative binomial models, PROC CNTSELECT adds the *_Alpha* parameter to the model. If *ERRORCOMP=*RANDOM is specified for the zero-inflated binomial and negative binomial models, then PROC CNTSELECT adds two implicit parameters to the model: *_Alpha* and *_Beta*.

For Conway-Maxwell Poisson models that do not include a *DISPMODEL* statement, the *_lnNu* parameter is added to the model.

Whenever your model type dictates the addition of one or more of these implicit parameters, you can impose restrictions on the implicit parameters by referring to them by name in a *RESTRICT* statement. For example, if your model type implies the existence of the *_Alpha* parameter, you can restrict *_Alpha* to be greater than 0.2 as follows:

```
RESTRICT _Alpha > 0.2;
```

Computational Resources

The time and memory that PROC CNTSELECT requires are proportional to the number of parameters in the model and the number of observations in the data table being analyzed. Less time and memory are required for smaller models and fewer observations. When PROC CNTSELECT is run in the high-performance distributed environment, the amount of time required is also affected by the number of nodes and the number of threads per node as specified in the PERFORMANCE statement.

The method that is chosen to calculate the variance-covariance matrix and the optimization method also affect the time and memory resources. All optimization methods available through the METHOD= option have similar memory use requirements. The processing time might differ for each method, depending on the number of iterations and functional calls needed. The data table is read into memory to save processing time. If not enough memory is available to hold the data, the CNTSELECT procedure stores the data in a utility file on disk and rereads the data as needed from this file, substantially increasing the execution time of the procedure. The gradient and the variance-covariance matrix must be held in memory. If the model has p parameters including the intercept, then at least $8 * (p + p * (p + 1)/2)$ bytes of memory are needed. The processing time is also a function of the number of iterations needed to converge to a solution for the model parameters. The number of iterations that are needed cannot be known in advance. You can use the MAXITER= option to limit the number of iterations that PROC CNTSELECT executes. You can alter the convergence criteria by using the nonlinear optimization options available in the PROC CNTSELECT statement. For a list of all the nonlinear optimization options, see “[Optimization Control Options](#)” on page 296.

Covariance Matrix Types

The COVEST= option in the PROC CNTSELECT statement enables you to specify the estimation method for the covariance matrix. COVEST=HESSIAN estimates the covariance matrix that is based on the inverse of the Hessian matrix; COVEST=OP uses the outer product of gradients; and COVEST=QML produces the covariance matrix that is based on both the Hessian and outer product matrices. Although all three methods produce asymptotically equivalent results, they differ in computational intensity and produce results that might differ in finite samples. The COVEST=OP option provides the covariance matrix that is typically the easiest to compute. In some cases, the OP approximation is considered more efficient than the Hessian or QML approximation because it contains fewer random elements. The QML approximation is computationally the most complex because it requires both the outer product of gradients and the Hessian matrix. In most cases, the OP or Hessian approximation is preferred to QML. The need for QML approximation arises in cases where the model is misspecified and the information matrix equality does not hold. The default is COVEST=HESSIAN.

Displayed Output

PROC CNTSELECT produces the following displayed output.

Model Fit Summary

The “Model Fit Summary” table contains the following information:

- dependent (count) variable name
- number of observations used
- number of missing values in data table, if any
- data table name
- type of model that was fit
- error components model type for panel data
- group id variable for panel data
- number of cross sections for panel data
- parameterization for the Conway-Maxwell-Poisson model
- frequency variable name, if any
- weight variable name, if any
- offset variable name, if any
- zero-inflated link function, if any
- zero-inflated offset variable name, if any
- log-likelihood value at solution
- maximum absolute gradient at solution
- number of iterations
- optimization technique used
- AIC value at solution (smaller value indicates better fit)
- SBC value at solution (smaller value indicates better fit)
- covariance matrix estimation method used

A line below the “Model Fit Summary” table indicates whether the algorithm successfully converged.

Parameter Estimates

The “Parameter Estimates” table gives the estimates of the model parameters. In zero-inflated (ZI) models, estimates are also given for the ZI intercept and ZI regressor parameters, which are labeled with the prefix “Inf_”. For example, the ZI intercept is labeled “Inf_intercept”. If you specify “Age” as a ZI regressor, then the “Parameter Estimates” table labels the corresponding parameter estimate “Inf_Age”. If you do not list any ZI regressors, then only the ZI intercept term is estimated.

If the DISPMODEL statement is specified for the Conway-Maxwell-Poisson model, the estimates are given for the dispersion intercept, and parameters are labeled with the prefix “Dsp_”. For example, the dispersion model intercept is labeled “Dsp_Intercept”. If you specify “Education” as a dispersion model regressor, then the “Parameter Estimates” table labels the corresponding parameter estimate “Dsp_Education”. If you do not list any dispersion regressors, then only the dispersion intercept is estimated.

“_Alpha” is the negative binomial dispersion parameter. The t statistic that is given for “_Alpha” is a test of overdispersion.

Covariance of Parameter Estimates

If you specify the COVB option in the PROC CNTSELECT or MODEL statement, the CNTSELECT procedure displays the estimated covariance matrix, which is defined as the inverse of the information matrix at the final iteration.

Correlation of Parameter Estimates

If you specify the CORRB option in the PROC CNTSELECT or MODEL statement, the CNTSELECT procedure displays the estimated correlation matrix, which is based on the Hessian matrix used at the final iteration.

OUTPUT OUT= Data Table

The OUTPUT statement creates a new data table that contains various estimates that you specify. You can request that the output data table contain the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the probability that the response variable will take the current value. In a zero-inflated model, you can also request that the output data table contain the estimates of $\mathbf{z}_i' \boldsymbol{\gamma}$, and the probability that the response is zero as a result of the zero-generating process. In a Conway-Maxwell-Poisson model, you can also request that the output data table contains estimates of $\mathbf{g}_i' \boldsymbol{\delta}$, λ , ν , μ , mode, variance and dispersion.

Except for the probability of the current value, these statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations with missing response values to the input data table, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit. Because of potential space limitations on the client workstation, the data table that is created by the OUTPUT statement does not contain the variables in the input data table.

ODS Table Names

PROC CNTSELECT assigns a name to each table that it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data tables. These table names are listed in [Table 8.4](#).

Table 8.4 ODS Tables Produced by PROC CNTSELECT

Table Name	Description	Statement	Option
BSplineDetails	Basis details for B-spline effects	EFFECT	SPLINE with DETAILS
ClassInfo	Level information from the CLASS statement	CLASS	Default
CollectionLevelInfo	Level details for collection effects	EFFECT	COLLECTION with DETAILS
ConvergenceStatus	Convergence status	MODEL	Default
CorrB	Correlation of parameter estimates	MODEL	CORRB
CovB	Covariance of parameter estimates	MODEL	COVB
EntryCandidates	Details about candidates for entry into the model	SELECTION	DETAILS=ALL
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
InitialParameterEstimates	Initial parameter estimates	PROC CNTSELECT or MODEL	PRINTALL
IterationHistory	Optimization iteration history	PROC CNTSELECT or MODEL	PRINTALL
MMLevelInfo	Level information for multimember effects	EFFECT	MULTIMEMBER with DETAILS
OptimizationResults	Optimization results	PROC CNTSELECT or MODEL	PRINTALL
ParameterEstimates	Parameter estimates	MODEL	Default
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	EFFECT	POLY with DETAILS
PolyScaling	Centering and scaling details	EFFECT	POLY with DETAILS
RemovalCandidates	Details about candidates for removal from the model	SELECTION	DETAILS=ALL
SelectionSummary	Summary information about model selection steps	SELECTION	Default
SplineKnots	Knot and boundary knot values	EFFECT	SPLINE with DETAILS
StopReason	Reason for termination of model selection	SELECTION	Default
TestResults	Hypothesis test statistics	TEST	Default
TimingDetails	Detailed summary of time taken for all phases of execution	PROC CNTSELECT	PRINTTIMING=(DETAILS) or PRINTALL

Table 8.4 *continued*

Table Name	Description	Statement	Option
TimingSummary	Summary of time taken for main phases of execution	PROC CNTSELECT	PRINTTIMING or PRINTALL
TPFSplineDetails	Truncated power function (TPF) spline basis details	EFFECT	SPLINE with BASIS=TPF and DETAILS
ZNLOSettings	Nonlinear optimizer settings	PROC CNTSELECT or MODEL	PRINTALL

Example: The CNTSELECT Procedure

Example 8.1: Zero-Inflated Poisson Model with CLASS Statement

This example shows the use of the CNTSELECT procedure to estimate a zero-inflated Poisson model with a classification variable `Group` that has two levels. The following DATA step generates 10,000 replicates from the zero-inflated Poisson (ZIP) model. The first 5,000 replicates belong to the first group, and the second 5,000 replicates belong to the second group. The model contains seven variables and three variables that correspond to the zero-inflated process.

```
data mycas.simulate;
  call streaminit(12345);
  array vars x1-x7;
  array zero_vars z1-z3;

  array parms{7} (.3 .4 .2 .4 -.3 -.5 -.3);
  array zero_parms{3} (-.6 .3 .2);

  intercept=0.5;
  group=1;
  z_intercept=-1;
  theta=0.5;

  do i=1 to 10000;
    sum_xb=0;
    sum_gz=0;
    if i>5000 then do;
      intercept=2;
      group=2;
    end;
    do j=1 to 7;
      vars[j]=rand('NORMAL',0,1);
      sum_xb=sum_xb+parms[j]*vars[j];
    end;
    mu=exp(intercept+sum_xb);
```



```

y_p=rand('POISSON', mu);

do j=1 to 3;
    zero_vars[j]=rand('NORMAL', 0, 1);
    sum_gz = sum_gz+zero_parms[j]*zero_vars[j];
end;
z_gamma = z_intercept+sum_gz;
pzero = cdf('LOGISTIC', z_gamma);
cut=rand('UNIFORM');
if cut<pzero then y_p=0;
output;
end;
keep y_p group x1-x7 z1-z3;
run;

```

The following statements estimate a zero-inflated Poisson model with the classification variable Group:

```

proc cntselect data=mycas.simulate dist=zip;
    class group;
    model y_p=group x1-x7;
    zeromodel y_p ~ z1-z3;
run;

```

Output 8.1.1 shows the results for the zero-inflated Poisson model.

Output 8.1.1 Zero-Inflated Poisson Model with CLASS Statement

The CNTSELECT Procedure

Class Level Information	
Class	Levels Values
group	2 1 2

Model Fit Summary	
Dependent Variable	y_p
Number of Observations	10000
Data Set	SIMULATE
Model	ZIP
ZI Link Function	Logistic
Log Likelihood	-18176.3
Maximum Absolute Gradient	0.001931
Number of Iterations	7
Optimization Method	Newton-Raphson
AIC	36378.62
SBC	36472.35
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-8) satisfied.	
---	--

Output 8.1.1 continued

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.001286	0.006746	296.67	<.0001
group 1	1	-1.491903	0.012383	-120.48	<.0001
group 2	0	0	.	.	.
x1	1	0.296633	0.004628	64.09	<.0001
x2	1	0.400643	0.004607	86.96	<.0001
x3	1	0.196278	0.004592	42.74	<.0001
x4	1	0.394971	0.004644	85.05	<.0001
x5	1	-0.299776	0.004508	-66.50	<.0001
x6	1	-0.497570	0.004809	-103.47	<.0001
x7	1	-0.295755	0.004510	-65.57	<.0001
Inf_Intercept	1	-0.969211	0.028573	-33.92	<.0001
Inf_z1	1	-0.609492	0.029128	-20.92	<.0001
Inf_z2	1	0.298572	0.027039	11.04	<.0001
Inf_z3	1	0.208062	0.026557	7.83	<.0001

The “Class Level Information” table shows that the classification variable Group has two levels. The “Model Fit Summary” table shows detailed information about the model and indicates that all 10,000 observations were used to fit the model. All parameter estimates in the “Parameter Estimates” table are highly significant and correspond to their theoretical values set during the data generating process.

References

- Cameron, A. C., and Trivedi, P. K. (1986). “Econometric Models Based on Count Data: Comparisons and Applications of Some Estimators and Tests.” *Journal of Applied Econometrics* 1:29–53.
- Cameron, A. C., and Trivedi, P. K. (1998). *Regression Analysis of Count Data*. Cambridge: Cambridge University Press.
- Conway, R. W., and Maxwell, W. L. (1962). “A Queuing Model with State Dependent Service Rates.” *Journal of Industrial Engineering* 12:132–136.
- Guikema, S. D., and Coffelt, J. P. (2008). “A Flexible Count Data Regression Model for Risk Analysis.” *Risk Analysis* 28:213–223.
- Lambert, D. (1992). “Zero-Inflated Poisson Regression with an Application to Defects in Manufacturing.” *Technometrics* 34:1–14.
- LaMotte, L. R. (1994). “A Note on the Role of Independence in t Statistics Constructed from Linear Statistics in Regression Models.” *American Statistician* 48:238–240.
- Long, J. S. (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: Sage Publications.

- Lord, D., Guikema, S. D., and Geedipally, S. R. (2008). “Application of the Conway-Maxwell-Poisson Generalized Linear Model for Analyzing Motor Vehicle Crashes.” *Accident Analysis and Prevention* 40:1123–1134.
- Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S., and Boatwright, P. (2005). “A Useful Distribution for Fitting Discrete Data: Revival of the Conway-Maxwell-Poisson Distribution.” *Journal of the Royal Statistical Society, Series C* 54:127–142.

Chapter 9

The CPANEL Procedure

Contents

Overview: CPANEL Procedure	340
PROC CPANEL Features	341
PROC CPANEL Compared with Other SAS Procedures	342
Using CAS Sessions and CAS Engine Librefs	343
Getting Started: CPANEL Procedure	344
Syntax: CPANEL Procedure	346
Functional Summary	346
PROC CPANEL Statement	348
BY Statement	349
CLASS Statement	349
COMPARE Statement	350
CORRELATED Statement	352
DISPLAY Statement	352
DISPLAYOUT Statement	353
ENDOGENOUS Statement	354
ID Statement	355
INSTRUMENTS Statement	356
MODEL Statement	357
OUTPUT Statement	366
PREDETERMINED Statement	367
RESTRICT Statement	367
TEST Statement	368
Details: CPANEL Procedure	369
Specifying the Input Data	369
Specifying the Regression Model	370
Missing Values	371
Unbalanced Data	371
Common Notation	371
Pooled Regression (POOLED Option)	372
Between-Groups Regression (BTWNG and BTWNT Options)	372
One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)	372
One-Way Fixed-Effects Model, First Differencing (FDONE Option)	373
Two-Way Fixed-Effects Model (FIXTWO Option)	374
One-Way Random-Effects Model (RANONE Option)	375
Two-Way Random-Effects Model (RANTWO Option)	378
Hausman-Taylor Estimation (HTAYLOR Option)	381

Amemiya-MaCurdy Estimation (AMACURDY Option)	383
Instrumental Variables Regression	383
Dynamic Panel Estimation (DYNDIFF and DYN SYS Options)	389
Poolability Test for Fixed Effects	399
Hausman Specification Tests	399
Restricted Estimation	399
Linear Hypothesis Testing	400
Heteroscedasticity-Corrected Covariance Matrices	401
Heteroscedasticity- and Autocorrelation-Consistent Covariance Matrices	404
R-Square	407
Displayed Output	407
ODS Table Names	410
Examples: CPANEL Procedure	412
Example 9.1: Airline Cost Data: Fixed Effects	412
Example 9.2: Analyzing Demand for Liquid Assets: Random Effects	416
Example 9.3: Panel Study of Income Dynamics (PSID): Hausman-Taylor Models	419
Example 9.4: Crime Rates in North Carolina: Instrumental Variables Regression	424
Example 9.5: Cigarette Sales Data: Dynamic Panel Estimation	428
References	434

Overview: CPANEL Procedure

The CPANEL procedure analyzes a class of linear econometric models that arise when time series and cross-sectional data are combined. This type of data is called *panel data*. Typical examples of panel data include observations over time on people, households, countries, or firms. For example, in the case of survey data on household income, the panel is created by repeatedly surveying the same households over many years. The individual entities that are followed over time are called *cross sections*.

Regression models for panel data are characterized by an error structure that can be divided into a cross-sectional component, a time component, and an observation-level component. Panel data models can be categorized by the exact structure of the error term and how it interacts with the regression equation. The CPANEL procedure supports the following models:

- one-way and two-way models
- fixed-effects, random-effects, and hybrid models
- instrumental variables (IV) models
- dynamic panel models

A one-way model depends only on the cross section to which the observation belongs. A two-way model depends on both the cross section and the time period to which the observation belongs. The models are called fixed-effects models if the effects are nonrandom and are called random-effects models otherwise.

If the effects are fixed, the models are essentially regression models with dummy variables that correspond to the effects. For fixed-effects models, ordinary least squares (OLS) estimation, with some facility for absorbing the dummy variables, is the best linear unbiased estimator. Random-effects models use a two-stage approach. In the first stage, variance components are estimated by your choice of the methods described by Swamy and Arora (1972); Fuller and Battese (1974); Wansbeek and Kapteyn (1989); Wallace and Hussain (1969); Nerlove (1971). In the second stage, variance components are used to standardize the data, and then OLS regression is performed.

Random-effects models are more efficient than fixed-effects models, and they can estimate effects for variables that do not vary within cross sections. The cost of these added features is that random-effects models carry much more stringent assumptions than their fixed-effects counterparts. The CPANEL procedure also supports models that blend the desirable features of random and fixed effects. These hybrid models are described by Hausman and Taylor (1981) and Amemiya and MaCurdy (1986).

Instrumental variables (IV) models are models that introduce auxiliary regression variables to deal with endogenous regressors—regressors that are correlated with the residual errors. The CPANEL procedure supports IV generalizations of the pooled regression model, the between-groups regression model, the one-way fixed-effects model, the one-way random-effects model, the two-way fixed-effects model, and the two-way random-effects model. For the one-way random-effects model, PROC CPANEL uses methods that were developed by Baltagi (1981). PROC CPANEL can also estimate these models by using the efficient generalized method of moments (GMM).

Dynamic panel models are autoregression models that include lagged versions of the dependent variable as regressors. Examples of such models can be found in Arellano and Bond (1991) and Blundell and Bond (1998). The CPANEL procedure fits these models by using the generalized method of moments (GMM). You can perform GMM on either the difference equations or the full system of difference and level equations.

PROC CPANEL Features

The CPANEL procedure fits panel-data regression models by using various techniques, and does the following:

- provides an **ID** statement for specifying the cross section and time variables
- provides flexible model-building syntax that supports continuous regressors, classification (**CLASS**) variables, interactions, and nested effects
- fits one-way and two-way fixed-effects models
- fits one-way and two-way random-effects models
- fits Hausman and Taylor (1981) models
- fits Amemiya and MaCurdy (1986) models
- fits instrumental variables (IV) regression models
- fits dynamic panel models
- enables you to fit multiple models by using multiple estimators, all within one procedure call

- provides a **COMPARE** statement for producing customizable comparisons of various models and estimators
- provides a **TEST** statement for performing linear tests of hypotheses
- provides a **RESTRICT** statement for imposing linear parameter constraints
- provides an **OUTPUT** statement for producing a data table that contains predictions and residuals

Because the CPANEL procedure runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on CAS
- exploits all the available cores and concurrent threads. For information about how PROC CPANEL uses threads, see the section “[Multithreading](#)” on page 65 in Chapter 3, “[Shared Concepts](#).”

PROC CPANEL Compared with Other SAS Procedures

The CPANEL procedure models panel data in a way that is comparable to how the HPPANEL and PANEL procedures in SAS/ETS software do this.

PROC CPANEL Compared with the HPPANEL Procedure

The functionality of the CPANEL procedure closely resembles that of the HPPANEL procedure, which is a high-performance procedure. The CPANEL procedure is the next generation of the HPPANEL procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads.

The CPANEL provides all the functionality of the HPPANEL procedure, and also provides the following:

- instrumental variable (IV) regression estimators
- dynamic panel estimators, as described by Arellano and Bond (1991) and Blundell and Bond (1998)
- Hausman and Taylor (1981) and Amemiya and MaCurdy (1986) estimators, which are hybrids of fixed-effects and random-effects estimators
- estimation of the fixed-effects model by first differencing (FD)
- Swamy and Arora (1972) estimates of variance components
- tables that compare parameters from multiple models and multiple estimation methods
- the option to specify a character variable as the cross section ID variable
- the option to not specify a time series ID variable when time is not relevant to the estimation

PROC CPANEL Compared with the PANEL Procedure

The CPANEL procedure provides the following functionality not found in PROC PANEL:

- instrumental variables (IV) regression estimators
- Swamy and Arora (1972) estimates of variance components
- support for estimation when the data contain cross sections that consist of only one observation
- the option to specify a character variable as the cross section ID variable
- the option to not specify a time-series ID variable when time is not relevant to the estimation

The PANEL procedure parameterizes the fixed-effects model by using the last ordered cross section and the latest time point as reference categories. The CPANEL procedure uses a mean-centered parameterization.

The PANEL procedure supports the Da Silva (1975) method for serial correlation, and the Parks (1967) method for first-order autocorrelation. These two methods are not provided in this release of the CPANEL procedure.

The PANEL procedure offers a wide variety of postfitting analyses, such as model specification tests and tests for unit root. The PANEL procedure is also able to construct utility data sets that contain lagged variables and data reformatted from wide form to long form. This release of the CPANEL procedure is limited in postfitting and data-generating functionality, because the focus for large data sets is primarily on model fitting and comparison.

The CPANEL procedure is specifically designed to operate in SAS Viya and performs computations in multiple threads. The PANEL procedure executes in a single thread on a single machine.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CPANEL Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “[Using CAS Sessions and CAS Engine Librefs](#)” on page 12 and “[Loading a SAS Data Set onto a CAS Server](#)” on page 13 in Chapter 3, “[Shared Concepts](#).”

The following DATA step creates the data table `Electricity` from the cost function data in Greene (1990). This DATA step assumes that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref. The variable `Production` is the log of output in millions of kilowatt-hours, and the variable `Cost` is the log of cost in millions of dollars.

```
data mycas.Electricity;
  input firm year production cost @@;
datalines;
1 1955 5.36598 1.14867 1 1960 6.03787 1.45185
1 1965 6.37673 1.52257 1 1970 6.93245 1.76627
2 1955 6.54535 1.35041 2 1960 6.69827 1.71109
2 1965 7.40245 2.09519 2 1970 7.82644 2.39480
3 1955 8.07153 2.94628 3 1960 8.47679 3.25967
3 1965 8.66923 3.47952 3 1970 9.13508 3.71795
4 1955 8.64259 3.56187 4 1960 8.93748 3.93400
4 1965 9.23073 4.11161 4 1970 9.52530 4.35523
5 1955 8.69951 3.50116 5 1960 9.01457 3.68998
5 1965 9.04594 3.76410 5 1970 9.21074 4.05573
6 1955 9.37552 4.29114 6 1960 9.65188 4.59356
6 1965 10.21163 4.93361 6 1970 10.34039 5.25520
;
```

Consider the model

$$C_{it} = \beta_0 + \beta_1 P_{it} + v_i + e_{it} \text{ for } i = 1, \dots, N \text{ and } t = 1, \dots, T$$

where C_{it} represents cost, P_{it} represents production, v_i is the cross-sectional error component, and e_{it} is the error variance component.

If you assume that the cross-sectional effects are random, five possible estimators are available for the variance components. The VCOMP=FB option in the following statements uses the Fuller and Battese (1974) estimator to fit the model:

```
proc cpanel data = mycas.Electricity;
  id firm year;
  model cost = production / ranone vcomp = fb;
run;
```

The output of these statements is shown in [Output 9.1](#).

Figure 9.1 One-Way Random-Effects Estimation Results

The CPANEL Procedure
One-Way Random Effects (RanOne)
Fuller and Battese Variance Components
Dependent Variable: cost

Model Description			
Estimation Method		RanOne	
Variance Estimation		Model Based	
Data Set		ELECTRICITY	
Number of Observations		24	
Number of Cross Sections		6	
Time Series Length		4	

Fit Statistics			
SSE	0.4143	DFE	22
MSE	0.0188	Root MSE	0.1372
R-Square	0.9164		

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.04109	0.202707
Error	0.015533	0.124632

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
1	1	9.08	0.0026

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	-3.27307	0.4277	-7.65	<.0001
production	1	0.779469	0.0502	15.53	<.0001

Printed first is a report that provides the estimation method and various data counts. Fit statistics and variance components estimates are printed next. A Hausman specification test compares this model to its fixed-effects counterpart. Finally, the table of regression parameter estimates shows the estimates, standard errors, and t tests.

Syntax: CPANEL Procedure

The following statements are available in the CPANEL procedure:

```

PROC CPANEL < options > ;
  BY variables ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  COMPARE < model-list > < / options > ;
  CORRELATED effects ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  ENDOGENOUS effects < / option > ;
  ID cross-section-id < time-series-id > ;
  INSTRUMENTS effects < / option > ;
  MODEL response = < effects > < / options > ;
  OUTPUT OUT=CAS-libref.data-table < options > ;
  PREDETERMINED effects < / option > ;
  RESTRICT equation1 < , equation2... > ;
  TEST equation1 < , equation2... > ;

```

Functional Summary

Table 9.1 summarizes the statements and options that the CPANEL procedure supports.

Table 9.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CPANEL	DATA=
Prevents partitioning of data by cross sections	PROC CPANEL	NOPART
Specifies the name of an output SAS data table	OUTPUT	OUT=
Variable Role Statements		
Specifies BY-group processing	BY	
Specifies the classification variables	CLASS	
Specifies the cross section and time ID variables	ID	
Declares correlated variables or effects	CORRELATED	
Declares endogenous variables or effects	ENDOGENOUS	
Declares instrumental variables or effects	INSTRUMENTS	
Declares predetermined variables or effects	PREDETERMINED	
Printing Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	

Table 9.1 *continued*

Description	Statement	Option
Prints correlations of the estimates	MODEL	CORRB
Prints covariances of the estimates	MODEL	COVB
Suppress parameter labels from output tables	MODEL	NOLABEL
Suppresses printed output	PROC CPANEL	NOPRINT
Prints all available output	PROC CPANEL	PRINTALL
Prints fixed effects	MODEL	PRINTFIXED
Model Estimation Technique Options		
Specifies the Amemiya-MaCurdy model	MODEL	AMACURDY
Specifies the between-groups model	MODEL	BTWNG
Specifies the between-time-periods model	MODEL	BTWNT
Specifies the dynamic panel model via first-differencing GMM	MODEL	DYNDIFF
Specifies the dynamic panel model via system GMM	MODEL	DYNSYS
Specifies the one-way fixed-effects model via first-differencing	MODEL	FDONE
Specifies the one-way fixed-effects model	MODEL	FIXONE
Specifies the one-way fixed-effects model with respect to time	MODEL	FIXONETIME
Specifies the two-way fixed-effects model	MODEL	FIXTWO
Specifies the Hausman-Taylor model	MODEL	HTAYLOR
Specifies the between-groups model with instrumental variables	MODEL	IVBTWNG
Specifies the one-way fixed-effects model with instrumental variables	MODEL	IVFIXONE
Specifies the one-way fixed-effects model for time effects with instrumental variables	MODEL	IVFIXONETIME
Specifies the two-way fixed-effects model with instrumental variables	MODEL	IVFIXTWO
Specifies the generalized two-stage least squares model with instrumental variables	MODEL	IVG2SLS
Specifies the pooled regression model with instrumental variables	MODEL	IVPOOLED
Specifies the one-way random-effects model with instrumental variables	MODEL	IVRANONE
Specifies the two-way random-effects model with instrumental variables	MODEL	IVRANTWO
Specifies the pooled regression model	MODEL	POOLED
Specifies the one-way random-effects model	MODEL	RANONE
Specifies the two-way random-effects model	MODEL	RANTWO
Model Estimation Control Options		
Sets the number of AR(m) tests	MODEL	ARTESTS=

Table 9.1 *continued*

Description	Statement	Option
Specifies bias-corrected variances for two-step GMM	MODEL	BIASCORRECTED
Requests the clustered HCCME estimator for the covariance matrix	MODEL	CLUSTER
Sets the number of dependent variable lags	MODEL	DLAGS=
Specifies the generalized inverse type	MODEL	GINV=
Specifies one-step or two-step GMM	MODEL	GMM=
Requests the HAC estimator for the variance-covariance matrix	MODEL	HAC
Requests the HCCME estimator for the covariance matrix	MODEL	HCCME=
Limits the number of GMM instruments	MODEL	MAXBAND=
Suppresses the intercept term	MODEL	NOINT
Specifies robust variances estimator	MODEL	ROBUST
Specifies the method for the variance components estimator	MODEL	VCOMP=
Tests and Restrictions		
Specifies linear equality restrictions on the parameters	RESTRICT	
Performs tests of linear hypotheses	TEST	WALD, LM, LR
Model Comparison Statements		
Create tables that display side-by-side model comparisons	COMPARE	

PROC CPANEL Statement

PROC CPANEL *<options>* ;

The PROC CPANEL statement invokes the procedure. You can specify the following *options*:

DATA=*CAS-libref.data-table*

names the input data table for PROC CPANEL to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 343.

data-table specifies the name of the input data table.

NOPART

prevents repartitioning of the data. Normally, the CPANEL procedure redistributes the data table so that observations for each cross section are not divided among multiple machines. Repartitioning the data makes model computations more efficient but has a cost in memory and computing cycles that are associated with redistribution. Specifying the NOPART option leaves the data distributed as they are, at the expense of less efficient model computations.

NOPRINT

suppresses all printed output.

PRINTALL

prints all available output.

In addition, any of the following MODEL statement options can be specified in the PROC CPANEL statement: **CORRB**, **COVB**, **NOLABEL**, and **PRINTFIXED**. When specified in the PROC CPANEL statement, these options apply globally to every MODEL statement. For a complete description of each of these options, see the section “MODEL Statement” on page 357.

BY Statement

BY *variables* ;

You can specify a BY statement in PROC CPANEL to obtain separate analyses of observations in groups that are defined by the values of the BY variables. If you specify more than one BY statement, only the last one specified is used. For more information, see the discussion of BY-group processing in *SAS Language Reference: Concepts*.

CLASS Statement

CLASS *variable* < (*options*) > . . . < *variable* < (*options*) > > < / *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. Table 9.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 14 in Chapter 3, “Shared Concepts.”

Table 9.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable

COMPARE Statement

COMPARE <model-list> </options> ;

The COMPARE statement creates tables of side-by-side comparisons of parameter estimates and other model statistics. You can fit multiple models simultaneously by specifying multiple MODEL statements, and you can specify a COMPARE statement to create tables that compare the models.

The COMPARE statement creates two tables: the first table compares model fit statistics such as R-square and mean square error; the second table compares regression coefficients, their standard errors, and (optionally) *t* tests.

By default, comparison tables are created for all fitted models, but you can use the optional *model-list* to limit the comparison to a subset of the fitted models. The *model-list* consists of a set of model labels, as specified in the MODEL statement; for more information, see the section “MODEL Statement” on page 357. If a model does not have a label, you refer to it generically as “Model *i*,” where the corresponding model is the *i*th MODEL statement specified.

You can specify one or more COMPARE statements. The following code illustrates the use of the COMPARE statement:

```
proc cpanel data = mycas.a;
  id csid tsid;
  mod_one: model y = x1 x2 x3 / fixone;
  second: y = x1 x2 / fixone;
  model y = x1 x2 x3 x4 / ranone;
  compare;
  compare "second" "Model 3";
run;
```

The first COMPARE statement compares all three fitted models. The second COMPARE statement uses the generic “Model 3” to identify the third model and compares it to the second model.

You can specify the following *options* after a slash (/):

MSTAT (*mstat-list*)

specifies a list of model fit statistics to be displayed. A set of statistics is displayed by default, but you can use this option to specify a custom set of model statistics.

The *mstat-list* can contain one or more of the following keywords:

ALL	displays all model fit statistics. Not all statistics are appropriate for all models, and thus not always calculated. A blank cell in the table indicates that the statistic is not appropriate for that model.
DFE	displays the error degrees of freedom. This statistic is displayed by default.
F	displays the <i>F</i> statistic of the overall test for no fixed effects.
FNUMDF	displays the numerator degrees of freedom of the overall test for no fixed effects.
FDENDF	displays the denominator degrees of freedom of the overall test for no fixed effects.
M	displays the Hausman test <i>m</i> statistic.
MDF	displays the Hausman test degrees of freedom.

MSE	displays the model mean square error. This statistic is displayed by default.
NCS	displays the number of cross sections. This statistic is displayed by default.
NOBS	displays the number of observations. This statistic is displayed by default.
NONE	suppresses the table of model fit statistics when specified alone, and is ignored when specified with other options.
NTS	displays the maximum length of the time series. This statistic is displayed by default.
PROBF	displays the significance level of the overall test for no fixed effects.
PROBM	displays the significance level of the Hausman test.
RMSE	displays the model root mean square error.
RSQUARE	displays the model R-square fit statistic. This statistic is displayed by default.
SSE	displays the model sum of squares.
VARCS	displays the variance component that corresponds to cross sections in random-effects models.
VARERR	displays the variance component that corresponds to error in random-effects models.
VARTS	displays the variance component that corresponds to time series in random-effects models.

OUTPARM=CAS-libref.data-table

names an output data table to contain the data from the comparison table for parameter estimates, standard errors, and *t* tests. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 343.

OUTSTAT=CAS-libref.data-table

names an output data table to contain the data from the comparison table for model fit statistics, such as R-square and mean square error. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 343.

PSTAT (pstat-list)

specifies a list of parameter statistics to be displayed. By default, estimated regression coefficients and their standard errors are displayed. Use this option to specify a custom set of parameter statistics.

pstat-list can contain one or more of the following keywords:

ALL	displays all parameter statistics.
ESTIMATE	displays the estimated regression coefficient. This statistic is displayed by default.
NONE	suppresses the table of parameter statistics when specified alone, and is ignored when specified with other options.
STDERR	displays the standard error. This statistic is displayed by default.

PROBT	displays the significance level of the t test.
T	displays the t statistic.

For a demonstration of the COMPARE statement, see [Example 9.2](#).

CORRELATED Statement

CORRELATED *effects* ;

The CORRELATED statement selects the subset of effects from the [MODEL](#) statement that are correlated with the individual (or cross-sectional) error term. This statement is valid only when you specify the [AMACURDY](#) or [HTAYLOR](#) model estimation method in the MODEL statement.

If you specify an *effect* in the CORRELATED statement but not in the MODEL statement, then the *effect* is added to the list of model effects and treated as correlated with the cross-sectional error term. If you have multiple MODEL statements, then each CORRELATED statement is associated with the MODEL statement that immediately precedes it.

The following statements fit a Hausman and Taylor (1981) model with model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are correlated with the cross-sectional (firm-level) error term.

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / htaylor;
  correlated X2 Z2;
run;
```

You can also specify classification variables, interactions, and nested effects. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 410. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the CAS output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 410. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

ENDOGENOUS Statement

ENDOGENOUS *effects* </ *option* > ;

The ENDOGENOUS statement declares a set of *effects* that are treated as endogenous during estimation, and is valid only when you perform either instrumental variables regression or dynamic panel estimation.

Instrumental Variables (IV) Regression

You perform instrumental variables (IV) regression by specifying one or more of the **IVBTWNG**, **IVFIXONE**, **IVFIXONETIME**, **IVFIXTWO**, **IVG2SLS**, **IVPOOLED**, **IVRANONE**, and **IVRANTWO** options in the **MODEL** statement.

When you use the ENDOGENOUS statement for IV regression, it selects the subset of effects from the **MODEL** statement that are endogenous—that is, the effects that are correlated with the observation-level error term.

If you specify an *effect* in the ENDOGENOUS statement but not in the **MODEL** statement, then the *effect* is added to the list of model effects and treated as endogenous. If you have multiple **MODEL** statements, then each ENDOGENOUS statement is associated with the **MODEL** statement that immediately precedes it.

The ENDOGENOUS statement is used in conjunction with the **INSTRUMENTS** statement to perform IV regression. The following statements perform IV fixed-effects regression with the model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are endogenous, and G1 and G2 are external instruments.

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

Dynamic Panel Estimation

You perform dynamic panel estimation by specifying the **DYNDIFF** or **DYNSYS** option in the **MODEL** statement.

When you use the **ENDOGENOUS** statement in dynamic panel estimation, it specifies a set of variables whose instruments follow a prescribed form that is appropriate for endogenous regressors.

If you have multiple **MODEL** statements, then each **ENDOGENOUS** statement is associated with the **MODEL** statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the endogenous instruments. You can specify the following values:

- BOTH** includes the endogenous instruments in both the difference and level equations.
- DIFF** includes the endogenous instruments only in the difference equations.
- LEVEL** includes the endogenous instruments only in the level equations.

By default, **EQ=BOTH**.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable **Sales** and standard instruments for the exogenous variable **Price**. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mycas.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  instruments Price / eq = diff;
run;
```

For both IV regression and dynamic panel estimation, you can also specify classification variables, interactions, and nested effects in the **ENDOGENOUS** statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

ID Statement

ID *cross-section-id* < *time-series-id* > ;

The **ID** statement specifies variables in the input data table that identify the cross section and time period for each observation. Sometimes estimation does not require a time variable, in which case specifying the *time-series-id* variable is optional.

INSTRUMENTS Statement

INSTRUMENTS *effects* *</ option>* ;

The INSTRUMENTS statement specifies instrumental variables and effects. It is valid only when you perform either instrumental variables regression or dynamic panel estimation.

Instrumental Variables (IV) Regression

You perform instrumental variables (IV) regression by specifying one or more of the [IVBTWNG](#), [IVFIXONE](#), [IVFIXONETIME](#), [IVFIXTWO](#), [IVG2SLS](#), [IVPOOLED](#), [IVRANONE](#), and [IVRANTWO](#) options in the MODEL statement.

The effects that you specify in the INSTRUMENTS statement are auxiliary effects external to those that you specify in the MODEL statement. If you have multiple MODEL statements, then each INSTRUMENTS statement is associated with the MODEL statement that immediately precedes it.

The INSTRUMENTS statement is used in conjunction with the ENDOGENOUS statement to perform IV regression. The following statements perform IV fixed-effects regression with the model effects X1, X2, Z1, and Z2. The effects X2 and Z2 are endogenous, and G1 and G2 are external instruments.

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

Dynamic Panel Estimation

You perform dynamic panel estimation by specifying the [DYNDIFF](#) or [DYN SYS](#) options in the MODEL statement.

When you use the INSTRUMENTS statement in dynamic panel estimation, it specifies a set of variables to be included as standard instruments.

If you have multiple MODEL statements, then each INSTRUMENTS statement is associated with the MODEL statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the standard instruments. You can specify the following values:

- BOTH** includes the standard instruments in both the difference and level equations.
- DIFF** includes the standard instruments only in the difference equations.
- LEVEL** includes the standard instruments only in the level equations.

By default, EQ=BOTH.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable Sales and standard instruments for the exogenous variable Price. Both sets of instruments are included only in the difference equations.

```

proc cpanel data = mycas.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  instruments Price / eq = diff;
run;

```

For both IV regression and dynamic panel estimation, you can also specify classification variables, interactions, and nested effects in the INSTRUMENTS statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

MODEL Statement

MODEL *response* = < *effects* > < / *options* > ;

The MODEL statement specifies the regression model, the error structure that is assumed for the regression residuals, and the estimation technique to be used. The response variable (*response*) on the left side of the equal sign is regressed on the independent variables (*effects*), which are listed after the equal sign. You can specify any number of MODEL statements. For each MODEL statement, you can specify only one *response*.

For information about constructing the model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

You can label models. Model labels are used in the printed output to identify the results for different models. If you do not specify a label, the model is referred to by numerical order wherever necessary. You can label models by prefixing the MODEL statement by a label followed by a colon as follows:

label: **MODEL** ...;

The MODEL statement supports many options, some more specific than others. [Table 9.3](#) summarizes the *options* available in the MODEL statement. These are subsequently discussed in detail in the order in which they are presented in the table.

Table 9.3 Summary of MODEL Statement Options

Option	Description
Estimation Technique Options	
AMACURDY	Fits a one-way model by using the Amemiya-MaCurdy estimator
BTWNG	Fits the between-groups model
BTWNT	Fits the between-time-periods model
DYNDIFF	Fits a dynamic panel model by using first-differences GMM
DYNSYS	Fits a dynamic panel model by using system GMM
FDONE	Fits a one-way model by using first differences
FIXONE	Fits a one-way fixed-effects model
FIXONETIME	Fits a one-way fixed-effects model for time effects
FIXTWO	Fits a two-way fixed-effects model

Table 9.3 *continued*

Option	Description
HTAYLOR	Fits a one-way model by using the Hausman-Taylor estimator
IVBTWNG	Fits a between-groups regression model by using instrumental variables
IVFIXONE	Fits a one-way fixed-effects model by using instrumental variables
IVFIXONETIME	Fits a one-way time fixed-effects model for time effects by using instrumental variables
IVFIXTWO	Fits a two-way fixed-effects model by using instrumental variables
IVG2SLS	Fits a generalized two-stage least squares model by using instrumental variables
IVPOOLED	Fits the pooled regression model by using instrumental variables
IVRANONE	Fits a one-way random-effects model by using instrumental variables
IVRANTWO	Fits a two-way random-effects model by using instrumental variables
POOLED	Fits the pooled regression model
RANONE	Fits a one-way random-effects model
RANTWO	Fits a two-way random-effects model
Estimation Control Options	
BIASCORRECTED	Computes bias-corrected two-step GMM variances for instrumental variables models and dynamic panel models
GMM=	Specifies one-step or two-step GMM for instrumental variables models and dynamic panel models
NOINT	Suppresses the intercept
VCOMP=	Specifies the type of variance component estimation for random-effects estimation
Dynamic Panel Estimation Options	
ARTESTS=	Specifies the maximum lag of $AR(m)$ tests
DLAGS=	Specifies the number of dependent variable lags to include
GINV=	Specifies the generalized inverse method
MAXBAND=	Limits the number of instruments
Alternative Variances Options	
CLUSTER	Corrects covariance for intracluster correlation
HAC	Specifies a heteroscedasticity- and autocorrelation-consistent (HAC) covariance

Table 9.3 *continued*

Option	Description
HCCME=	Specifies a heteroscedasticity-corrected covariance matrix estimator (HCCME)
MAXBAND=	Limits the number of instruments
ROBUST	Computes robust variances
Printed Output Options	
CORRB	Prints the parameter correlation matrix
COVB	Prints the parameter covariance matrix
NOLABEL	Suppresses variable labels
PRINTFIXED	Estimates and prints the fixed effects

You can specify the following *options* after a slash (/).

Estimation Technique Options

These options specify the assumed error structure and estimation method. You can specify more than one option, in which case the analysis is repeated for each. The default is FIXONE (one-way fixed effects).

All estimation methods are described in detail in the section “[Details: CPANEL Procedure](#)” on page 369.

AMACURDY

requests Amemiya-MaCurdy estimation for a model that has correlated individual (cross-sectional) effects. You specify the correlated effects by using the CORRELATED statement.

BTWNG

estimates a between-groups model.

BTWNT

estimates a between-time-periods model.

DYNDIFF

estimates a dynamic panel by using the generalized method of moments (GMM) on equations that are formed by first differencing.

DYNSYS

estimates a dynamic panel by using GMM on the system that combines first-differenced equations and level equations.

FDONE

estimates a one-way model by using first-differenced methods.

FIXONE

estimates a one-way fixed-effects model that corresponds to cross-sectional effects only.

FIXONETIME

estimates a one-way fixed-effects model that corresponds to time effects only.

FIXTWO

estimates a two-way fixed-effects model.

HTAYLOR

requests Hausman-Taylor estimation for a model that has correlated individual (cross-sectional) effects. You specify the correlated effects by using the `CORRELATED` statement.

IVBTWNG

requests instrumental variables regression for between-groups estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXONE

requests instrumental variables regression for one-way fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXONETIME

requests instrumental variables regression for one-way time fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVFIXTWO

requests instrumental variables regression for two-way fixed-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVG2SLS

requests instrumental variables regression for generalized two-stage least squares estimation for a model that has endogenous effects. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVPOOLED

requests instrumental variables regression for pooled regression. You specify the endogenous effects by using the `ENDOGENOUS` statement, and you specify external instruments by using the `INSTRUMENTS` statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVRANONE

requests instrumental variables regression for one-way random-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the **ENDOGENOUS** statement, and you specify external instruments by using the **INSTRUMENTS** statement. You can choose to use two-stage least squares or GMM to estimate the model.

IVRANTWO

requests instrumental variables regression for two-way random-effects estimation for a model that has endogenous effects. You specify the endogenous effects by using the **ENDOGENOUS** statement, and you specify external instruments by using the **INSTRUMENTS** statement. You can choose to use two-stage least squares or GMM to estimate the model.

POOLED

estimates a pooled (OLS) model.

RANONE

estimates a one-way random-effects model.

RANTWO

estimates a two-way random-effects model.

Estimation Control Options

These options define parameters that control the estimation and can be specific to the chosen technique (for example, how to estimate variance components in a random-effects model).

BIASCORRECTED

computes the bias-corrected covariance matrix of the two-step generalized method of moments (GMM) estimator. This statement is valid only when you perform either dynamic panel estimation or instrumental variables regression.

CLUSTER

specifies the cluster correction for the covariance matrix. You can specify this option when you specify **HCCME=0, 1, 2, or 3**.

GMM=ONESTEP | TWOSTEP

specifies the number of GMM stages to use in the estimation. This statement is valid only when you perform either dynamic panel estimation or instrumental variables regression.

You can specify the following values:

ONESTEP	uses one-step GMM, which is computationally simple but dependent on model assumptions.
TWOSTEP	uses two-step GMM, which is computationally intensive but more robust to violations of model assumptions.

By default, **GMM=ONESTEP**.

The following statements perform two-step GMM in a dynamic panel model, which is requested by the **DYNDIFF** option in the **MODEL** statement. You can use similar statements for a dynamic panel estimation that uses a full system of difference and level equations by specifying the **DYNSYS** option

instead of the DYNDIFF option in the MODEL statement. The endogenous variable is *Sales*, and the GMM-style instruments is the predetermined variable *Price*. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mycas.a;
  id firm year;
  model Sales = Price / dyndiff GMM = TWOSTEP;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
run;
```

The following statements perform two-step GMM in an instrumental variables model. The endogenous variables are *X2* and *Z2*, and the external instruments are *G1* and *G2*.

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone GMM = TWOSTEP;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

The IVFIXONE option in the MODEL statement requests IV one-way fixed-effects estimation. You can request other types of IV estimation by specifying IVBTWNG, IVFIXONETIME, IVFIXTWO, IVG2SLS, IVPOOLED, IVRANONE, or IVRANTWO instead of IVFIXONE in the MODEL statement.

HAC <(options)>

specifies the heteroscedasticity- and autocorrelation-consistent (HAC) covariance matrix estimator. This option is not available for between-groups or between-time-periods models, and cannot be combined with the HCCME= option.

You can specify the following *options* within parentheses and separated by spaces:

ADJUSTDF

makes a small-sample adjustment to the degrees of freedom in the covariance calculation.

BANDWIDTH=*number* | *method*

specifies the fixed bandwidth value or bandwidth selection method to be used in the kernel function. You can specify either a fixed value (*number*) or one of the *methods* listed after *number*.

number

specifies a fixed value of the bandwidth parameter.

ANDREWS | **AN**

specifies the Andrews (1991) bandwidth selection method.

NEWWEYWEST<(C=number)> | NW <(C=number)>

specifies the bandwidth selection method of Newey and West (1994) You can also specify *C=number* to calculate the lag selection parameter; by default, C=12.

SAMPLESIZE<(options)> | SS<(options)>

calculates the bandwidth according to the following equation based on the sample size,

$$b = \gamma T^r + c$$

where *b* is the bandwidth parameter; *T* is the sample size; and γ , *r*, and *c* are values that you specify using the following *options* within parentheses and separated by spaces:

CONSTANT=number

specifies the constant *c* in the equation. By default, CONSTANT=0.5.

GAMMA=number

specifies the coefficient γ in the equation. By default, GAMMA=0.75.

INTSS

specifies that the bandwidth parameter be an integer; that is, $b = \lfloor \gamma T^r + c \rfloor$, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to *x*.

RATE=number

specifies the growth rate *r* in the equation. By default, RATE=0.3333.

By default, BANDWIDTH=ANDREWS.

KERNEL=BARTLETT | PARZEN | QS | TH | TRUNCATED

specifies the type of kernel function. You can specify the following values:

BARTLETT	specifies the Bartlett kernel function.
PARZEN	specifies the Parzen kernel function.
QS	specifies the quadratic spectral kernel function.
TH	specifies the Tukey-Hanning kernel function.
TRUNCATED	specifies the truncated kernel function.

By default, KERNEL=QS.

KERNELLB=number

specifies the lower bound of the kernel weight value. Any kernel weight less than *number* is regarded as 0, which accelerates the calculation in large samples, especially for the quadratic spectral kernel function. By default, KERNELLB=0.

PREWHITENING

requests prewhitening in the covariance calculation.

The following statements perform one-way fixed-effects estimation by using the Newey-West bandwidth and a Bartlett kernel based on a prewhitened gradient vector:

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone gmm=twostep
  hac(bandwidth=neweywest(c=12) kernel=bartlett prewhitening);
run;
```

HCCME=0 | 1 | 2 | 3

specifies the type of adjustment to the HCCME covariance matrix. You can specify the values 0 through 3, which indicate the type of covariance adjustment.

NOINT

suppresses the intercept parameter from the model.

ROBUST

computes the robust covariance matrix. This option can be applied to dynamic panel models, instrumental variables regression models, and static panel models. For dynamic panel models and instrumental variables regression models, the option provides heteroscedasticity-corrected standard errors according to the GMM estimation method. The following statements perform IV fixed-effects regression with robust standard errors:

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / ivfixone gmm=twostep robust;
  endogenous X2 Z2;
  instruments G1 G2;
run;
```

For static panel models, this option produces cross-sectional clustered standard errors when HCCME=0. The following statements perform one-way fixed-effects regression with clustered standard errors:

```
proc cpanel data = mycas.a;
  id firm year;
  model Y = X1 X2 Z1 Z2 / fixone robust;
run;
```

VCOMP=FB | NL | SA | WH | WK

specifies the type of variance component estimate to use. You can specify the following values:

- FB** uses the Fuller-Battese method.
- NL** uses the Nerlove method.
- SA** uses the Swamy-Arora method.
- WH** uses the Wallace-Hussain method.
- WK** uses the Wansbeek-Kapteyn method.

By default, VCOMP=SA.

Dynamic Panel Estimation Options

These options are specific to dynamic panel estimation, which you obtain by specifying the DYNDIFF or DYN SYS option in the MODEL statement.

ARTESTS=integer

specifies the maximum order of the test for the presence of autoregression (AR) effects. By default, ARTESTS=2.

DLAGS=integer

specifies the number of dependent-variable lags to use as regressors. By default, DLAGS=1.

GINV=G2 | G4

specifies what type of generalized inverse to use. You can specify the following values:

G2 uses the G2 generalized inverse.

G4 uses the G4 generalized inverse.

The difference between G2 and G4 becomes evident when you invert singular matrices. The G2 generalized inverse drops rows and columns from singular matrices to produce a viable inverse. The G4 inverse, on the other hand, is the Moore-Penrose generalized inverse, which averages the variance effects between collinear rows. The G4 inverse is usually more stable, but it is computationally intensive. By default, GINV=G2. If you have trouble reproducing published results, often the solution is to switch to GINV=G4.

MAXBAND=integer

if specified, sets the maximum number of GMM-style instruments per observation, for each variable. Because the number of GMM-style instruments grows quadratically with the number of time periods, this option makes estimation more feasible when you have many time periods.

Printed Output Options

These options alter how results are presented.

CORRB

prints the matrix of estimated correlations between the parameter estimates.

COVB

prints the matrix of estimated covariances between the parameter estimates.

NOLABEL

suppresses variable labels from the printed output.

PRINTFIXED

estimates and prints the fixed effects in models where they would normally be absorbed within the estimation.

OUTPUT Statement

OUTPUT OUT=*CAS-libref.data-table* < options > ;

The OUTPUT statement creates a data table that contains observationwise statistics that PROC CPANEL computes after fitting the model. In order to avoid data duplication for large data tables, only the identification and dependent variables are included in the output data table unless you specify additional variables in the **COPYVAR=** option. To avoid further data duplication, the OUTPUT statement is allowed only with MODEL statements that request a single estimation. If you have multiple MODEL statements, then each OUTPUT statement is associated with the MODEL statement that immediately precedes it.

You must specify the following option:

OUT=*CAS-libref.data-table*

names the output data table for PROC CPANEL to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>CAS-libref</i> , see the section “ Using CAS Sessions and CAS Engine Librefs ” on page 343.
-------------------	---

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

You can also specify the following options:

COPYVAR=*variable*

COPYVARS=(*variables*)

transfers one or more *variables* from the input data table to the output data table.

PREDICTED=*name*

P=*name*

writes the predicted values to the output data table.

RESIDUAL=*name*

R=*name*

writes the residuals to the output data table.

PREDETERMINED Statement

PREDETERMINED *effects* < / *option* > ;

The PREDETERMINED statement specifies a set of variables whose instruments follow a prescribed form that is appropriate for predetermined regressors and is valid only when you perform dynamic panel estimation by specifying the **DYNDIFF** or **DYNSYS** option in the MODEL statement.

If you have multiple MODEL statements, then each PREDETERMINED statement is associated with the MODEL statement that immediately precedes it.

You can specify the following *option* after a slash (/):

EQ=BOTH | DIFF | LEVEL

specifies which equations include the predetermined instruments. You can specify the following values:

BOTH includes the predetermined instruments in both the difference and level equations.

DIFF includes the predetermined instruments only in the difference equations.

LEVEL includes the predetermined instruments only in the level equations.

By default, EQ=BOTH.

The following statements perform dynamic panel estimation by using GMM-style instruments for the endogenous variable Sales and GMM-style instruments for the predetermined variable Price. Both sets of instruments are included only in the difference equations.

```
proc cpanel data = mycas.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
run;
```

You can also specify classification variables, interactions, and nested effects in the PREDETERMINED statement. For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

RESTRICT Statement

RESTRICT *equation* < , *equation2* . . > ;

The RESTRICT statement specifies linear equality restrictions on the parameters in the previous MODEL statement. There can be as many unique restrictions as the number of parameters in the preceding MODEL statement. Multiple RESTRICT statements are understood as joint restrictions on a model’s parameters. Restrictions on the intercept are obtained by the use of the INTERCEPT keyword. RESTRICT statements before the first MODEL statement are automatically associated with the first MODEL statement, in addition to any RESTRICT statements that follow it but precede subsequent MODEL statements.

Currently, only linear equality restrictions are permitted in PROC CPANEL. Tests and restriction expressions can be composed of only algebraic operations that involve the addition symbol (+), subtraction symbol (–), and multiplication symbol (*).

The following statements illustrate the use of the RESTRICT statement:

```
proc cpanel data = mycas.a;
  id csid tsid;
  model y = x1 x2 x3;
  restrict x1 = 0, x2 * .5 + 2 * x3 = 0;
  restrict intercept = 1;
run;
```

If you are fitting a dynamic panel model, you can place restrictions on lags of the dependent variable by referencing the name of the dependent variable, followed by an underscore and the lag order. For example:

```
proc cpanel data = mycas.a;
  model sales = price / dyndiff;
  restrict sales_1 = 0.5;
run;
```

Note that a RESTRICT statement cannot include a division sign in its formulation.

TEST Statement

TEST *equation <, equation2... > < /options> ;*

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters in the preceding MODEL statement. Like RESTRICT statements, TEST statements before the first MODEL statement are automatically associated with the first MODEL statement, in addition to any TEST statements that follow it but precede subsequent MODEL statements. Each equation specifies a linear hypothesis to be tested. All hypotheses in one TEST statement are tested jointly. Variable names in the equations must correspond to regressors in the preceding MODEL statement, and each name represents the coefficient of the corresponding regressor. The INTERCEPT keyword refers to the coefficient of the intercept.

You can specify the following *options* after a slash (/):

ALL

specifies Lagrange multiplier, likelihood ratio, and Wald tests.

LM

specifies the Lagrange multiplier test.

LR

specifies the likelihood ratio test.

WALD

specifies the Wald test.

The Wald test is performed by default.

The following statements illustrate the use of the TEST statement:

```
proc cpanel data = mycas.a;
  id csid tsid;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0 / all;
  test intercept = 0, x3 = 0;
run;
```

The first test investigates the joint hypothesis that $\beta_1 = 0$ and $0.5\beta_2 + 2\beta_3 = 0$.

Currently, only linear equality restrictions and tests are permitted in PROC CPANEL. Tests and restriction expressions can be composed only of algebraic operations that involve the addition symbol (+), subtraction symbol (−), and multiplication symbol (*).

If you are fitting a dynamic panel model, you can perform tests on lags of the dependent variable by referencing the name of the dependent variable, followed by an underscore and the lag order. For example:

```
proc cpanel data = mycas.a;
  model sales = price / dyndiff;
  test sales_1 = 0.5 / wald;
run;
```

When you use the TEST statement along with the MODEL statement options [AMACURDY](#), [DYNDIFF](#), [DYNYSYS](#), [HTAYLOR](#), [IVBTWNG](#), [IVFIXONE](#), [IVFIXONETIME](#), [IVFIXTWO](#), [IVG2SLS](#), [IVPOOLED](#), [IVRANONE](#), and [IVRANTWO](#), only the Wald test is available.

Details: CPANEL Procedure

Specifying the Input Data

Panel data are identified by both a cross section identification (ID) variable and a time variable. Suppose that cross sections are identified by the variable *State* and time periods are identified by the variable *Date*. You specify the cross section and time series variables in an ID statement. The following statements show the appropriate syntax:

```
proc panel data = mycas.a;
  id State Date;
  model ...;
run;
```

For some estimation strategies (such as one-way fixed effects), time is not relevant to the estimation. In these cases, specifying the time variable is optional. The CPANEL procedure alerts you if do not specify a time variable when one is required.

When taking lags or first differences, the CPANEL procedure determines that observations are adjacent in time only if their time values are exactly one unit apart. For this reason, you should index time by using a variable that contains consecutive integers.

PROC CPANEL performs its estimations regardless of how the data are partitioned logically across machines, but it might temporarily repartition the data to perform computations more efficiently. You can specify the

NOPART option in the PROC CPANEL statement to bypass this repartitioning, at the expense of slower execution.

Specifying the Regression Model

The CPANEL procedure is similar to other regression procedures in SAS software. Suppose you want to regress the variable Y on regressors X1 and X2. You specify the dependent variable first, followed by an equal sign, followed by the list of regression variables, as shown in the following statements:

```
proc panel data mycas.a;
    id state date;
    model y = x1 x2;
run;
```

One advantage of using PROC CPANEL is that you can incorporate a model for the structure of the error terms. It is important to consider what type of model is appropriate for your data and to specify the corresponding option in the MODEL statement. The following model estimation options are supported: POOLED, BTWNG, BTWNT, FIXONE, FDONE, FIXTWO, RANONE, RANTWO, HTAYLOR, AMACURDY, IVPOOLED, IVBTWNG, IVFIXONE, IVFIXONETIME, IVFIXTWO, IVG2SLS, IVRANONE, IVRANTWO, DYNDIFF, and DYN SYS. The methods that underlie these estimation options are described, in order, beginning with the section “[Pooled Regression \(POOLED Option\)](#)” on page 372.

The following statements fit a one-way random-effects model with variance components estimated by the Fuller-Battese (FB) method:

```
proc panel data mycas.a;
    id State Date;
    model Y = X1 X2 / ranone vcomp = fb;
run;
```

You can specify more than one estimation option in the MODEL statement, and the analysis is repeated for each specified method. You can use multiple MODEL statements to estimate different regression models or to estimate the same model by different methods.

You can also specify classification variables, interactions, and nested effects, as demonstrated by the following statements:

```
proc panel data mycas.a;
    id State Date;
    class C1 C2 C3;
    model Y = X1 X2 C1 C2 (C1) C3 C1*C3 / ranone vcomp = fb;
run;
```

For more information about constructing model effects, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

Missing Values

Any observation in the input data table that has a missing value for the cross-section ID, time series ID, dependent variable, or any model effect is ignored by PROC CPANEL when it fits the model.

If there are observations in your data where only the dependent variable is missing, you can still compute predicted values for these observations and store them in an output data table by using the OUTPUT statement.

Unbalanced Data

Unbalanced data occur when not all time values are observed for all cross sections or, when time is not part of the estimation, the cross sections are not all the same size.

Whether the data are unbalanced by design or because of missing values, almost all the methods supported by the CPANEL procedure take proper account of the unbalanced data. The one exception is the AMACURDY method, which is suitable only for balanced data.

Common Notation

This section presents notation that is common to all subsequent sections. Consider the panel regression:

$$y_{it} = \alpha + \sum_{k=1}^K x_{itk} \beta_k + u_{it} \quad i = 1, \dots, N; t = 1, \dots, T_i$$

The total number of observations is $M = \sum_{i=1}^N T_i$. For balanced data, $T_i = T$ for all i . For unbalanced data, define T to be the number of unique time periods.

The exact representation of u_{it} and the underlying assumptions depend on the estimation method.

In matrix notation the model is

$$y_{it} = \alpha + \mathbf{x}_{it} \boldsymbol{\beta} + u_{it}$$

where $\mathbf{x}_{it}$ is a $1 \times K$ row vector of independent variables and $\boldsymbol{\beta}$ is the $K \times 1$ vector of coefficients. Let $\mathbf{y}$ and $\mathbf{X}$ be matrices that are formed by arranging the dependent and independent variables by cross section, and by time within each cross section. Let $\mathbf{X}_\alpha$ be the $\mathbf{X}$ matrix augmented by a first column of ones, which corresponds to the intercept term α .

Define the following utility matrices:

$\mathbf{I}_p$ is an identity matrix of dimension p .

$\mathbf{j}_p$ is a $p \times 1$ column vector of ones.

$\mathbf{J}_p = \mathbf{j}_p \mathbf{j}_p'$ is a matrix of ones of dimension p .

$\bar{\mathbf{J}}_p = p^{-1} \mathbf{J}_p$.

$$\mathbf{E}_p = \mathbf{I}_p - \bar{\mathbf{J}}_p.$$

In the following sections, the panel data are assumed to be unbalanced unless otherwise indicated. If the data are balanced, the formulas reduce appropriately.

Pooled Regression (POOLED Option)

You perform pooled regression by specifying the POOLED option in the MODEL statement. Pooled regression is standard ordinary least squares (OLS) regression without any cross-sectional or time effects. The error structure is simply $u_{it} = e_{it}$, where the e_{it} are independently and identically distributed (iid) with zero mean and variance σ_e^2 .

Between-Groups Regression (BTWNG and BTWNT Options)

You perform between-groups regression by specifying the BTWNG option in the MODEL statement. When the data are balanced, between-groups regression is ordinary least squares (OLS) regression performed on data that have been collapsed into cross-sectional means. When the data are unbalanced, the regression is weighted to assign more importance to larger cross sections. The cross-sectional means are multiplied by the weighting factor $w_i = \sqrt{T_i/\bar{T}}$, where $\bar{T} = M/N$ is the average cross section size.

The BTWNT option works similarly, except that the data are collapsed by time period instead of by cross section.

One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)

You perform one-way fixed-effects estimation by specifying the FIXONE option in the MODEL statement. The error structure for the one-way fixed-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 .

The fixed-effects model can be estimated by ordinary least squares (OLS), treating the v_i as coefficients on dummy variables that identify the cross sections. However, when N is large, you might want to estimate only β and not v_i .

Let $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$. The matrix $\mathbf{Q}_0$ represents the *within transformation*, the conversion of the raw data to deviations from a cross section's mean. Let $\mathbf{X}_w = \mathbf{Q}_0\mathbf{X}$ and $\mathbf{y}_w = \mathbf{Q}_0\mathbf{y}$. The within estimator of β is

$$\hat{\beta}_w = (\mathbf{X}_w' \mathbf{X}_w)^{-1} \mathbf{X}_w' \mathbf{y}_w$$

The previous estimation does not involve the intercept term because $\hat{\beta}_w$ is the same whether or not the intercept α is included in the model.

Standard errors, t statistics, and fit statistics such as mean square error (MSE) are all equivalent to those obtained from OLS regression of $\mathbf{y}_w$ on $\mathbf{X}_w$. The only exception is the error degrees of freedom, which equals $M - N - K$ to account for the tacit estimation of the N fixed effects.

The estimate of the intercept is

$$\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..}\hat{\beta}_w$$

where $\bar{y}_{..}$ is the overall mean of y_{it} and $\bar{x}_{..}$ is the overall mean of x_{it} .

Fixed effects are not estimated as part of the regression, but they can be obtained by specifying the PRINTFIXED option in the MODEL statement. Each fixed effect is estimated as

$$\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_w$$

where $\bar{y}_{i.}$ and $\bar{x}_{i.}$ are cross-sectional means.

Variance estimates of $\hat{\alpha}$ and $\hat{v}_i$ are obtained by the delta method. If you specify the NOINT option, then the $\hat{v}_i$ are shifted to absorb $\hat{\alpha}$.

The FIXONETIME option works similarly, except that the data are grouped by time period instead of by cross section.

One-Way Fixed-Effects Model, First Differencing (FDONE Option)

You perform one-way fixed-effects estimation via first differencing by specifying the FDONE option in the MODEL statement. The method of first differencing offers an alternative to the within estimator $\hat{\beta}_w$. Consider the following one-way fixed-effects model:

$$y_{it} = \alpha + x_{it}\beta + v_i + e_{it}$$

For this model, the fixed effects are removed by subtracting first-order lags from both sides of the equation:

$$y_{it} - y_{i,t-1} = (x_{it} - x_{i,t-1})\beta + (e_{it} - e_{i,t-1})$$

Define $\Delta y_{it} = y_{it} - y_{i,t-1}$ and $\Delta x_{it} = x_{it} - x_{i,t-1}$, for $i = 1, \dots, N$ and $t = 2, \dots, T_i$. You obtain the first-differenced estimator, $\hat{\beta}_d$, and its variance by performing OLS regression of Δy_{it} on Δx_{it} .

You cannot obtain the error variance directly from the OLS regression because that regression is based on the differenced residual $e_{it} - e_{i,t-1}$. Instead, you can estimate σ_e^2 by $\hat{\sigma}_e^2 = \text{SSE}/(M - N - K)$, where

$$\text{SSE} = \sum_{i=1}^N \sum_{t=1}^{T_i} (y_{it} - x_{it}\hat{\beta}_d - \hat{v}_i)^2$$

and $\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_d$.

Estimation and inference for α and v_i are identical to that described in the section “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 372, with $\hat{\beta}_w$ replaced by $\hat{\beta}_d$.

Two-Way Fixed-Effects Model (FIXTWO Option)

You perform two-way fixed-effects estimation by specifying the `FIXTWO` option in the `MODEL` statement. The error specification for the two-way fixed-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i and λ_t are nonrandom parameters to be estimated. Assume the identifiability restriction $\sum_t \lambda_t = 0$.

Estimation is similar to that for one-way fixed effects, for which a within transformation is used to convert the problem to OLS regression. For two-way models under the general case of unbalanced data, the within transformation is more complex.

Following Wansbeek and Kapteyn (1989) and Baltagi (2013, sec. 9.4), let $\mathbf{X}^*$ and $\mathbf{y}^*$ be versions of $\mathbf{X}$ and $\mathbf{y}$ whose rows are sorted by time period, and by cross section within each time period. With the data sorted in this manner, define $\mathbf{D}_N$ to be the $M \times N$ design matrix for cross sections. Each row of $\mathbf{D}_N$ contains a 1 in the column that corresponds to that observation's cross section, and 0s in the remaining columns. Similarly, define $\mathbf{D}_T$ to be the $M \times T$ design matrix for time periods. In balanced data, $\mathbf{D}_N = \mathbf{j}_T \otimes \mathbf{I}_N$ and $\mathbf{D}_T = \mathbf{I}_T \otimes \mathbf{j}_N$.

Define the following:

$$\begin{aligned}\Delta_N &= \mathbf{D}_N' \mathbf{D}_N & (N \times N) \\ \Delta_T &= \mathbf{D}_T' \mathbf{D}_T & (T \times T) \\ \mathbf{A} &= \mathbf{D}_T' \mathbf{D}_N & (T \times N) \\ \bar{\mathbf{D}} &= \mathbf{D}_T - \mathbf{D}_N \Delta_N^{-1} \mathbf{A}' & (M \times T) \\ \mathbf{Q} &= \Delta_T - \mathbf{A} \Delta_N^{-1} \mathbf{A}' & (T \times T) \\ \mathbf{P} &= \mathbf{I}_M - \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N' - \bar{\mathbf{D}} \mathbf{Q}^{-1} \bar{\mathbf{D}}' & (M \times M)\end{aligned}$$

The matrix $\mathbf{P}$ provides the two-way within transformation. If the data are balanced, this amounts to transforming any data value z_{it} to $z_{it} - \bar{z}_{i.} - \bar{z}_{.t} + \bar{z}_{..}$.

Applying the two-way within transformation means that you can use OLS regression of $\mathbf{P}\mathbf{y}^*$ on $\mathbf{P}\mathbf{X}^*$ to obtain $\hat{\boldsymbol{\beta}}_f$, $\text{Var}(\hat{\boldsymbol{\beta}}_f)$, and fit statistics such as mean-square error (MSE), provided that you adjust the error degrees of freedom to equal $M - N - T - K + 1$.

Define the residual vector $\mathbf{r}^* = \mathbf{y}^* - \mathbf{X}^* \hat{\boldsymbol{\beta}}_f$. Estimates of the time effects are $\hat{\boldsymbol{\lambda}} = \mathbf{Q}^{-1} \bar{\mathbf{D}}' \mathbf{r}^*$, and estimates of the cross-sectional effects are $\hat{\mathbf{v}} = (\Theta_1 - \Theta_2 + \Theta_3) \mathbf{r}^*$, where

$$\begin{aligned}\Theta_1 &= \Delta_N^{-1} \mathbf{D}_N' \\ \Theta_2 &= \Delta_N^{-1} \mathbf{A}' \mathbf{Q}^{-1} \mathbf{D}_T' \\ \Theta_3 &= \Delta_N^{-1} \mathbf{A}' \mathbf{Q}^{-1} \mathbf{A} \Delta_N^{-1} \mathbf{D}_N'\end{aligned}$$

If the model includes an intercept, then $\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..} \hat{\boldsymbol{\beta}}_f$, and the $\hat{v}_i$ are shifted to exclude $\hat{\alpha}$.

Variance and covariance estimates for $\hat{\alpha}$, $\hat{\mathbf{v}}$, and $\hat{\boldsymbol{\lambda}}$ are obtained by the delta method, because each of these quantities is a linear transformation of $\mathbf{y}^*$ and $\hat{\boldsymbol{\beta}}_f$.

One-Way Random-Effects Model (RANONE Option)

You perform one-way random-effects estimation by specifying the RANONE option in the MODEL statement. The specification for the one-way random-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

Estimation proceeds in two steps. First, you obtain estimates of the variance components σ_v^2 and σ_e^2 . The CPANEL procedure provides five methods for estimating variance components; these methods are described in the following subsections.

Second, with the variance components in hand, you form a weight for each cross section,

$$\hat{\theta}_i = 1 - \hat{\sigma}_e / \hat{w}_i$$

where $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$. Taking $\hat{\theta}_i$, you form the partial deviations:

$$\tilde{y}_{it} = y_{it} - \hat{\theta}_i \bar{y}_i.$$

$$\tilde{\mathbf{x}}_{\alpha,it} = \mathbf{x}_{\alpha,it} - \hat{\theta}_i \bar{\mathbf{x}}_{\alpha,i}.$$

The random-effects estimation is then the result of OLS regression on the transformed data.

The CPANEL procedure provides five options for estimating variance components, as described in the following subsections.

Swamy-Arora Method

The Swamy and Arora (1972) method is the default method of estimating variance components; it can also be specified as the VCOMP=SA option in the MODEL statement. The Swamy-Arora method is part of a class of methods known as analysis of variance (ANOVA) estimators.

ANOVA estimators obtain variance components by solving a system of equations that is based on expected sums of squares. The following quadratic forms correspond to the within and between sums of squares, respectively:

$$q_e = \mathbf{u}' \mathbf{Q}_0 \mathbf{u}$$

$$q_v = \mathbf{u}' \mathbf{P}_0 \mathbf{u}$$

In these equations, $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$, $\mathbf{P}_0 = \text{diag}(\bar{\mathbf{J}}_{T_i})$, and $\mathbf{u}$ is the vector of true residuals.

The ANOVA methods differ only in how they estimate $\mathbf{u}$. The Swamy-Arora method substitutes into q_e the within residuals from one-way fixed-effects estimation and substitutes into q_v the residuals from between-groups regression.

The expected values of the quadratic forms are

$$E(\hat{\mathbf{u}}_w' \mathbf{Q}_0 \hat{\mathbf{u}}_w) = (M - N - K) \sigma_e^2$$

$$E(\hat{\mathbf{u}}_b' \mathbf{P}_0 \hat{\mathbf{u}}_b) = \left[M - \text{tr} \left\{ (\mathbf{X}'_{\alpha} \mathbf{P}_0 \mathbf{X}_{\alpha})^{-1} \mathbf{X}'_{\alpha} \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_{\alpha} \right\} \right] \sigma_v^2 + (N - K - 1) \sigma_e^2$$

where $\mathbf{Z}_0 = \text{diag}(\mathbf{j}_{T_i})$.

Estimates of σ_v^2 and σ_e^2 are obtained by first setting the expected sums of squares to their observed values, and then solving.

Wallace-Hussain Method

You can use the Wallace and Hussain (1969) method of estimating variance components by specifying the VCOMP=WH option in the MODEL statement. The Wallace-Hussain method is an ANOVA method that uses the residuals from pooled (OLS) regression, $\hat{\mathbf{u}}_p$, in both quadratic forms.

The expected values of the quadratic forms are

$$E(\hat{\mathbf{u}}_p' \mathbf{Q}_0 \hat{\mathbf{u}}_p) = (d_1 - d_3)\sigma_v^2 + (M - N - K - 1 + d_2)\sigma_e^2$$

$$E(\hat{\mathbf{u}}_p' \mathbf{P}_0 \hat{\mathbf{u}}_p) = (M - 2d_1 + d_3)\sigma_v^2 + (N - d_2)\sigma_e^2$$

where

$$d_1 = \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\}$$

$$d_2 = \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha \right\}$$

$$d_3 = \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\}$$

Wansbeek-Kapteyn Method

You can use the Wansbeek-Kapteyn method of estimating variance components by specifying the VCOMP=WK option in the MODEL statement. The method is a specialization (Baltagi and Chang 1994) of the approach used by Wansbeek and Kapteyn (1989) for unbalanced two-way models. The method was also suggested by Amemiya (1971) for balanced data.

The Wansbeek and Kapteyn method is an ANOVA method that uses the within residuals from one-way fixed effects, $\hat{\mathbf{u}}_w$, in both quadratic forms.

The expected values of the quadratic forms are

$$E(\hat{\mathbf{u}}_w' \mathbf{Q}_0 \hat{\mathbf{u}}_w) = (M - N - K)\sigma_e^2$$

$$E(\hat{\mathbf{u}}_w' \mathbf{P}_0 \hat{\mathbf{u}}_w) = (N - 1 + d)\sigma_e^2 + \left(M - M^{-1} \sum_{i=1}^N T_i^2 \right) \sigma_v^2$$

where

$$d = \text{tr} \left\{ (\mathbf{X}' \mathbf{Q}_0 \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_0 \mathbf{X} \right\} - \text{tr} \left\{ (\mathbf{X}' \mathbf{Q}_0 \mathbf{X})^{-1} \mathbf{X}' \bar{\mathbf{J}}_M \mathbf{X} \right\}$$

Fuller-Battese Method

You can use the Fuller and Battese (1974) method of estimating variance components by specifying the VCOMP=FB option in the MODEL statement. Following Baltagi (2013, sec. 9.2), obtain $\hat{\sigma}_e^2$ as the mean square error (MSE) from one-way fixed effects. The cross-sectional variance is

$$\hat{\sigma}_v^2 = \frac{R(v|\beta) - (N-1)\hat{\sigma}_e^2}{M - \text{tr}\{\mathbf{Z}'_0 \mathbf{X}_\alpha (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0\}}$$

where

$$R(v|\beta) = R(\beta|v) + R(v) - R(\beta)$$

for

$$\begin{aligned} R(v) &= \mathbf{y}' \mathbf{Z}_0 (\mathbf{Z}'_0 \mathbf{Z}_0)^{-1} \mathbf{Z}'_0 \mathbf{y} \\ R(\beta|v) &= \mathbf{y}'_w \mathbf{X}'_w (\mathbf{X}'_w \mathbf{X}_w)^{-1} \mathbf{X}'_w \mathbf{y}_w \\ R(\beta) &= \mathbf{y}' \mathbf{X}'_\alpha (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{y} \end{aligned}$$

Nerlove Method

You can use the Nerlove (1971) method of estimating variance components by specifying the VCOMP=NL option in the MODEL statement. Nerlove's method provides a simple alternative to previous estimation strategies. You estimate σ_v^2 as the sample variance of the cross-sectional effects, estimated from a one-way fixed-effects regression. Specifically, $\hat{\sigma}_v^2 = (N-1)^{-1} \sum_{i=1}^N (\hat{v}_i - \bar{v})^2$, where $\bar{v}$ is the mean of the estimated fixed effects. You estimate σ_e^2 by taking the error sum of squares from one-way fixed-effects regression and then dividing by M .

Selecting the Appropriate Variance Component Method

Baltagi and Chang (1994) conducted an extensive simulation study regarding the finite-sample properties of the variance estimators that are supported by the CPANEL procedure. The choice of method has little bearing on estimates of regression coefficients, their standard errors, and estimation of the error variance σ_e^2 . If your goal is inference on β , then the variance-component method will matter little.

The methods have varying performance in how they estimate σ_v^2 , the cross-sectional variance. All five methods tend to perform poorly if either the data are severely unbalanced or the ratio σ_v^2/σ_e^2 is much greater than 1.

The Nerlove method is the only method that guarantees a nonnegative estimate of σ_v^2 ; the other four methods reset a negative estimate to 0. However, the Nerlove method is particularly unsuitable for unbalanced data because the sample variance that it computes is not weighted by T_i .

Two-Way Random-Effects Model (RANTWO Option)

You perform two-way random-effects estimation by specifying the RANTWO option in the MODEL statement. The specification for the two-way random-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , the λ_t are iid with zero mean and variance σ_λ^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

Estimation proceeds in two steps. First, you obtain estimates of the variance components σ_v^2 , σ_λ^2 , and σ_e^2 . The CPANEL procedure provides five methods for estimating variance components; these methods are described in the following subsections.

Second, with the variance-component estimates in hand, you transform the data in such a way that estimation can take place using ordinary least squares (OLS). In two-way models with unbalanced data, the transformation is quite complex. Throughout this section, $\mathbf{y}$ and $\mathbf{X}$ are treated as being sorted first by time, and then by cross-section within time. For the definitions of the design matrices $\mathbf{D}_N$ and $\mathbf{D}_T$, see the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 374. The variance of $\mathbf{y}$ is

$$\mathbf{\Omega} = \sigma_e^2 \mathbf{I}_M + \sigma_v^2 \mathbf{D}_N \mathbf{D}_N' + \sigma_\lambda^2 \mathbf{D}_T \mathbf{D}_T'$$

and estimation proceeds as OLS regression of $\hat{\sigma}_e \hat{\mathbf{\Omega}}^{-1/2} \mathbf{y}$ on $\hat{\sigma}_e \hat{\mathbf{\Omega}}^{-1/2} \mathbf{X}$.

Rather than invert the $M \times M$ matrix $\hat{\mathbf{\Omega}}$ directly, Wansbeek and Kapteyn (1989) provide the more convenient form

$$\hat{\sigma}_e^2 \hat{\mathbf{\Omega}}^{-1} = \mathbf{V} - \mathbf{V} \mathbf{D}_T \tilde{\mathbf{P}}^{-1} \mathbf{D}_T' \mathbf{V}$$

where

$$\begin{aligned} \mathbf{V} &= \mathbf{I}_M - \mathbf{D}_N \tilde{\Delta}_N^{-1} \mathbf{D}_N' \\ \tilde{\mathbf{P}} &= \tilde{\Delta}_T - \mathbf{D}_T' \mathbf{D}_N \tilde{\Delta}_N^{-1} \mathbf{D}_N' \mathbf{D}_T \end{aligned}$$

with $\tilde{\Delta}_N = \mathbf{D}_N' \mathbf{D}_N + (\hat{\sigma}_e^2 / \hat{\sigma}_v^2) \mathbf{I}_N$ and $\tilde{\Delta}_T = \mathbf{D}_T' \mathbf{D}_T + (\hat{\sigma}_e^2 / \hat{\sigma}_\lambda^2) \mathbf{I}_T$.

If the data are balanced, then the calculations are simplified considerably—the data are transformed from z_{it} to $z_{it} - \hat{\theta}_1 \bar{z}_{i.} - \hat{\theta}_2 \bar{z}_{.t} + \hat{\theta}_3 \bar{z}_{..}$, where

$$\begin{aligned} \hat{\theta}_1 &= 1 - \hat{\sigma}_e (T \hat{\sigma}_v^2 + \hat{\sigma}_e^2)^{-1/2} \\ \hat{\theta}_2 &= 1 - \hat{\sigma}_e (N \hat{\sigma}_\lambda^2 + \hat{\sigma}_e^2)^{-1/2} \\ \hat{\theta}_3 &= \hat{\theta}_1 + \hat{\theta}_2 + \hat{\sigma}_e (T \hat{\sigma}_v^2 + N \hat{\sigma}_\lambda^2 + \hat{\sigma}_e^2)^{-1/2} - 1 \end{aligned}$$

The CPANEL procedure provides five options for estimating variance components, as described in the following subsections.

Swamy-Arora Method

The Swamy and Arora (1972) method is the default method of estimating variance components; it can also be specified as the VCOMP=SA option in the MODEL statement. The Swamy-Arora method is part of a class of methods known as analysis of variance (ANOVA) estimators.

ANOVA estimators obtain variance components by solving a system of equations that is based on expected sums of squares. The following quadratic forms correspond to the two-way within sum of squares, the sum of squares between time periods, and the sum of squares between cross sections, respectively:

$$\begin{aligned} q_e &= \mathbf{u}' \mathbf{P} \mathbf{u} \\ q_\lambda &= \mathbf{u}' \mathbf{D}_T \Delta_T^{-1} \mathbf{D}_T' \mathbf{u} \\ q_v &= \mathbf{u}' \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N' \mathbf{u} \end{aligned}$$

The matrix $\mathbf{P}$ is the two-way within transformation defined in the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 374, $\Delta_T = \mathbf{D}_T' \mathbf{D}_T$, $\Delta_N = \mathbf{D}_N' \mathbf{D}_N$, and $\mathbf{u}$ is the vector of true residuals.

The ANOVA methods differ only in how they estimate $\mathbf{u}$. The Swamy-Arora method substitutes into q_e the within residuals from two-way fixed-effects estimation, substitutes into q_λ the residuals from between-time-periods regression, and substitutes into q_v the residuals from between-cross-sections regression.

The expected values of the quadratic forms are

$$\begin{aligned} E(\hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1)\sigma_e^2 \\ E(\hat{\mathbf{u}}_\lambda' \mathbf{P}_\lambda \hat{\mathbf{u}}_\lambda) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_\lambda \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{D}_T \mathbf{D}_T' \mathbf{X}_\alpha \right\} \right] \sigma_\lambda^2 + (T - K - 1)\sigma_e^2 \\ E(\hat{\mathbf{u}}_v' \mathbf{P}_v \hat{\mathbf{u}}_v) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_v \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{D}_N \mathbf{D}_N' \mathbf{X}_\alpha \right\} \right] \sigma_v^2 + (N - K - 1)\sigma_e^2 \end{aligned}$$

where $\mathbf{P}_\lambda = \mathbf{D}_T \Delta_T^{-1} \mathbf{D}_T'$ and $\mathbf{P}_v = \mathbf{D}_N \Delta_N^{-1} \mathbf{D}_N'$.

Estimates of σ_v^2 , σ_λ^2 , and σ_e^2 are obtained by setting the expected sums of squares to their observed values and solving.

Wallace-Hussain Method

You can use the Wallace and Hussain (1969) method of estimating variance components by specifying the VCOMP=WH option in the MODEL statement. The Wallace-Hussain method is an ANOVA method that uses the residuals from pooled (OLS) regression, $\hat{\mathbf{u}}_p$, in all three quadratic forms.

The expected values of the quadratic forms are

$$\begin{aligned} E(\hat{\mathbf{u}}_p' \mathbf{P} \hat{\mathbf{u}}_p) &= d_{11}\sigma_e^2 + d_{12}\sigma_v^2 + d_{13}\sigma_\lambda^2 \\ E(\hat{\mathbf{u}}_p' \mathbf{P}_\lambda \hat{\mathbf{u}}_p) &= d_{21}\sigma_e^2 + d_{22}\sigma_v^2 + d_{23}\sigma_\lambda^2 \\ E(\hat{\mathbf{u}}_p' \mathbf{P}_v \hat{\mathbf{u}}_p) &= d_{31}\sigma_e^2 + d_{32}\sigma_v^2 + d_{33}\sigma_\lambda^2 \end{aligned}$$

Define $\Sigma = (\mathbf{X}'_\alpha \mathbf{X}_\alpha)^{-1}$, which is the inverse crossproducts matrix from pooled regression. Also define $\mathbf{S}_v = \mathbf{X}'_\alpha \mathbf{D}_N \mathbf{D}_N' \mathbf{X}_\alpha$ and $\mathbf{S}_\lambda = \mathbf{X}'_\alpha \mathbf{D}_T \mathbf{D}_T' \mathbf{X}_\alpha$, which are the individual-level sum of squares and the time-

level sum of squares, respectively. The coefficients are

$$\begin{aligned}
 d_{11} &= M - N - T + 1 - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{12} &= \text{tr}(\mathbf{S}_v \boldsymbol{\Sigma} \mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{13} &= \text{tr}(\mathbf{S}_{\lambda} \boldsymbol{\Sigma} \mathbf{X}'_{\alpha} \mathbf{P} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{21} &= T - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{22} &= T - 2\text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{D}_N \mathbf{D}'_N \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_v \boldsymbol{\Sigma}) \\
 d_{23} &= M - 2\text{tr}(\mathbf{S}_{\lambda} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_{\lambda} \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_{\lambda} \boldsymbol{\Sigma}) \\
 d_{31} &= N - \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) \\
 d_{32} &= M - 2\text{tr}(\mathbf{S}_v \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_v \boldsymbol{\Sigma}) \\
 d_{33} &= N - 2\text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{D}_T \mathbf{D}'_T \mathbf{X}_{\alpha} \boldsymbol{\Sigma}) + \text{tr}(\mathbf{X}'_{\alpha} \mathbf{P}_v \mathbf{X}_{\alpha} \boldsymbol{\Sigma} \mathbf{S}_{\lambda} \boldsymbol{\Sigma})
 \end{aligned}$$

Wansbeek-Kapteyn Method

You can use the Wansbeek-Kapteyn method of estimating variance components by specifying the VCOMP=WK option in the MODEL statement. The method is a specialization (Baltagi and Chang 1994) of the approach used by Wansbeek and Kapteyn (1989) for unbalanced two-way models.

The Wansbeek and Kapteyn method is an ANOVA method that uses the within residuals from two-way fixed effects, $\hat{\mathbf{u}}_f$, in all three quadratic forms.

The expected values of the quadratic forms are

$$\begin{aligned}
 E(\hat{\mathbf{u}}'_f \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1) \sigma_e^2 \\
 E(\hat{\mathbf{u}}'_f \mathbf{P}_{\lambda} \hat{\mathbf{u}}_f) &= (T + k_N - k_0) \sigma_e^2 + (T - \delta_N) \sigma_v^2 + (M - \delta_T) \sigma_{\lambda}^2 \\
 E(\hat{\mathbf{u}}'_f \mathbf{P}_v \hat{\mathbf{u}}_f) &= (N + k_T - k_0) \sigma_e^2 + (M - \delta_N) \sigma_v^2 + (N - \delta_T) \sigma_{\lambda}^2
 \end{aligned}$$

where $\delta_N = M^{-1} \sum_{i=1}^N T_i^2$ and $\delta_T = M^{-1} \sum_{t=1}^T N_t^2$. The other constants are defined by

$$\begin{aligned}
 k_0 &= 1 + M^{-1} \mathbf{j}'_M \mathbf{X} (\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{j}_M \\
 k_N &= \text{tr}\{(\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_{\lambda} \mathbf{X}\} \\
 k_T &= \text{tr}\{(\mathbf{X}' \mathbf{P} \mathbf{X})^{-1} \mathbf{X}' \mathbf{P}_v \mathbf{X}\}
 \end{aligned}$$

When the NOINT option is specified, the variance-component equations change slightly: k_0 , δ_N , and δ_T are all replaced by 0.

Fuller-Battese Method

You can use the Fuller and Battese (1974) method of estimating variance components by specifying the VCOMP=FB option in the MODEL statement. Following the discussion in Baltagi, Song, and Jung (2002), the Fuller-Battese method is a variation of the three ANOVA methods discussed previously.

The quadratic form, q_e , is the same as in the previous methods, and $\mathbf{u}$ is estimated by the two-way within residuals $\hat{\mathbf{u}}_f$. The other two quadratic forms, q_λ and q_v , are replaced by the error sums of squares from one-way fixed-effects estimations.

The resulting system of equations is

$$\begin{aligned} E(\hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f) &= (M - N - T - K + 1)\sigma_e^2 \\ E(\tilde{\mathbf{u}}_\lambda' \tilde{\mathbf{u}}_\lambda) &= (M - T - K)\sigma_e^2 + \left[M - T - \text{tr} \left\{ \mathbf{X}' \mathbf{W}_\lambda \mathbf{D}_N \mathbf{D}_N' \mathbf{W}_\lambda \mathbf{X} (\mathbf{X}' \mathbf{W}_\lambda \mathbf{X})^{-1} \right\} \right] \sigma_v^2 \\ E(\tilde{\mathbf{u}}_v' \tilde{\mathbf{u}}_v) &= (M - N - K)\sigma_e^2 + \left[M - N - \text{tr} \left\{ \mathbf{X}' \mathbf{W}_v \mathbf{D}_T \mathbf{D}_T' \mathbf{W}_v \mathbf{X} (\mathbf{X}' \mathbf{W}_v \mathbf{X})^{-1} \right\} \right] \sigma_\lambda^2 \end{aligned}$$

where $\mathbf{W}_\lambda = \mathbf{I}_M - \mathbf{P}_\lambda$, $\mathbf{W}_v = \mathbf{I}_M - \mathbf{P}_v$, $\tilde{\mathbf{u}}_\lambda$ are the residuals from a one-way model with time fixed effects, and $\tilde{\mathbf{u}}_v$ are the residuals from a one-way model with individual fixed effects.

Nerlove Method

You can use the Nerlove (1971) method of estimating variance components by specifying the VCOMP=NL option in the MODEL statement.

Begin by fitting a two-way fixed-effects model. The estimator of the error variance is

$$\hat{\sigma}_e^2 = M^{-1} \hat{\mathbf{u}}_f' \mathbf{P} \hat{\mathbf{u}}_f$$

You obtain $\hat{\sigma}_v^2$ as the sample variance of the N estimated individual effects, and $\hat{\sigma}_\lambda^2$ as the sample variance of the T estimated time effects.

Hausman-Taylor Estimation (HTAYLOR Option)

You perform Hausman-Taylor estimation by specifying the HTAYLOR option in the MODEL statement. The Hausman and Taylor (1981) model is a hybrid that combines the consistency of a fixed-effects model with the efficiency and applicability of a random-effects model. One-way random-effects models assume exogeneity of the regressors, namely that they are independent of both the cross-sectional and observation-level errors. When some regressors are correlated with the cross-sectional errors, the random effects model can be adjusted to deal with this form of endogeneity.

Consider the one-way model:

$$y_{it} = \mathbf{x}_{1it} \boldsymbol{\beta}_1 + \mathbf{x}_{2it} \boldsymbol{\beta}_2 + \mathbf{z}_{1i} \boldsymbol{\gamma}_1 + \mathbf{z}_{2i} \boldsymbol{\gamma}_2 + v_i + e_{it}$$

The regressors are subdivided so that $\mathbf{x}_{1it}$ and $\mathbf{x}_{2it}$ vary within cross sections whereas $\mathbf{z}_{1i}$ and $\mathbf{z}_{2i}$ do not and would otherwise be dropped from a fixed-effects model. The subscript 1 denotes variables that are independent of both error terms (exogenous variables), and the subscript 2 denotes variables that are

independent of the observation-level errors e_{it} but correlated with cross-sectional errors v_i . The intercept term (if your model has one) is included as part of $\mathbf{z}_{1i}$.

The Hausman-Taylor estimator is a two-stage least squares (2SLS) regression on data that are weighted similarly to data for random-effects estimation. The weights are functions of the estimated variance components.

The observation-level variance is estimated from a one-way fixed-effects model fit. Obtain $\mathbf{y}_w$, $\mathbf{X}_w$, and $\hat{\boldsymbol{\beta}}_w$ from the section “[One-Way Fixed-Effects Model \(FIXONE and FIXONETIME Options\)](#)” on page 372. Then $\hat{\sigma}_e^2 = \text{SSE}/(M - N)$, where

$$\text{SSE} = (\mathbf{y}_w - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)' (\mathbf{y}_w - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)$$

To estimate the cross-sectional error variance, form the mean-residual vector $\mathbf{r} = \mathbf{P}'_0(\mathbf{y} - \mathbf{X}_w \hat{\boldsymbol{\beta}}_w)$, where $\mathbf{P}_0 = \text{diag}(\bar{\mathbf{J}}_{T_i})$. You can use the mean residuals to obtain intermediate estimates of the coefficients for $\mathbf{z}_1$ and $\mathbf{z}_2$ via two-stage least squares (2SLS) estimation. At the first stage, use $\mathbf{x}_1$ and $\mathbf{z}_1$ as instrumental variables to predict $\mathbf{z}_2$. At the second stage, regress $\mathbf{r}$ on both $\mathbf{z}_1$ and the predicted $\mathbf{z}_2$ to obtain $\hat{\boldsymbol{\gamma}}_1^m$ and $\hat{\boldsymbol{\gamma}}_2^m$.

To estimate the cross-sectional variance, compute $\hat{\sigma}_v^2 = \{R(v)/N - \hat{\sigma}_e^2\}/\bar{T}$, where $\bar{T} = N/(\sum_{i=1}^N T_i^{-1})$ and

$$R(v) = (\mathbf{r} - \mathbf{Z}_1 \hat{\boldsymbol{\gamma}}_1^m - \mathbf{Z}_2 \hat{\boldsymbol{\gamma}}_2^m)' (\mathbf{r} - \mathbf{Z}_1 \hat{\boldsymbol{\gamma}}_1^m - \mathbf{Z}_2 \hat{\boldsymbol{\gamma}}_2^m)$$

The design matrices $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are formed by stacking the data observations of $\mathbf{z}_{1i}$ and $\mathbf{z}_{2i}$, respectively.

After variance-component estimation, transform the dependent variable into partial deviations: $y_{it}^* = y_{it} - \hat{\theta}_i \bar{y}_{i.}$. Likewise, transform the regressors to form $\mathbf{x}_{1it}^*$, $\mathbf{x}_{2it}^*$, $\mathbf{z}_{1i}^*$, and $\mathbf{z}_{2i}^*$. The partial weights $\hat{\theta}_i$ are determined by $\hat{\theta}_i = 1 - \hat{\sigma}_e/\hat{w}_i$, with $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$.

Finally, you obtain the Hausman-Taylor estimates by performing 2SLS regression of y_{it}^* on $\mathbf{x}_{1it}^*$, $\mathbf{x}_{2it}^*$, $\mathbf{z}_{1i}^*$, and $\mathbf{z}_{2i}^*$. For the first-stage regression, use the following instruments:

- $\bar{x}_{it}$, the deviations from cross-sectional means for all time-varying variables (correlated and uncorrelated) for the i th cross section during time period t
- $(1 - \hat{\theta}_i) \bar{x}_{1i.}$, where $\bar{x}_{1i.}$ are the means of the time-varying exogenous variables for the i th cross section
- $(1 - \hat{\theta}_i) \mathbf{z}_{1i}$

Multiplication by the factor $(1 - \hat{\theta}_i)$ is redundant in balanced data, but necessary in the unbalanced case to produce accurate instrumentation; see Gardner (1998).

Let k_1 equal the number of regressors in $\mathbf{x}_1$, and let g_2 equal the number of regressors in $\mathbf{z}_2$. Then the Hausman-Taylor model is identified only if $k_1 \geq g_2$; otherwise, no estimation takes place.

Hausman and Taylor (1981) describe a specification test that compares their model to fixed effects. For a null hypothesis of fixed effects, Hausman's m statistic is calculated by comparing the parameter estimates and variance matrices for both models identically to how it is calculated for one-way random effects models; for more information, see the section “[Hausman Specification Tests](#)” on page 399. However, the degrees of freedom of the test are not based on matrix rank but instead are equal to $k_1 - g_2$.

Amemiya-MaCurdy Estimation (AMACURDY Option)

You perform Amemiya-MaCurdy estimation by specifying the AMACURDY option in the MODEL statement. The Amemiya and MaCurdy (1986) model is similar to the Hausman-Taylor model. Following the development in the section “Hausman-Taylor Estimation (HTAYLOR Option)” on page 381, estimation is identical up to the final 2SLS instrumental variables regression. In addition to the set of instruments used by the Hausman-Taylor estimator, use the following:

- $\mathbf{x}_{1i1}, \mathbf{x}_{1i2}, \dots, \mathbf{x}_{1iT}$

For each observation in the i th cross section, you use the data on the time-varying exogenous regressors for the entire cross section. Because of the structure of the added instruments, the Amemiya-MaCurdy estimator can be applied only to balanced data.

The Amemiya-MaCurdy model attempts to gain efficiency over Hausman-Taylor by adding instruments. This comes at a price of a more stringent assumption on the exogeneity of the $\mathbf{x}_1$ variables. Although the Hausman-Taylor model requires only that the cross-sectional means of $\mathbf{x}_1$ be orthogonal to v_i , the Amemiya-MaCurdy estimation requires orthogonality at every point in time; see Baltagi (2013, sec. 7.4).

A Hausman specification test is provided to test the validity of the added assumption. Define $\boldsymbol{\alpha}' = (\boldsymbol{\beta}'_1, \boldsymbol{\beta}'_2, \boldsymbol{\gamma}'_1, \boldsymbol{\gamma}'_2)$, its Hausman-Taylor estimate as $\hat{\boldsymbol{\alpha}}_{HT}$, and its Amemiya-MaCurdy estimate as $\hat{\boldsymbol{\alpha}}_{AM}$. Under the null hypothesis, both estimators are consistent and $\hat{\boldsymbol{\alpha}}_{AM}$ is efficient. The Hausman test statistic is

$$m = (\hat{\boldsymbol{\alpha}}_{HT} - \hat{\boldsymbol{\alpha}}_{AM})' (\hat{\boldsymbol{\Sigma}}_{HT} - \hat{\boldsymbol{\Sigma}}_{AM})^{-1} (\hat{\boldsymbol{\alpha}}_{HT} - \hat{\boldsymbol{\alpha}}_{AM})$$

where $\hat{\boldsymbol{\Sigma}}_{HT}$ and $\hat{\boldsymbol{\Sigma}}_{AM}$ are variance-covariance estimates of $\hat{\boldsymbol{\alpha}}_{HT}$ and $\hat{\boldsymbol{\alpha}}_{AM}$, respectively. Under the null hypothesis, m follows a χ^2 distributed with degrees of freedom equal to the rank of $(\hat{\boldsymbol{\Sigma}}_{HT} - \hat{\boldsymbol{\Sigma}}_{AM})^{-1}$.

Instrumental Variables Regression

Recall the original specification of the regression model:

$$y_{it} = \alpha + \sum_{k=1}^K x_{itk} \beta_k + u_{it} \quad i = 1, \dots, N; t = 1, \dots, T_i$$

Consider that a subset of the regressors are *endogenous*—that is, these regressors are correlated with the observation-level errors u_{it} .

Formally, divide $\mathbf{x}$ and $\boldsymbol{\beta}$ so that

$$y_{it} = \mathbf{x}_{1it} \boldsymbol{\beta}_1 + \mathbf{x}_{2it} \boldsymbol{\beta}_2 + u_{it}$$

where $\mathbf{x}_{1it}$ are k_1 uncorrelated (exogenous) regressors, $\mathbf{x}_{2it}$ are k_2 endogenous regressors, and the intercept (if your model has one) is contained in $\mathbf{x}_{1it}$.

In matrix notation, the model is

$$\mathbf{y} = \mathbf{X}_1 \boldsymbol{\beta}_1 + \mathbf{X}_2 \boldsymbol{\beta}_2 + \mathbf{u}$$

where the matrix rows are arranged by cross section, and by time within cross section.

Instrumental variables (IV) regression is the process of using external regressors, called *instruments*, to deal with the endogeneity. Let $\mathbf{Z}$ represent a set of p instruments such that $p \geq k_2$. The set of endogenous regressors, $\mathbf{X}_2$, are specified as effects in the ENDOGENOUS statement. The set of external instruments, $\mathbf{Z}$, are specified as effects in the INSTRUMENTS statement. Define $\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2]$.

The CPANEL procedure performs instrumental variables regression by the method of two-stage least squares (2SLS) or the efficient GMM. Two-stage least squares is a special case of the GMM and is equivalent to the one-step GMM method. The efficient GMM method, which is also called two-step GMM, is a GMM that uses the optimal weighting matrix. The optimal weighting matrix is based on the residuals from the one-step GMM. If you define the estimated residual vector as $\hat{\mathbf{u}}$ and the degrees of freedom as df , then you can obtain the estimated variance of the error term as $\sigma_u^2 = \mathbf{u}'\mathbf{u}/df$ and use it to form the optimal weighting matrix.

The 2SLS estimator can be obtained as

$$\hat{\boldsymbol{\beta}}_{2SLS} = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{y}$$

The estimated variance-covariance matrix of the 2SLS estimator is

$$\text{Var}(\hat{\boldsymbol{\beta}}_{2SLS}) = \hat{\sigma}^2[\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

You can also obtain the heteroscedasticity robust standard error for 2SLS by specifying the ROBUST option. Define $\hat{\Omega}$ as the diagonal matrix of squared residuals from the 2SLS estimation:

$$\hat{\Omega} = \begin{pmatrix} \hat{u}_1^2 & & 0 \\ & \ddots & \\ 0 & & \hat{u}_{NT}^2 \end{pmatrix}$$

Define $P_Z = \mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'$ and $W_2 = \mathbf{Z}'\hat{\Omega}\mathbf{Z}$. Then the estimated robust variance-covariance matrix of the 2SLS estimator is

$$\text{Var}_r(\hat{\boldsymbol{\beta}}_{2SLS}) = [\mathbf{X}'P_Z\mathbf{X}]^{-1}[\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\mathbf{Z})^{-1}W_2(\mathbf{Z}'\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}][\mathbf{X}'P_Z\mathbf{X}]^{-1}$$

The two-step GMM estimator can be obtained as

$$\hat{\boldsymbol{\beta}}_{GMM} = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{y}$$

with asymptotic variance

$$\text{Var}(\hat{\boldsymbol{\beta}}_{GMM}) = [\mathbf{X}'\mathbf{Z}(\mathbf{Z}'\hat{\Omega}\mathbf{Z})^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

You can also obtain the heteroscedasticity robust standard error for two-step GMM. Define $\hat{\Omega}_2$ as the diagonal matrix of squared residuals from the two-step GMM estimation and $W_3 = \mathbf{Z}'\hat{\Omega}_2\mathbf{Z}$. Then the estimated robust variance-covariance matrix of the two-step GMM estimator is

$$\text{Var}_r(\hat{\boldsymbol{\beta}}_{GMM}) = [\mathbf{X}'\mathbf{Z}(W_2)^{-1}\mathbf{Z}'\mathbf{X}]^{-1}[\mathbf{X}'\mathbf{Z}(W_2)^{-1}W_3(W_2)^{-1}\mathbf{Z}'\mathbf{X}][\mathbf{X}'\mathbf{Z}(W_2)^{-1}\mathbf{Z}'\mathbf{X}]^{-1}$$

Similar to the two-step GMM estimation for a dynamic panel in the section “[Estimation](#)” on page 392, you can obtain a bias-corrected variance of $\hat{\boldsymbol{\beta}}_{GMM}$ for IV models by using the BIASCORRECTED option. The

formulas are as the same as those from the section “Estimation” on page 392. Define the one-step and two-step residuals as $\hat{u}_{1i} = y_i - X_i \hat{\beta}_{2SLS}$ and $\hat{u}_{2i} = y_i - X_i \hat{\beta}_{GMM}$, respectively. Also define

$$P_e = \sum_{i=1}^N Z_i' \hat{u}_{2i}; \quad P_x = \sum_{i=1}^N Z_i' X_i^s$$

Formulate the matrix D such that its k th column is $D_k = V_2 P_x' (W_2)^{-1} F_k (W_2)^{-1} P_e$, where $V_2 = \text{Var}(\hat{\beta}_{GMM})$. The matrix F_k is the quadratic form

$$F_k = \sum_{i=1}^{NT} Z_i' (x_{ik} \hat{u}_{1i}' + \hat{u}_{1i} x_{ik}') Z_i$$

where x_{ik} is the k th column of X_i . Note that the F_k here is different from the F_k in the dynamic panel section. By using this F_k , the bias-corrected standard errors only correct for the bias from the two-step GMM, but not robust to serial correlation.

The Windmeijer (2005) bias-corrected variance is

$$\text{Var}^w(\hat{\beta}_{GMM}) = V_2 + DV_2 + V_2 D' + DV_1^r D'$$

where V_1^r is the robust variance estimate of $\hat{\beta}_{2SLS}$.

Depending on the error structure of u_{it} , there are eight IV models available, as described in the following subsections. In each model different X , Z , and y are substituted in the preceding formulas to obtain the 2SLS and the GMM estimators.

Instrumental Variables Pooled Regression (IVPOOLED Option)

You perform IV pooled regression by specifying the IVPOOLED option in the MODEL statement. Pooled model uses the standard regression model without any cross-sectional or time effects. The error structure is simply $u_{it} = e_{it}$, where the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with x_{1it} , but correlated with x_{2it} . The degree of freedom is $M - K - 1$.

Instrumental Variables Between-Groups Regression (IVBTWNG Option)

You perform IV between-groups regression by specifying the IVBTWNG option in the MODEL statement. When the data are balanced, between-groups IV model estimation is two-stage least squares (2SLS) regression or two-step GMM estimation performed on data that have been collapsed into cross-sectional means. When the data are unbalanced, the regression is weighted to assign more importance to larger cross sections. The cross-sectional means are multiplied by the weighting factor $w_i = \sqrt{T_i/\bar{T}}$, where $\bar{T} = M/N$ is the average cross section size.

Instrumental Variables One-Way Fixed Effects (IVFIXONE Option)

You perform IV one-way fixed-effects estimation by specifying the IVFIXONE option in the MODEL statement. The error structure for the one-way fixed-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with x_{1it} , but correlated with x_{2it} .

The IV fixed-effects model can be estimated as an IV pooled regression, treating the v_i as coefficients on dummy variables that identify the cross sections. However, when N is large, you might want to estimate only β and not v_i .

Let $\mathbf{Q}_0 = \text{diag}(\mathbf{E}_{T_i})$. The matrix $\mathbf{Q}_0$ is the within transformation that removes means (and v_i) from the estimation. Define $\mathbf{y}_w = \mathbf{Q}_0\mathbf{y}$, and use the same transformation to form $\mathbf{X}_{1w}$, $\mathbf{X}_{2w}$ and $\mathbf{Z}_w$. Because the within transformation also removes the intercept term, the intercept is not included as part of $\mathbf{X}_1$.

The within-group estimates $\hat{\beta}_{\text{WIV}}$, $\text{Var}(\hat{\beta}_{\text{WIV}})$ can be obtained by substituting $\mathbf{y}_w$, $\mathbf{X}_w$, and $\mathbf{Z}_w$ into the general estimation equation in section “[Instrumental Variables Regression](#)” on page 383. You can obtain these estimates by using 2SLS method or two-step GMM method. When forming the error variance $\hat{\sigma}_e^2$, the error degrees of freedom are adjusted to equal $M - N - K$ to account for the N fixed effects.

The estimate of the intercept of the IVFIXONE model is

$$\hat{\alpha} = \bar{y}_{..} - \bar{x}_{..}\hat{\beta}_{\text{WIV}}$$

where $\bar{y}_{..}$ is the overall mean of y_{it} and $\bar{x}_{..}$ is the overall mean of x_{it} .

Fixed effects are not estimated as part of the regression but can be obtained by specifying the PRINTFIXED option in the MODEL statement. Each fixed effect in 2SLS is estimated as

$$\hat{v}_i = \bar{y}_{i.} - \bar{x}_{i.}\hat{\beta}_{\text{WIV}}$$

where $\bar{y}_{i.}$ and $\bar{x}_{i.}$ are cross-sectional means.

Variance estimates of $\hat{\alpha}$ and $\hat{v}_i$ are obtained by the delta method. If you specify the NOINT option, then the $\hat{v}_i$ are shifted to absorb $\hat{\alpha}$.

When the estimation method is 2SLS, PROC CPANEL reports a poolability test F . The F test assumes the null hypothesis that all fixed effects are jointly 0. Following the methodology of Wooldridge (2002, pp. 98–99) for restricted 2SLS, the test statistic is

$$F = \frac{(\text{SSE}_{2r} - \text{SSE}_{2u}) / (N - 1)}{\text{SSE}_u / (M - N - K)} \sim F(N - 1, M - N - K)$$

SSE_u is the final unrestricted error sum of squares

$$\text{SSE}_u = \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \mathbf{X}_{2w}\hat{\beta}_{2,\text{WIV}} \right)' \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \mathbf{X}_{2w}\hat{\beta}_{2,\text{WIV}} \right)$$

SSE_{2u} is the second-stage unrestricted error sum of squares

$$\text{SSE}_{2u} = \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \hat{\mathbf{X}}_{2w}\hat{\beta}_{2,\text{WIV}} \right)' \left(\mathbf{y}_w - \mathbf{X}_{1w}\hat{\beta}_{1,\text{WIV}} - \hat{\mathbf{X}}_{2w}\hat{\beta}_{2,\text{WIV}} \right)$$

and SSE_{2r} is the second-stage restricted error sum of squares, which is equal to the residual sum of squares from a pooled OLS regression of $\mathbf{y}$ on $\mathbf{X}_1$ and $\mathbf{X}_2^* = \hat{\mathbf{X}}_{2w} + \mathbf{P}_0\mathbf{X}_2$.

Instrumental Variables One-Way Time Fixed Effects (IVFIXONETIME Option)

You perform IV one-way time fixed-effects estimation by specifying the IVFIXONETIME option in the MODEL statement. The error structure for the one-way time fixed-effects model is

$$u_{it} = \lambda_t + e_{it}$$

where the λ_t are nonrandom parameters that are restricted to sum to 0, and the e_{it} are iid with zero mean and variance σ_e^2 . The errors e_{it} are uncorrelated with $\mathbf{x}_{1it}$, but correlated with $\mathbf{x}_{2it}$.

The steps of the estimation process of the one-way time fixed-effects model are the same as they are for the one-way fixed-effects model in the section “[Instrumental Variables One-Way Fixed Effects \(IVFIXONE Option\)](#)” on page 385, except IVFIXONETIME uses a different within-group transformation matrix $\mathbf{Q}_0 = \mathbf{I}_M - \bar{\mathbf{J}}_p \otimes \mathbf{I}_T$. By using $\mathbf{Q}_0$, the within-group transformation removes the time mean rather than the cross section means.

Instrumental Variables One-Way Time Fixed Effects (IVFIXTWO Option)

You perform IV two-way fixed-effects estimation by specifying the IVFIXTWO option in the MODEL statement. The error specification for the two-way fixed-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i and λ_t are nonrandom parameters to be estimated. Assume the identifiability restriction $\sum_t \lambda_t = 0$.

To estimate the IV two-way fixed-effects model, first, a within transformation is needed to remove the cross-section means and time means. This step is the same as it is in section “[Two-Way Fixed-Effects Model \(FIXTWO Option\)](#)” on page 374 for both balanced and unbalanced panel. After the transformation is done, the within-group estimates $\hat{\beta}_{WIV}$, $\text{Var}(\hat{\beta}_{WIV})$ can be obtained by substituting $\mathbf{y}_w$, $\mathbf{X}_w$, and $\mathbf{Z}_w$ into the general estimation equation in section “[Instrumental Variables Regression](#)” on page 383.

Instrumental Variables One-Way Random Effects (IVRANONE Option)

You perform IV one-way random-effects estimation by specifying the IVRANONE option in the MODEL statement. The specification for the one-way random-effects model is

$$u_{it} = v_i + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . The regressors in $\mathbf{x}_{1it}$ are orthogonal to e_{it} , but the regressors in $\mathbf{x}_{2it}$ are correlated with e_{it} . This form of endogeneity is different from that assumed by Hausman and Taylor (1981), where $\mathbf{x}_{2it}$ is correlated with v_i but not with e_{it} .

The estimation performed by the CPANEL procedure is based on the error-component two-stage least squares (EC2SLS) method of Baltagi (1981). Estimation proceeds in two steps.

First, estimates of the variance components are obtained by using a variant of the Swamy-Arora method. Consider the system of equations

$$\begin{aligned} E(\hat{\mathbf{u}}_F' \mathbf{Q}_0 \hat{\mathbf{u}}_F) &= (M - N - K)\sigma_e^2 \\ E(\hat{\mathbf{u}}_B' \mathbf{P}_0 \hat{\mathbf{u}}_B) &= \left[M - \text{tr} \left\{ (\mathbf{X}'_\alpha \mathbf{P}_0 \mathbf{X}_\alpha)^{-1} \mathbf{X}'_\alpha \mathbf{Z}_0 \mathbf{Z}'_0 \mathbf{X}_\alpha \right\} \right] \sigma_v^2 + (N - K - 1)\sigma_e^2 \end{aligned}$$

where $\hat{u}_F$ are the residuals from IV one-way fixed effects, $\hat{u}_B$ are the residuals from IV between-groups regression, and the other quantities are as defined in the section “[Swamy-Arora Method](#)” on page 375. Estimates of σ_v^2 and σ_e^2 are obtained by setting the expected sums of squares to their observed values and solving.

Second, with the variance components in hand, you form a weight for each cross section:

$$\hat{\theta}_i = 1 - \hat{\sigma}_e / \hat{w}_i$$

where $\hat{w}_i^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2$. Taking $\hat{\theta}_i$, you form the partial deviations $y_{it}^* = y_{it} - \hat{\theta}_i \bar{y}_{i.}$ and similarly to form x_{1it}^* , x_{2it}^* , and z_{it}^* . Estimation then proceeds as a 2SLS or two-step GMM estimation of y_{it}^* on x_{1it}^* and x_{2it}^* . The following instruments are used in the estimation: x_{1it}^* , $\bar{x}_{1i.}$, z_{it}^* , and $\bar{z}_{i.}$. When the GMM=TWOSTEP option is specified, the estimation of the variance by the Swamy-Arora method is based on 2SLS estimation of within-group estimation and between-group estimation.

A Hausman specification test that compares IV random effects to IV fixed effects is formed by using the method described in the section “[Hausman Specification Tests](#)” on page 399. Failure to reject the null hypothesis favors the random-effects specification.

Instrumental Variables Generalized Two-Stage Least Squares (IVG2SLS Option)

You perform IV generalized two-stage least squares estimation by specifying the IVG2SLS option in the MODEL statement. The model specification and the model estimation steps are as the same as they are for the IVRANONE option in the section “[Instrumental Variables One-Way Random Effects \(IVRANONE Option\)](#)” on page 387, except for the choice of instrumental variables.

According to Baltagi (1981), in the generalized two-stage least squares model, the following instruments are used in the estimation: x_{1it}^* and z_{it}^* . The instrument z_{it}^* always includes an intercept whether an intercept is specified or not. Asymptotically, the estimates from the IVG2SLS and IVRANONE options yield the same variance-covariance matrix. In a small sample, the IVRANONE option can be more efficient by including the redundant instruments $\bar{x}_{1i.}$ and $\bar{z}_{i.}$.

Instrumental Variables Two-Way Random Effects (IVRANTWO Option)

You perform IV two-way random-effects estimation by specifying the IVRANTWO option in the MODEL statement. The specification for the error term of the IV two-way random-effects model is

$$u_{it} = v_i + \lambda_t + e_{it}$$

where the v_i are iid with zero mean and variance σ_v^2 , the λ_t are iid with zero mean and variance σ_λ^2 , and the e_{it} are iid with zero mean and variance σ_e^2 . Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with $\mathbf{X}$.

IVRANTWO deals only with balanced panel data. The estimation proceeds in three steps, and the details are the same as they are for the RANTWO model in Baltagi (2013, sec. 3.3). First, estimate the variance components σ_v^2 , σ_λ^2 , and σ_e^2 . Define $Q_1 = I_N \otimes I_T$, $Q_2 = I_N \otimes \bar{J}_T$, $Q_3 = \bar{J}_N \otimes I_T$, $Q_4 = \bar{J}_N \otimes \bar{J}_T$, and $Q = Q_1 - Q_2 - Q_3 + k_0 Q_4$. If an intercept is specified, $k_0 = 1$; if an intercept is not specified, $k_0 = 0$. Run 2SLS regression on Qy , QX , and QZ to obtain the sum of squared errors for IVFIXTWO and name it sse_1 . Also define the degrees of freedom for IVFIXTWO to be df_1 . Run 2SLS regression on Q_2y , Q_2X , and Q_2Z to obtain the sum of squared errors for IVBTWNG and name it sse_2 . Also define the degrees of freedom for IVBTWNG to be df_2 . Run 2SLS regression on Q_3y , Q_3X , and Q_3Z to obtain the

sum of squared errors for IV between times and name it sse_3 . Also define the degrees of freedom for IV between times to be df_3 . Define

$$\hat{\lambda}_1 = \hat{\sigma}_e^2 = sse_1/df_1$$

$$\hat{\lambda}_2 = sse_2/df_2$$

$$\hat{\lambda}_3 = sse_3/df_3$$

The estimated variances are

$$\hat{\sigma}_v^2 = (\hat{\lambda}_2 - \hat{\lambda}_1)/T$$

$$\hat{\sigma}_\lambda^2 = (\hat{\lambda}_3 - \hat{\lambda}_1)/N$$

Define $\lambda_4 = T\sigma_v^2 + N\sigma_\lambda^2 + \sigma_e^2$, and obtain

$$\theta_1 = 1 - \sigma_e/\lambda_2^{1/2}$$

$$\theta_2 = 1 - \sigma_e/\lambda_3^{1/2}$$

$$\theta_3 = \theta_1 + \theta_2 + (\sigma_e/\lambda_4^{1/2}) - 1$$

Second, transform the data using θ_1 , θ_2 , and θ_3 . Define $Q_r = Q_1 - \theta_1 Q_2 - \theta_2 Q_3 + \theta_3 Q_4$. The transformed variables are obtained as $Q_r y$, $Q_r X$, and $Q_r Z$. Lastly, substitute the transformed variables into the general estimation equation that is shown in the section “[Instrumental Variables Regression](#)” on page 383 to obtain the estimates for the IVRANTWO model. Notice that if $\sigma_e^2 = 0$, this model collapses to the IVPOOLED model. If $\sigma_\lambda^2 = 0$ and $\sigma_v^2 = 0$, this model also collapses to the IVPOOLED model. If $\sigma_\lambda^2 = 0$ and $\sigma_v^2 > 0$, this model collapses to the IVRANONE model. If $\sigma_\lambda^2 > 0$ and $\sigma_v^2 = 0$, this model collapses to the IVRANONE model with respect to time.

Similar to the IVRANONE model, when the estimation option is two-step GMM, the variance components are obtained using 2SLS. When $\sigma_\lambda^2 > 0$, $\sigma_v^2 > 0$, and $\sigma_e^2 > 0$, the instrumental variables include only the transformed exogenous variable $\mathbf{X}_1^*$ and the transformed instrumental variable $\mathbf{Z}^*$ without the redundant instruments such as cross-section means and time means.

Dynamic Panel Estimation (DYNDIFF and DYNSSYS Options)

You perform dynamic panel estimation that uses first differences by specifying the DYNDIFF option in the MODEL statement. For dynamic panel estimation that uses a full system of difference and level equations, specify the DYNSSYS option. For an example of dynamic panel estimation, see [Example 9.5](#).

Dynamic panel models are regression models that include lagged versions of the dependent variable as covariates. Consider the following panel regression, which includes L lags of the dependent variable:

$$y_{it} = \sum_{j=1}^L \phi_j y_{i,t-j} + \sum_{k=1}^K x_{itk} \beta_k + v_i + \epsilon_{it}$$

The effect v_i is common to all observations for that individual, so it is correlated with any lagged y because it played a role in its realization. For this reason, lagged dependent variables are endogenous regressors and require special consideration.

First Differencing

For ease of notation, consider the special case $L = K = 1$. A first attempt to remove the source of the correlation would be to take first differences, which removes v_i . That is,

$$\Delta y_{it} = \phi \Delta y_{i,t-1} + \Delta x_{it} \beta + \eta_{it}$$

where $\Delta y_{it} = y_{i,t} - y_{i,t-1}$, $\Delta x_{it} = x_{i,t} - x_{i,t-1}$, and $\eta_{it} = \epsilon_{i,t} - \epsilon_{i,t-1}$. Even though the individual effects are removed, the problem of endogeneity persists because $\Delta y_{i,t-1}$ is correlated with the differenced error term η_{it} . That is because $\epsilon_{i,t-1}$ is a component of $y_{i,t-1}$ (Nickell 1981).

Arellano and Bond (1991) show that you can use the generalized method of moments (GMM) to obtain a consistent estimator. In GMM parlance, the moment condition that $E\{(\Delta y_{i,t-1})\eta_{it}\} = 0$ is violated. Estimation requires a set of instrumental variables that do meet their moment conditions and that can adequately predict $\Delta y_{i,t-1}$. A natural set of instruments is $y_{i,t-2}$ and all other previous realizations of y . These lags of y are not correlated with $\epsilon_{i,t-1}$ because they occurred before time $t - 1$. Given the autoregressive nature of the model, $y_{i,t-1}$ (and hence $\Delta y_{i,t-1}$) is well predicted by its previous values.

Begin with $t = 3$, the first time period in which the differenced model holds. The dynamic regression model for individual i can be expressed as

$$\mathbf{y}_i^d = \mathbf{X}_i^d \boldsymbol{\gamma} + \boldsymbol{\eta}_i^d$$

where

$$\mathbf{y}_i^d = \begin{pmatrix} \Delta y_{i3} \\ \Delta y_{i4} \\ \vdots \\ \Delta y_{iT} \end{pmatrix} \quad \mathbf{X}_i^d = \begin{pmatrix} \Delta y_{i2} & \Delta x_{i3} \\ \Delta y_{i3} & \Delta x_{i4} \\ \vdots & \vdots \\ \Delta y_{i,T-1} & \Delta x_{iT} \end{pmatrix} \quad \boldsymbol{\gamma} = \begin{pmatrix} \phi \\ \beta \end{pmatrix} \quad \boldsymbol{\eta}_i^d = \begin{pmatrix} \eta_{i3} \\ \eta_{i4} \\ \vdots \\ \eta_{iT} \end{pmatrix}$$

Proceeding with the idea that you can use $(y_{i1}, \dots, y_{i,t-1})$ as instruments for Δy_{it} , the instrument matrix for the lagged dependent variables is

$$\mathbf{Z}_i^d = \begin{pmatrix} y_{i1} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & y_{i1} & y_{i2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & y_{i1} & y_{i2} & y_{i3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & y_{i1} & \cdots & y_{i,T-2} \end{pmatrix}$$

This extends naturally to $L > 1$ and $K > 1$; simply add columns to $\mathbf{X}_i^d$ and elements to $\boldsymbol{\gamma}$ as appropriate. When an observation is either missing or lost because of missing lags, delete the corresponding rows of $\mathbf{y}_i^d$, $\mathbf{X}_i^d$, $\boldsymbol{\eta}_i^d$, and $\mathbf{Z}_i^d$. Even if an observation is not missing with respect to the regression model, some of the lagged instruments might not be available because previous observations are missing. When that occurs, replace any missing instrument with 0.

When you specify the DYNDIFF option in the MODEL statement, PROC CPANEL by default treats x variables as exogenous and uses a projection that leaves these variables unchanged in the differenced regression. The full instrument matrix is then $\mathbf{Z}_i = (\mathbf{Z}_i^d, \mathbf{D}_i)$, where

$$\mathbf{D}_i = \begin{pmatrix} \Delta x_{i31} & \Delta x_{i32} & \cdots & \Delta x_{i3K} \\ \Delta x_{i41} & \Delta x_{i42} & \cdots & \Delta x_{i4K} \\ \vdots & \vdots & \vdots & \vdots \\ \Delta x_{iT1} & \Delta x_{iT2} & \cdots & \Delta x_{iT K} \end{pmatrix}$$

When $L = 1$, the default $\mathbf{Z}_i$ has $(T - 1)(T - 2)/2 + K$ columns. Each column $\mathbf{z}_c$ of $\mathbf{Z}_i$ satisfies the moment condition $E(\mathbf{z}_c' \boldsymbol{\eta}_i^d) = 0$.

System GMM

Blundell and Bond (1998) proposed a system GMM estimator that uses additional moment conditions to increase efficiency. The efficiency gain can be substantial when there is strong serial correlation in the dependent variable.

When either ϕ is near 1 or $\sigma_v^2/\sigma_\epsilon^2$ is large, the lagged dependent variables $y_{i,t-1}$ are weak instruments for the differenced variables Δy_{it} . System GMM solves the weak instrument problem by augmenting the difference equations described previously with a set of *level equations*. When $L = K = 1$, the level equations are

$$\mathbf{y}_i^\ell = \mathbf{X}_i^\ell \boldsymbol{\gamma} + \boldsymbol{\epsilon}_i^\ell$$

where

$$\mathbf{y}_i^\ell = \begin{pmatrix} y_{i2} \\ y_{i3} \\ \vdots \\ y_{iT} \end{pmatrix} \quad \mathbf{X}_i^\ell = \begin{pmatrix} y_{i1} & x_{i2} \\ y_{i2} & x_{i3} \\ \vdots & \vdots \\ y_{i,T-1} & x_{iT} \end{pmatrix} \quad \boldsymbol{\epsilon}_i^\ell = \begin{pmatrix} v_i + \epsilon_{i2} \\ v_i + \epsilon_{i3} \\ \vdots \\ v_i + \epsilon_{iT} \end{pmatrix}$$

Blundell and Bond (1998) note that you can use lagged differences of y as instruments for the levels of y . The main instrument matrix for the level equations is then

$$\mathbf{Z}_i^\ell = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \Delta y_{i2} & 0 & \cdots & 0 \\ 0 & 0 & \Delta y_{i3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta y_{i,T-1} \end{pmatrix}$$

where the first row corresponds to time $t = 2$. You can extend this to $L > 1$ and $K > 1$ by adding columns to $\mathbf{X}_i^\ell$ and elements to $\boldsymbol{\gamma}$ as appropriate. Higher-order lags require deletion of the leading rows of $\mathbf{y}_i^\ell$, $\mathbf{X}_i^\ell$, $\boldsymbol{\epsilon}_i^\ell$, and $\mathbf{Z}_i^\ell$.

Regression on the full system is obtained by stacking $\mathbf{y}_i^d$ and $\mathbf{y}_i^\ell$ to form $\mathbf{y}_i^s$, stacking $\mathbf{X}_i^d$ and $\mathbf{X}_i^\ell$ to form $\mathbf{X}_i^s$, and stacking $\boldsymbol{\eta}_i^d$ and $\boldsymbol{\epsilon}_i^\ell$ to form $\boldsymbol{\epsilon}_i^s$.

When you specify the DYNSSYS model option, the default instrument matrix for the full system is

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} \end{pmatrix}$$

Estimation

The estimation in this section assumes system GMM. To obtain difference GMM, restrict estimation to the rows that correspond to the difference equations.

The initial moment matrix is derived from the theoretical variance of the combined residuals and is expressed as $\mathbf{H}_{1i} = \text{diag}(\mathbf{G}_{1i}, \mathbf{G}_{2i})$, where

$$\mathbf{G}_{1i} = \begin{pmatrix} 1 & -0.5 & 0 & \cdots & 0 & 0 & 0 \\ -0.5 & 1 & -0.5 & \cdots & 0 & 0 & 0 \\ 0 & -0.5 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -0.5 & 0 \\ 0 & 0 & 0 & \cdots & -0.5 & 1 & -0.5 \\ 0 & 0 & 0 & \cdots & 0 & -0.5 & 1 \end{pmatrix}$$

and $\mathbf{G}_{2i}$ is 0.5 times the identity matrix.

Define the weighting matrix as

$$\mathbf{W}_1 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{1i} \mathbf{Z}_i \right)^{-1}$$

and define the projections as

$$\mathbf{P}_y = \sum_{i=1}^N \mathbf{Z}_i' \mathbf{y}_i^s; \quad \mathbf{P}_x = \sum_{i=1}^N \mathbf{Z}_i' \mathbf{X}_i^s$$

The one-step GMM estimate of $\boldsymbol{\gamma}$ is the weighted OLS estimator

$$\hat{\boldsymbol{\gamma}}_1 = \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_y$$

The variance of $\hat{\boldsymbol{\gamma}}_1$ is

$$\text{Var}(\hat{\boldsymbol{\gamma}}_1) = \hat{\sigma}_\epsilon^2 \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1}$$

where $\hat{\sigma}_\epsilon^2$ is the mean square error (MSE) that is derived solely from the difference equations, namely

$$\hat{\sigma}_\epsilon^2 = (M - K)^{-1} \sum_{i=1}^N \left(\mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_1 \right)' \left(\mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_1 \right)$$

The total number of observations, M , is equal to the number of observations for which the difference equations hold.

A disadvantage of $\hat{\boldsymbol{\gamma}}_1$ is its reliance on the theoretical basis of $\mathbf{H}_{1i}$. The two-step GMM estimate of $\boldsymbol{\gamma}$ replaces $\mathbf{H}_{1i}$ with a version that is obtained from the observed one-step residuals. Let $\mathbf{H}_{2i}$ be the outer product of $\hat{\boldsymbol{\epsilon}}_i^s = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_1$. Then

$$\hat{\boldsymbol{\gamma}}_2 = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_y$$

where

$$\mathbf{W}_2 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{2i} \mathbf{Z}_i \right)^{-1}$$

The variance of $\hat{\boldsymbol{\gamma}}_2$ is

$$\text{Var}(\hat{\boldsymbol{\gamma}}_2) = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1}$$

Robust variances are calculated by the sandwich method. The robust variance of $\hat{\boldsymbol{\gamma}}_1$ is

$$\text{Var}^r(\hat{\boldsymbol{\gamma}}_1) = \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_1 \mathbf{W}_2^{-1} \mathbf{W}_1 \mathbf{P}_x \left(\mathbf{P}_x' \mathbf{W}_1 \mathbf{P}_x \right)^{-1}$$

The robust variance of $\hat{\boldsymbol{\gamma}}_2$ is

$$\text{Var}^r(\hat{\boldsymbol{\gamma}}_2) = \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1} \mathbf{P}_x' \mathbf{W}_2 \mathbf{W}_3^{-1} \mathbf{W}_2 \mathbf{P}_x \left(\mathbf{P}_x' \mathbf{W}_2 \mathbf{P}_x \right)^{-1}$$

where

$$\mathbf{W}_3 = \left(\sum_{i=1}^N \mathbf{Z}_i' \mathbf{H}_{3i} \mathbf{Z}_i \right)^{-1}$$

and $\mathbf{H}_{3i}$ is the outer product of $\mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_2$.

Arellano and Bond (1991), among others, note that robust two-step variance estimators are biased. Windmeijer (2005) derived a bias-corrected variance of $\hat{\boldsymbol{\gamma}}_2$; you can obtain this correction by specifying the `BIASCORRECTED` option in the `MODEL` statement.

Define the one-step and two-step residuals as $\hat{\mathbf{e}}_{1i} = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_1$ and $\hat{\mathbf{e}}_{2i} = \mathbf{y}_i^s - \mathbf{X}_i^s \hat{\boldsymbol{\gamma}}_2$, respectively. Also define the projected two-step residual as

$$\mathbf{P}_e = \sum_{i=1}^N \mathbf{Z}_i' \hat{\mathbf{e}}_{2i}$$

Formulate the matrix $\mathbf{D}$ such that its k th column is $\mathbf{D}_k = \mathbf{V}_2 \mathbf{P}_x' \mathbf{W}_2 \mathbf{F}_k \mathbf{W}_2 \mathbf{P}_e$, where $\mathbf{V}_2 = \text{Var}(\hat{\boldsymbol{\gamma}}_2)$. The matrix $\mathbf{F}_k$ is the quadratic form

$$\mathbf{F}_k = \sum_{i=1}^N \mathbf{Z}_i' \left(\mathbf{x}_{ik} \hat{\mathbf{e}}_{1i}' + \hat{\mathbf{e}}_{1i} \mathbf{x}_{ik}' \right) \mathbf{Z}_i$$

where $\mathbf{x}_{ik}$ is the k th column of $\mathbf{X}_i^s$.

The Windmeijer (2005) bias-corrected variance is

$$\text{Var}^w(\hat{\boldsymbol{\gamma}}_2) = \mathbf{V}_2 + \mathbf{D} \mathbf{V}_2 + \mathbf{V}_2 \mathbf{D}' + \mathbf{D} \mathbf{V}_1^r \mathbf{D}'$$

where $\mathbf{V}_1^r$ is the robust variance estimate of $\hat{\boldsymbol{\gamma}}_1$.

Estimating the Intercept

The intercept term vanishes when you take first differences and is thus identified only in the level equations. If you specify the DYNDIFF option in the MODEL statement and your model includes an intercept, then PROC CPANEL fits the model by using system GMM with the (default) instrumentation

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{j}_i \end{pmatrix}$$

where $\mathbf{j}_i$ is a column of ones. Because all the level instruments are zero except the constant, parameter estimates other than the intercept are unaffected by the added level equations.

If you specify the DYNDIFF option in the MODEL statement and your model does not include an intercept, then the level equations are excluded from the estimation.

If you specify the DYNSSYS option in the MODEL statement, then there is no issue regarding the intercept. Under the default instrument specification, if $\mathbf{X}_i^\ell$ includes an intercept, then the level instruments include an added column of ones. That is,

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} & \mathbf{j}_i \end{pmatrix}$$

Customizing Instruments

When you specify the DYNSSYS option to perform system GMM, the default instrument matrix is

$$\mathbf{Z}_i = \begin{pmatrix} \mathbf{Z}_i^d & \mathbf{0} & \mathbf{D}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{Z}_i^\ell & \mathbf{0} & \mathbf{c}_i \end{pmatrix}$$

where $\mathbf{c}_i$ is a column of ones or, if you specify the NOINT option, $\mathbf{c}_i$ is $\mathbf{0}$.

You can override the default set of instruments by specifying the ENDOGENOUS, PREDTERMINED, and INSTRUMENTS statements. The following discussion assumes that you are performing system GMM by using the DYNSSYS option in the MODEL statement. When you specify the DYNDIFF option instead, any specification (except the constant $\mathbf{c}_i$) that pertains to the level equations is ignored.

Endogenous Variables

A variable x_{it} is endogenous if $E(x_{it} \epsilon_{is}) \neq 0$ for $s \leq t$ and 0 otherwise.

The ENDOGENOUS statement specifies a list of endogenous variables that are used to form instrument matrices. For the difference equations, the instruments are “GMM-style” and mirror the form that is used in $\mathbf{Z}_i^d$ for the dependent variable. Suppose that the model includes one lag of the dependent variable ($L = 1$). Specifying the variable X in the ENDOGENOUS statement adds the following instruments to the difference equations:

$$\mathbf{G}_i^d = \begin{pmatrix} x_{i1} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & x_{i1} & x_{i2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{i1} & x_{i2} & x_{i3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_{i1} & \cdots & x_{i, T-2} \end{pmatrix}$$

The first row corresponds to time $t = 3$. The instruments are in lagged levels.

For the level equations, the ENDOGENOUS statement produces instruments that mirror the form used in $\mathbf{Z}_i^\ell$ for the dependent variable. Suppose that the model includes one lag of the dependent variable ($L = 1$). Specifying the variable X in the ENDOGENOUS statement adds the following instruments to the level equations:

$$\mathbf{G}_i^\ell = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & \Delta x_{i2} & 0 & \cdots & 0 \\ 0 & 0 & \Delta x_{i3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta x_{i, T-1} \end{pmatrix}$$

The first row corresponds to time $t = 2$. Because the instruments are used for the level equations, they are in lagged differences.

By default, the ENDOGENOUS statement generates instruments for both the difference equations and the level equations. You can specify the EQ=DIFF option to generate instruments for only the difference equations. You can specify the EQ=LEVEL option to generate instruments for only the level equations.

The following code fits a dynamic panel model by using difference equations. It includes GMM-style instruments for both the dependent variable Sales and the regression variable Price.

```
proc cpanel data = mycas.a;
  id State Year;
  endogenous Sales Price / eq = diff;
  model Sales = Price PopDensity / dyndiff;
run;
```

Predetermined Variables

A variable x_{it} is predetermined if $E(x_{it}\epsilon_{is}) \neq 0$ for $s < t$ and 0 otherwise.

The PREDETERMINED statement specifies a list of predetermined variables that are used to form instrument matrices. For the difference equations, the instruments are similar to those that are formed for endogenous variables, except that each observation contains an extra instrument that reflects orthogonality in the current time period. If $L = 1$, specifying the variable X in the PREDETERMINED statement adds the following instruments to the difference equations:

$$\mathbf{P}_i^d = \begin{pmatrix} x_{i1} & x_{i2} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & x_{i1} & x_{i2} & x_{i3} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_{i1} & \cdots & x_{i, T-1} \end{pmatrix}$$

The first row corresponds to time $t = 3$.

For the level equations, the PREDETERMINED statement produces instruments that are similar to those in $\mathbf{Z}_i^\ell$, except that the lag is shifted up to reflect orthogonality in the current time period. If $L = 1$, specifying the variable X in the PREDETERMINED statement adds the following instruments to the level equations:

$$\mathbf{P}_i^\ell = \begin{pmatrix} \Delta x_{i2} & 0 & 0 & \cdots & 0 \\ 0 & \Delta x_{i3} & 0 & \cdots & 0 \\ 0 & 0 & \Delta x_{i4} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta x_{i, T} \end{pmatrix}$$

The first row corresponds to time $t = 2$.

By default, the **PREDETERMINED** statement generates instruments for both the difference equations and the level equations. You can specify the **EQ=DIFF** option to generate instruments for only the difference equations. You can specify the **EQ=LEVEL** option to generate instruments for only the level equations.

The following code fits a dynamic panel model by using difference equations. The instrument set includes GMM-style instruments for both the endogenous dependent variable **Sales** and the predetermined regression variable **Price**.

```
proc cpanel data = mycas.a;
  id State Year;
  endogenous Sales / eq = diff;
  predetermined Price / eq = diff;
  model Sales = Price PopDensity / dyndiff;
run;
```

Exogenous Variables (Standard Instruments)

Exogenous variables are uncorrelated with both the level residuals and the differenced residuals. If a regression variable is exogenous, you might want to include that variable in the instrument set as a standard instrument. The **INSTRUMENTS** statement specifies a list of variables to be included as standard instruments.

For the difference equations, the standard instruments form the matrix $\mathbf{D}_i$, as described in the section “[First Differencing](#)” on page 390. These variables are usually exogenous regressors that you want to preserve under the projection to the instrument space. Because these instruments belong to the difference equations, the variables are automatically differenced.

For the level equations, the standard instruments are left untransformed. They usually correspond to external instruments that are not part of the main regression but that can be used as instruments for the regression variables in levels. If $L = 1$, specifying the **X1** and **X2** variables in the **INSTRUMENTS** statement adds the following instruments to the level equations:

$$\mathbf{L}_i = \begin{pmatrix} x_{i21} & x_{i22} \\ x_{i31} & x_{i32} \\ \vdots & \vdots \\ x_{iT1} & x_{iT2} \end{pmatrix}$$

The first row corresponds to time $t = 2$.

By default, the **INSTRUMENTS** statement generates instruments for both the difference equations and the level equations. You can specify the **EQ=DIFF** option to generate instruments for only the difference equations. You can specify the **EQ=LEVEL** option to generate instruments for only the level equations.

The following example illustrates how you would use the **ENDOGENOUS** and **INSTRUMENTS** statements to obtain the default set of instruments for system GMM:

```
proc cpanel data = mycas.a;
  id State Year;
  endogenous Sales / eq = both;
  instruments Price PopDensity / eq = diff;
  model Sales = Price PopDensity / dynsys;
run;
```

Note that the standard instruments for **Price** and **PopDensity** are included only in the difference equations.

Limiting the Number of Instruments

Arellano and Bond's (1991) technique of expanding instruments is a useful method of dealing with autocorrelation in the response variable. However, using too many instruments can bias the estimator. The number of instruments grows quadratically with the number of time periods, making computations less feasible for larger T .

By default, PROC CPANEL uses all available lags. You can limit the number of instruments by specifying the MAXBAND= option in the MODEL statement. For example, specifying MAXBAND=5 limits the number of GMM-style instruments to five per observation, for each variable. The MAXBAND= option applies to all GMM-style instruments: those from the ENDOGENOUS statement and those from the PREDETERMINED statement.

Sargan Test of Overidentifying Restrictions

A Sargan test is a referendum on your choice of instruments in a dynamic panel model. The Sargan test statistic for one-step GMM is

$$J = \frac{1}{\hat{\sigma}_\epsilon^2} \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{1i} \right)' \mathbf{W}_1 \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{1i} \right)$$

The Sargan test statistic for two-step GMM is

$$J = \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{2i} \right)' \mathbf{W}_2 \left(\sum_{i=1}^N \mathbf{Z}_i' \hat{\epsilon}_{2i} \right)$$

The null hypothesis of the Sargan test is that the moment conditions (as defined by the columns $\mathbf{Z}_i$) hold, and thus $\mathbf{Z}_i$ form an adequate set of instruments. Under the null, J is distributed as χ^2 with degrees of freedom equal to the rank of $\mathbf{W}_c$ ($c = 1, 2$) minus the number of parameters K . The nominal rank of $\mathbf{W}_c$ is equal to the number of instruments. However, this number can be reduced because of collinearity and redundancy in the instrument specification. Furthermore, the maximum rank of $\mathbf{W}_2$ is N , regardless of the number of instruments.

You should treat Sargan tests with caution when robust variances are used in the estimation. The theoretical distribution of J does not hold under conditions that favor robust variances.

AR(m) Tests

An AR(m) test is a test for autocorrelation of order m in the model residuals. Let $\mathbf{R}_i^s$ be the working variance of the residuals from the full system. The precise definition of $\mathbf{R}_i^s$ depends on the GMM stage and whether robust variances are specified. Table 9.4 provides detailed definitions.

Table 9.4 Definitions of the Working Residual Variance

Estimator	$\mathbf{R}_i^s$
One-step	$\hat{\sigma}_\epsilon^2 \mathbf{H}_{1i}$
One-step, robust	$\mathbf{H}_{2i}$
Two-step	$\mathbf{H}_{2i}$
Two-step, robust	$\mathbf{H}_{3i}$

Define the residual vector

$$\hat{\mathbf{e}}_i = \begin{pmatrix} \hat{\eta}_i^d \\ \mathbf{0} \end{pmatrix}$$

where $\hat{\eta}_i^d = \mathbf{y}_i^d - \mathbf{X}_i^d \hat{\boldsymbol{\gamma}}_c$ are the residuals from the difference equations, evaluated at the final estimate of $\hat{\boldsymbol{\gamma}}_c$. The trailing zeros correspond to the level equations. Define $\hat{\boldsymbol{\omega}}_{mi}$ as a lagged version of $\hat{\mathbf{e}}_i$ such that the following statements are true:

1. The first m elements of $\hat{\boldsymbol{\omega}}_{mi}$ are 0.
2. The next $p - m$ elements of $\hat{\boldsymbol{\omega}}_{mi}$ are the first $p - m$ elements of $\hat{\mathbf{e}}_i$, where p is the number of difference equations.
3. The trailing elements of $\hat{\boldsymbol{\omega}}_{mi}$ that correspond to the level equations are 0.

Define the following:

$$\begin{aligned} \mathbf{P}_m &= \sum_{i=1}^N \mathbf{Z}_i' \mathbf{R}_i^s \hat{\boldsymbol{\omega}}_{mi} \\ \mathbf{Q}_m &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \mathbf{X}_i^s \end{aligned}$$

The AR(m) test statistic is $Z_m = k_{0m} \{k_{1m} + k_{2m} + k_{3m}\}^{-1/2}$, where

$$\begin{aligned} k_{0m} &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \hat{\mathbf{e}}_i \\ k_{1m} &= \sum_{i=1}^N \hat{\boldsymbol{\omega}}_{mi}' \mathbf{R}_i^s \hat{\boldsymbol{\omega}}_{mi} \\ k_{2m} &= -2\mathbf{Q}_m (\mathbf{P}_x' \mathbf{W}_c \mathbf{P}_x)^{-1} \mathbf{P}_x' \mathbf{W}_c \mathbf{P}_m \\ k_{3m} &= \mathbf{Q}_m \mathbf{V} \mathbf{Q}_m' \end{aligned}$$

The matrix $\mathbf{V}$ is the estimated variance matrix of the parameters, corresponding to the GMM stage that is specified, and either model-based, robust, or bias-corrected.

Under the null hypothesis of no autocorrelation, Z_m follows a standard normal distribution. Because of the differencing in the errors, well-specified models present autocorrelation of order $m = 1$, but any autocorrelation at higher orders indicates a violation of assumptions.

Poolability Test for Fixed Effects

A poolability test is an F test of the null hypothesis that all fixed effects are jointly 0; it is obtained by comparing fixed-effects estimates to those from pooled regression. The F statistic is

$$F = \frac{(\text{SSE}_r - \text{SSE}_u) / df_1}{\text{SSE}_u / df_2} \sim F(df_1, df_2)$$

where SSE_r is the error sum of squares from the restricted model (pooled regression) and SSE_u is the error sum of squares from the unrestricted fixed-effects model.

The numerator degrees of freedom, df_1 , equals $N - 1$ for one-way models and $(N - 1) + (T - 1)$ for two-way models. The denominator degrees of freedom, df_2 , is equal to the error degrees of freedom from the fixed-effects estimation. If you specify the NOINT option, add 1 to df_1 to account for the added restriction to the pooled regression.

Hausman Specification Tests

For models that include random effects, the CPANEL procedure outputs the results of the Hausman (1978) specification test. This test was also proposed by Wu (1973) and further extended in Hausman and Taylor (1982).

Consider two estimators, $\hat{\beta}_e$ and $\hat{\beta}_c$, which under the null hypothesis are both consistent but only $\hat{\beta}_e$ is asymptotically efficient. Under the alternative hypothesis, only $\hat{\beta}_c$ is consistent. The m statistic is

$$m = (\hat{\beta}_c - \hat{\beta}_e)' (\hat{\Sigma}_c - \hat{\Sigma}_e)^{-1} (\hat{\beta}_c - \hat{\beta}_e)$$

where $\hat{\Sigma}_c$ and $\hat{\Sigma}_e$ are estimates of the asymptotic covariance matrices of $\hat{\beta}_c$ and $\hat{\beta}_e$. The statistic m follows a χ^2 distribution with k degrees of freedom, where k is the rank of $(\hat{\Sigma}_c - \hat{\Sigma}_e)^{-1}$. This rank is normally equal to the dimension of $\hat{\beta}_c - \hat{\beta}_e$, but is reduced when regressors that are constant within cross sections are dropped from the fixed-effects model.

The null hypothesis is that the effects are independent of the regressors. Under the null hypothesis, the fixed-effects estimator is consistent yet inefficient, whereas the random-effects estimator is both consistent and efficient. Failure to reject the null hypothesis favors the random-effects specification.

Restricted Estimation

The CPANEL procedure can fit models that have linear restrictions, producing a Lagrange multiplier (LM) test for each restriction. Consider a set of J linear restrictions $\mathbf{R}\beta = \mathbf{q}$, where $\mathbf{R}$ is $J \times K$ and $\mathbf{q}$ is $J \times 1$.

The restricted regression is performed by minimizing the error sum of squares subject to the restrictions. In matrix terms, the Lagrangian for this problem is

$$L = (\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) + 2\lambda(\mathbf{R}\beta - \mathbf{q})$$

The Lagrangian is minimized by the restricted estimator β^* , and it can be shown that

$$\beta^* = \hat{\beta} - (\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}'\lambda$$

where $\hat{\beta}$ is the unrestricted estimator.

Because $\mathbf{R}\beta^* = \mathbf{q}$, you can solve for λ to obtain the Lagrange multipliers

$$\lambda^* = \left[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}' \right]^{-1} (\mathbf{R}\hat{\beta} - \mathbf{q})$$

The standard errors of the Lagrange multipliers are the square roots of the diagonal elements of the variance matrix

$$\text{Var}(\lambda^*) = \hat{\sigma}_e^2 \left[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}' \right]^{-1}$$

where $\hat{\sigma}_e^2$ is the mean square error (MSE) under the null hypothesis. A significant Lagrange multiplier indicates a restriction that is binding.

For each restriction, the Lagrange multiplier test is represented by a row in the parameter estimates table that displays the Lagrange multiplier, standard error, test statistic, and p -value. Although LM test statistics converge to a chi-square distribution under the null hypothesis, PROC CPANEL computes p -values by using the beta distribution, which is the exact distribution for finite samples.

Lagrange multiplier tests are not appropriate for instrumental variables regression and dynamic panel models. If you fit such a model, the linear restrictions are applied, but no formal test of their validity is performed.

Linear Hypothesis Testing

Consider a linear hypothesis of the form $\mathbf{R}\beta = \mathbf{q}$, where $\mathbf{R}$ is $J \times K$ and $\mathbf{q}$ is $J \times 1$. The Wald test statistic is

$$\chi_W^2 = (\mathbf{R}\hat{\beta} - \mathbf{q})' (\mathbf{R}\hat{\mathbf{V}}\mathbf{R}')^{-1} (\mathbf{R}\hat{\beta} - \mathbf{q})$$

where $\hat{\mathbf{V}}$ is the estimated variance of $\hat{\beta}$.

In simple linear models, the Wald test statistic is equal to the F test statistic

$$F = \frac{(\text{SSE}_r - \text{SSE}_u)/J}{\text{SSE}_u/df_e}$$

where SSE_r is the restricted error sum of squares, SSE_u is the unrestricted error sum of squares, and df_e is the unrestricted error degrees of freedom.

The F statistic represents a more direct comparison of the restricted model to the unrestricted model. Comparing error sums of squares is appealing in complex models for which restrictions are applied not only during the final regression but also during intermediate calculations.

The likelihood ratio (LR) test and the Lagrange multiplier (LM) test are derived from the F statistic. The LR test statistic is

$$\chi_{\text{LR}}^2 = M \ln \left[1 + \frac{JF}{M - K} \right]$$

The LM test statistic is

$$\chi_{\text{LM}}^2 = M \left[\frac{JF}{M - K + JF} \right]$$

The distribution of these test statistics is χ^2 with J degrees of freedom. The three tests are asymptotically equivalent, but they possess different small-sample properties. For more information, see Greene (2000, p. 392) and Davidson and MacKinnon (1993, pp. 456–458).

Heteroscedasticity-Corrected Covariance Matrices

The HCCME= option in the MODEL statement selects the type of heteroscedasticity-consistent covariance matrix. In the presence of heteroscedasticity, the covariance matrix has a complicated structure that can result in inefficiencies in the OLS estimates and biased estimates of the covariance matrix. The variances for cross-sectional and time dummy variables and the covariances with or between the dummy variables are not corrected for heteroscedasticity in the one-way and two-way models. Whether or not you specify the HCCME= option, these variances are the same. For the two-way models, the variance and the covariances for the intercept are not corrected.¹

Consider the following simple linear model:

$$y = X\beta + \epsilon$$

This discussion parallels the discussion in Davidson and MacKinnon (1993, pp. 548–562). For panel data models, heteroscedasticity-corrected covariance matrix estimation (HCCME) is applied to the transformed data ($\tilde{y}$ and $\tilde{X}$). In other words, first the random or fixed effects are removed through transforming the data,² and then the heteroscedasticity (also autocorrelation with the HAC option) is corrected in the residual. The assumptions that make the linear regression best linear unbiased estimator (BLUE) are $E(\epsilon) = 0$ and $E(\epsilon\epsilon') = \Omega$, where Ω has the simple structure $\sigma^2 I$. Heteroscedasticity results in a general covariance structure, and it is not possible to simplify Ω . The result is as follows:

$$\tilde{\beta} = (X'X)^{-1}X'y = (X'X)^{-1}X'(X\beta + \epsilon) = \beta + (X'X)^{-1}X'\epsilon$$

As long as the following is true, you are assured that the OLS estimate is consistent and unbiased:

$$\text{plim}_{n \rightarrow \infty} \left(\frac{1}{n} X' \epsilon \right) = 0$$

If the regressors are nonrandom, then it is possible to write the variance of the estimated β as

$$\text{Var}(\beta - \tilde{\beta}) = (X'X)^{-1}X'\Omega X(X'X)^{-1}$$

You can ameliorate the effect of structure in the covariance matrix by using generalized least squares (GLS), provided that Ω^{-1} can be calculated. Using Ω^{-1} , you premultiply both sides of the regression equation,

$$L^{-1}y = L^{-1}X\epsilon + L^{-1}\epsilon$$

¹The dummy variables are removed by the within transformations, so their variances and covariances cannot be calculated the same way as the other regressors. They are recovered by the formulas in the sections “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 372 and “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 374. The formulas assume homoscedasticity, so they do not apply when HCCME is used. Therefore, standard errors, variances, and covariances are reported only when the HCCME= option is ignored. HCCME standard errors for dummy variables and intercept can be calculated by the dummy variable approach with the pooled model.

²For more information about transforming the data, see the sections “One-Way Fixed-Effects Model (FIXONE and FIXONETIME Options)” on page 372, “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 374, “One-Way Random-Effects Model (RANONE Option)” on page 375, and “Two-Way Random-Effects Model (RANTWO Option)” on page 378.

where L denotes the Cholesky root of Ω (that is, $\Omega = LL'$ with L lower triangular).

The resulting GLS β is

$$\hat{\beta} = (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\mathbf{y}$$

Using the GLS β , you can write

$$\begin{aligned}\hat{\beta} &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\mathbf{y} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'(\Omega^{-1}\mathbf{X}_e + \Omega^{-1}\epsilon) \\ &= \beta + (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\epsilon\end{aligned}$$

The resulting variance expression for the GLS estimator is

$$\begin{aligned}\text{Var}(\beta - \hat{\beta}) &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\epsilon\epsilon'\Omega^{-1}\mathbf{X}(\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\mathbf{X}'\Omega^{-1}\Omega\Omega^{-1}\mathbf{X}(\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1} \\ &= (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}\end{aligned}$$

The difference in variance between the OLS estimator and the GLS estimator can be written as

$$(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\Omega\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} - (\mathbf{X}'\Omega^{-1}\mathbf{X})^{-1}$$

By the Gauss-Markov theorem, the difference matrix must be positive definite under most circumstances (zero if OLS and GLS are the same, when the usual classical regression assumptions are met). Thus, OLS is not efficient for a general error structure. It is crucial to realize that OLS does not produce biased results. It would suffice if you had a method of estimating a consistent covariance matrix and you used the OLS β . Estimation of the Ω matrix is certainly not simple. The matrix is square and has M^2 elements; unless some sort of structure is assumed, it becomes an impossible problem to solve. However, the heteroscedasticity can have quite a general structure. White (1980) shows that it is not necessary to have a consistent estimate of Ω . On the contrary, it suffices to calculate an estimate of the middle expression. That is, you need an estimate of

$$\Lambda = \mathbf{X}'\Omega\mathbf{X}$$

This matrix, Λ , is easier to estimate because its dimension is K . PROC CPANEL provides the following classical HCCME estimators for Λ . The approximation of the matrix depends on the value of the HCCME= option.

- HCCME=0

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}'$$

Here N is the number of cross sections and T_i is the number of observations in the i th cross section. The $\mathbf{x}_{it}'$ is from the t th observation in the i th cross section, constituting the $(\sum_{j=1}^{i-1} T_j + t)$ th row of the matrix $\mathbf{X}$. If you specify the CLUSTER option, one extra term is added to the preceding equation so that the estimator of matrix Λ is

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}' + \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \hat{\epsilon}_{it} \hat{\epsilon}_{is} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

The formula is the same as the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152), and it is derived under the assumptions of section 7.3.2 of Wooldridge (2002). You can also obtain the preceding clustered standard error with HCCME=0 by specifying the ROBUST option alone.

- HCCME=1

$$\frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}'$$

Here M is the total number of observations, $\sum_{j=1}^N T_j$, and K is the number of parameters. If you specify the CLUSTER option, the estimator becomes

$$\frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \hat{\epsilon}_{it}^2 \mathbf{x}_{it} \mathbf{x}_{it}' + \frac{M}{M-K} \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \hat{\epsilon}_{it} \hat{\epsilon}_{is} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment term $M/(M-K)$.

- HCCME=2

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{1 - \hat{h}_{it}} \mathbf{x}_{it} \mathbf{x}_{it}'$$

The $\hat{h}_{it}$ term is the $(\sum_{j=1}^{i-1} T_j + t)$ th diagonal element of the hat matrix. The expression for $\hat{h}_{it}$ is $\mathbf{x}_{it}'(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_{it}$. The hat matrix attempts to adjust the estimates for the presence of influence or leverage points. If you specify the CLUSTER option, the estimator becomes

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{1 - \hat{h}_{it}} \mathbf{x}_{it} \mathbf{x}_{it}' + 2 \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \frac{\hat{\epsilon}_{it}}{\sqrt{1 - \hat{h}_{it}}} \frac{\hat{\epsilon}_{is}}{\sqrt{1 - \hat{h}_{is}}} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment.

- HCCME=3

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{(1 - \hat{h}_{it})^2} \mathbf{x}_{it} \mathbf{x}_{it}'$$

If you specify the CLUSTER option, the estimator becomes

$$\sum_{i=1}^N \sum_{t=1}^{T_i} \frac{\hat{\epsilon}_{it}^2}{(1 - \hat{h}_{it})^2} \mathbf{x}_{it} \mathbf{x}_{it}' + 2 \sum_{i=1}^N \sum_{t=1}^{T_i} \sum_{s=1}^{t-1} \frac{\hat{\epsilon}_{it}}{1 - \hat{h}_{it}} \frac{\hat{\epsilon}_{is}}{1 - \hat{h}_{is}} (\mathbf{x}_{it} \mathbf{x}_{is}' + \mathbf{x}_{is} \mathbf{x}_{it}')$$

This formula is similar to the formula for the robust variance matrix estimator in Wooldridge (2002, p. 152) with the heteroscedasticity adjustment.

In their discussion, Davidson and MacKinnon (1993, p. 554) argue that the formula specified by HCCME=1 should always be preferred to the formula specified by HCCME=0. Although an HCCME= option value of 3 is generally preferred to 2, and 2 is preferred to 1, the calculation of HCCME=1 is as simple as the calculation of HCCME=0. Therefore, HCCME=1 is preferred when the calculation of the hat matrix is too tedious.

All HCCMEs have well-defined asymptotic properties. The small-sample properties are not well known, and you must exercise care when sample sizes are small.

The HCCME of $\text{Var}(\boldsymbol{\beta})$ is used to drive the covariance matrices for the fixed effects and the Lagrange multiplier standard errors. Robust estimates of the covariance matrix for $\boldsymbol{\beta}$ imply robust covariance matrices for all other parameters.

Heteroscedasticity- and Autocorrelation-Consistent Covariance Matrices

You use the HAC option in the MODEL statement to select the type of heteroscedasticity- and autocorrelation-consistent covariance matrix. As with the HCCME= option, an estimator of the middle expression Λ in sandwich form is needed. Define $g_{i,t} = x_{it}\epsilon_{it}$ as the gradient vector of the i th observation in the i th cross section. For the HAC option, it is estimated as

$$\Lambda_{\text{HAC}} = a \sum_{i=1}^N \left(\sum_{t=1}^T g_{i,t} g'_{i,t} + \sum_{j=1}^{T-1} k\left(\frac{j}{b_i}\right) \sum_{t=1}^{T-j} (g_{i,t} g'_{i,t+j} + g_{i,t+j} g'_{i,t}) \right)$$

where $k(\cdot)$ is the real-valued kernel function, b_i is the bandwidth parameter for the i th cross section, and a is the adjustment factor of small-sample degrees of freedom (that is, $a = 1$ if you do not specify the ADJUSTDF option, and $a = NT/(NT - k)$ otherwise, where k is the number of parameters, including dummy variables). If the BANDWIDTH option is specified as a number, b_i is equal to the same number for every cross section. The types of kernel functions are listed in Table 9.5.

Table 9.5 Kernel Functions

Kernel Type	Equation
Bartlett	$k(x) = \begin{cases} 1 - x & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Parzen	$k(x) = \begin{cases} 1 - 6x^2 + 6 x ^3 & 0 \leq x \leq 1/2 \\ 2(1 - x)^3 & 1/2 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Quadratic spectral	$k(x) = \frac{25}{12\pi^2 x^2} \left(\frac{\sin(6\pi x/5)}{6\pi x/5} - \cos(6\pi x/5) \right)$
Truncated	$k(x) = \begin{cases} 1 & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
Tukey-Hanning	$k(x) = \begin{cases} (1 + \cos(\pi x))/2 & x \leq 1 \\ 0 & \text{otherwise} \end{cases}$

When you specify the BANDWIDTH=ANDREWS option, the bandwidth parameter is estimated as shown in Table 9.6.

Table 9.6 Bandwidth Parameter Estimation

Kernel Type	Bandwidth Parameter
Bartlett	$b_i = 1.1447(\alpha(1)T)^{1/3}$
Parzen	$b_i = 2.6614(\alpha(2)T)^{1/5}$
Quadratic spectral	$b_i = 1.3221(\alpha(2)T)^{1/5}$
Truncated	$b_i = 0.6611(\alpha(2)T)^{1/5}$
Tukey-Hanning	$b_i = 1.7462(\alpha(2)T)^{1/5}$

The values $\alpha(1)$ and $\alpha(2)$ are calculated separately for each cross section by using the following steps. Let g_{ait} denote the a th row of g_{it} , and let (ρ_a, σ_a^2) denote the corresponding estimates of the autoregressive and innovation variance parameters of the AR(1) model on $\{g_{ait}\}$, $a = 1, \dots, k$, where the AR(1) model is parameterized as $g_{ait} = \rho_a g_{ait-1} + \epsilon_{ait}$ with $\text{Var}(\epsilon_{ait}) = \sigma_a^2$. Notice that ρ_a and σ_a^2 are different for each cross section. Obtain $\alpha(1)$ and $\alpha(2)$ by using

$$\alpha(1) = \frac{\sum_{a=1}^k \frac{4\rho_a^2 \sigma_a^4}{(1-\rho_a)^6(1+\rho_a)^2}}{\sum_{a=1}^k \frac{\sigma_a^4}{(1-\rho_a)^4}} \quad \alpha(2) = \frac{\sum_{a=1}^k \frac{4\rho_a^2 \sigma_a^4}{(1-\rho_a)^8}}{\sum_{a=1}^k \frac{\sigma_a^4}{(1-\rho_a)^4}}$$

When you specify BANDWIDTH=NEWKEYWEST94, according to Newey and West (1994) the bandwidth parameter is estimated as shown in Table 9.7.

Table 9.7 Bandwidth Parameter Estimation

Kernel Type	Bandwidth Parameter
Bartlett	$b_i = 1.1447(\{s_1/s_0\}^2 T)^{1/3}$
Parzen	$b_i = 2.6614(\{s_1/s_0\}^2 T)^{1/5}$
Quadratic spectral	$b_i = 1.3221(\{s_1/s_0\}^2 T)^{1/5}$
Truncated	$b_i = 0.6611(\{s_1/s_0\}^2 T)^{1/5}$
Tukey-Hanning	$b_i = 1.7462(\{s_1/s_0\}^2 T)^{1/5}$

The terms s_0 and s_1 are estimated separately for each cross section by using

$$s_0 = \sigma_0 + 2 \sum_{j=1}^n \sigma_j \quad s_1 = 2 \sum_{j=1}^n j \sigma_j$$

where n is the lag selection parameter and is determined by kernels for each cross section, as listed in Table 9.8.

Table 9.8 Lag Selection Parameter Estimation

Kernel Type	Lag Selection Parameter
Bartlett	$n = c(T/100)^{2/9}$
Parzen	$n = c(T/100)^{4/25}$

Table 9.8 *continued*

Kernel Type	Lag Selection Parameter
Quadratic spectral	$n = c(T/100)^{2/25}$
Truncated	$n = c(T/100)^{1/5}$
Tukey-Hanning	$n = c(T/100)^{1/5}$

The c in Table 9.8 is specified by the C= option; by default, C=12 for all cross sections.

The σ_j are estimated for the i th cross section by the equation

$$\sigma_j = T^{-1} \sum_{t=j+1}^T \left(\sum_{a=s}^k g_{ait} \sum_{a=s}^k g_{ait-j} \right), \quad j = 0, \dots, n$$

where g_{ait} is the same as in the Andrews method and s is 1 if you specify the NOINT option in the MODEL statement, and 2 otherwise.

When you specify BANDWIDTH=SAMPLESIZE, the bandwidth parameter is estimated by the equation

$$b_i = \begin{cases} \lfloor \gamma T^r + c \rfloor & \text{if the BANDWIDTH=SAMPLESIZE(INTSS) option is specified} \\ \gamma T^r + c & \text{otherwise} \end{cases}$$

where T is the sample size; $\lfloor x \rfloor$ is the largest integer less than or equal to x ; and γ , r , and c are values specified by the BANDWIDTH=SAMPLESIZE(GAMMA= RATE= CONSTANT=) options, respectively. These specified values are the same for every cross section.

If you specify the PREWHITENING option in the MODEL statement, g_{it} is prewhitened by the VAR(1) model,

$$g_{it} = A_i g_{it-1} + w_{it}$$

Then Λ_{HAC} is calculated by

$$\Lambda_{\text{HAC}} = a \sum_{i=1}^N \left\{ \left(\sum_{t=1}^T w_{it} w'_{it} + \sum_{t=1}^T \sum_{s=1}^{t-1} k\left(\frac{s-t}{b_i}\right) (w_{it} w'_{is} + w_{is} w'_{it}) \right) (I - A_i)^{-1} ((I - A_i)^{-1})' \right\}$$

For unbalanced data, the HAC option treats those missing values as 0.

R-Square

The R-square statistic is the proportion of variability in the dependent variable that is attributed to the independent variables. Because of the transformations that are used prior to fitting the final regression model, the conventional R-square measure is not appropriate for most of the models that the CPANEL procedure supports. In random-effects models that use a GLS transform, PROC CPANEL calculates the modified R-square statistic proposed by Buse (1973),

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}'\mathbf{D}'\hat{\mathbf{\Omega}}^{-1}\mathbf{D}\mathbf{y}}$$

where SSE is the error sum of squares from the final model fit, $\hat{\mathbf{\Omega}}^{-1/2}$ represents the GLS transform, and $\mathbf{D} = \mathbf{I}_M - a^{-1}\mathbf{J}_M\hat{\mathbf{\Omega}}^{-1}$, for $a = \mathbf{j}_M'\hat{\mathbf{\Omega}}^{-1}\mathbf{j}_M$.

In GLS models that do not have an intercept, the alternative R-square measure, which is attributed to Theil (1961), is calculated as follows:

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}'\hat{\mathbf{\Omega}}^{-1}\mathbf{y}}$$

In fixed-effects models, the R-square measure is

$$R^2 = 1 - \frac{\text{SSE}}{\mathbf{y}_w'\mathbf{y}_w}$$

where $\mathbf{y}_w$ is the within-transformed dependent variable.

In the case of pooled OLS estimation, all three of the R-square formulas reduce to the usual R-square statistic for linear models.

The R-square statistic is not appropriate for IV regression and dynamic panel estimation, and thus it is not reported for these models.

Displayed Output

The following sections describe the output that PROC CPANEL produces. The output is organized into various tables, which are discussed in their order of appearance.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the **CLASS** statement. You should check this information to make sure that the data are correct. You can adjust the order of the **CLASS** variable levels by using the **ORDER=** option in the **CLASS** statement.

Model Description

The “Model Information” table displays basic information about the model: the response variable, the name of the data table, the estimation method, the number of observations, the number of cross sections, and the length of the time series. The reported numbers are useful for verifying that observations were not dropped unexpectedly and for verifying that the data are balanced when they should be.

Fit Statistics

The “Fit Statistics” table displays model fit statistics that are widely used in linear models. This table reports the error sum of squares, mean square error, root mean square error, R-square statistic, and the error degrees of freedom.

Variance Component Estimates

If you fit a model that contains random effects, then the “Variance Component Estimates” table reports the variances and standard deviations of the observation-level errors, the cross-sectional effects, and the time effects (in two-way models).

Test for Poolability

If you fit a fixed-effects model, then an F test of poolability is produced. The null hypothesis is that all fixed effects are jointly 0, making pooled OLS regression a viable alternative. The “F Test for No Fixed Effects” table reports the numerator degrees of freedom, denominator degrees of freedom, test statistic, and p -value.

Hausman Specification Tests

If you fit a model that contains random effects, then a table that summarizes the Hausman test is produced. The table presents the number of coefficients included in the test, the degrees of freedom, the m -statistic, and the p -value.

If you fit a one-way random-effects model (RANONE), then the fitted model is compared to the alternative model, a one-way fixed-effects (FIXONE) model.

If you fit a two-way random-effects model (RANTWO), then the fitted model is compared to the alternative model, a two-way fixed-effects (FIXTWO) model.

If you fit a Hausman-Taylor model (HTAYLOR), then the fitted model is compared to the alternative model, a one-way fixed-effects (FIXONE) model.

If you fit an Amemiya-MaCurdy model (AMACURDY), then the fitted model is compared to the alternative model, a Hausman-Taylor (HTAYLOR) model.

If you fit an instrumental variables random-effects model (IVRANONE), then the fitted model is compared to the alternative model, an IV fixed-effects (IVFIXONE) model.

Structural Summary

If you fit an instrumental variables (IV) model, then the “Structural Summary” table lists the endogenous effects and the instrumental variables.

Instruments

If you fit a dynamic panel model, then the “Instruments” table lists the instruments that are used and tallies the columns of the instruments matrix. The instruments are categorized by the type of equation (difference or level) and by the type of variable (endogenous, predetermined, or exogenous).

Sargan Test

If you fit a dynamic panel model, then the CPANEL procedure computes a Sargan test of overidentifying restrictions. This test is a referendum on your choice of instruments, and the null hypothesis is that your choice is adequate. The “Sargan Test” table reports the degrees of freedom, test statistic, and p -value.

Parameter Estimates

The parameter estimates, degrees of freedom, standard errors, t statistics and p -values for the hypothesis that the parameter is 0 are presented in the “Parameter Estimates” table. If one or more of your regression variables are labeled, then a column that contains variable labels is added to this table. Use the [NOLABEL](#) option in the MODEL statement to suppress this column.

If you specify a RESTRICT statement, then the table includes a row for each linear restriction. This row reports the restriction label, Lagrange multiplier (LM) statistic, standard error, t statistic, p -value, and a label that describes the restriction.

If you fit an instrumental variables (IV) model, then the table includes a column that identifies the endogenous variables. If you fit a Hausman-Taylor or Amemiya-MaCurdy model, then both the correlated variables and the time-invariant variables are identified in the table.

AR(m) Tests

If you fit a dynamic panel model, then the CPANEL procedure performs tests for autocorrelation in the residuals. In well-fitting models, you can expect some autocorrelation of lag order 1, but any autocorrelation at higher lags indicates a violation of assumptions. By default, AR tests for the first two lags are reported, but you can request tests for more lags by specifying the ARTESTS= option in the MODEL statement. For each test, the “AR(m) Test” table reports the lag, test statistic, and p -value.

Parameter Estimates Covariance Matrix

When you specify the [COVB](#) option in the [PROC CPANEL](#) statement, the CPANEL procedure displays the covariance matrix of the parameter estimates.

Parameter Estimates Correlation Matrix

When you specify the **CORRB** option in the **PROC CPANEL** statement, the CPANEL procedure displays the correlation matrix of the parameter estimates.

Tests of Linear Hypotheses

If you specify a **TEST** statement, a table that describes the tests is provided. For each test of hypothesis, the table reports the test label, the type of test (Wald, LR, or LM), the test statistic, the *p*-value, and a label that states the null hypothesis.

Model Comparison Tables

If you specify a **COMPARE** statement, two comparison tables are produced.

The first table is the “Comparison of Model Statistics” table, which lists the number of observations, number of cross sections, time series length, mean square error (MSE), root MSE, and R-square for each fitted model.

The second is table is the “Comparison of Model Parameter Estimates” table, which presents the parameter estimate and standard error of each regressor for each fitted model.

Both tables can be customized; for more information, see the section “**COMPARE Statement**” on page 350.

ODS Table Names

Each table that the CPANEL procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. These names are listed in [Table 9.9](#).

Table 9.9 ODS Tables Produced by PROC CPANEL

Table Name	Description	Statement	Option
ARTest	Autoregression tests for the residuals	MODEL	DYNDIFF or DYN SYS
ClassInfo	Level information from the CLASS statement	CLASS	Default
CorrB	Correlation matrix of parameter estimates	MODEL	CORRB
CovB	Covariance matrix of parameter estimates	MODEL	COVB
FitStatistics	Fit statistics	Default	
FixedEffectsTest	<i>F</i> test for poolability	MODEL	FIXONE, FIXONETIME, FIXTWO, or IVFIXONE
HausmanTestForRandom	Hausman test for random effects	MODEL	RANONE, RANTWO, IVRANONE, or IVRANTWO

Table 9.9 *continued*

Table Name	Description	Statement	Option
HausmanTestVsFixed	Hausman test for Hausman-Taylor models	MODEL	HTAYLOR
HausmanTestVsHausmanTaylor	Hausman test for Amemiya-MaCurdy models	MODEL	AMACURDY
Instruments	Instruments summary	MODEL	DYNDIFF or DYSYS
ModelDescription	Model summary and data counts	Default	
ParameterComparisonTable	Comparison of model parameter estimates, standard errors, and <i>t</i> tests	COMPARE	
ParameterEstimates	Solutions for the parameter estimates associated with effects in MODEL statements	Default	
ParameterEstimatesLegend	Legend that defines the abbreviations that were used in the parameter estimates table	Default	
SarganTest	Sargan test for overidentifying restrictions	MODEL	DYNDIFF or DYSYS
StatComparisonTable	Comparison of model summary statistics	COMPARE	
StructuralSummary	Endogenous and instrumental variable lists	MODEL	IVPOOLED, IVBTWNG, IVFIXONE, IV-FIXONETIME, IVFIXTWO, IVG2SLS, IVRANONE, or IVRANTWO
TestResults	Hypothesis tests of linear restrictions	TEST	
VarianceComponents	Variance component estimates	MODEL	RANONE, RANTWO, HTAYLOR, AMACURDY, IVRANONE, or IVRANTWO

Examples: CPANEL Procedure

Example 9.1: Airline Cost Data: Fixed Effects

The Christenson Associates airline data are a frequently cited data set (Greene 2000). The data measure the costs, prices of inputs, and utilization rates for six airlines from 1970 to 1984. This example analyzes the log transformations of cost (variable C), quantity (variable Q), and price (variable PF), and the untransformed load factor (variable LF). You speculate the following model,

$$\log(C_{it}) = \alpha + \beta_1 \log(Q_{it}) + \beta_2 \log(PF_{it}) + \beta_3 LF_{it} + v_i + \epsilon_{it}$$

where the v_i are airline effects. The actual model in the original, untransformed variables is highly nonlinear:

$$C_{it} = \exp(\alpha + \beta_3 LF_{it} + v_i + \epsilon_{it}) Q_{it}^{\beta_1} PF_{it}^{\beta_2}$$

The following statements create the data and perform the necessary log transformations. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.airline;
  input Obs Airline T C Q PF LF;
  Year = T + 1969;
  lC   = log(C);
  lQ   = log(Q);
  lPF  = log(PF);
  label lC   = "Log Transformation of Costs";
  label lQ   = "Log Transformation of Quantity";
  label lPF  = "Log Transformation of Price of Fuel";
  label LF   = "Load Factor (utilization index)";
datalines;
1   1   1   1140640   0.95276   106650   0.53449
2   1   2   1215690   0.98676   110307   0.53233
3   1   3   1309570   1.09198   110574   0.54774
4   1   4   1511530   1.17578   121974   0.54085
5   1   5   1676730   1.16017   196606   0.59117

... more lines ...
```

The following statements fit a one-way fixed-effects model:

```
proc cpanel data = mycas.airline;
  id Airline Year;
  model lC = lQ lPF LF / fixone printfixed;
run;
```

Output 9.1.1 provides a model and data description. There are six cross sections and 15 time points.

Output 9.1.1 Airline Cost Data, Model Description

The CPANEL Procedure
One-Way Fixed Effects (FixOne)
Dependent Variable: IC (Log Transformation of Costs)

Model Description	
Estimation Method	FixOne
Variance Estimation	Model Based
Data Set	AIRLINE
Number of Observations	90
Number of Cross Sections	6
Time Series Length	15

The R-square and degrees of freedom can be seen in [Output 9.1.2](#). The R-square statistic nearly 1, indicating a reasonable fit. The error degrees of freedom is derived from 90 observations minus 5 cross sections, minus 4 regressors.

Output 9.1.2 Airline Cost Data, Fit Statistics

Fit Statistics			
SSE	0.2926	DFE	81
MSE	0.0036	Root MSE	0.0601
R-Square	0.9974		

The F test for fixed effects is shown in [Output 9.1.3](#). You easily reject the null hypothesis of poolability. There are significant effects due to airlines, and it would be unreasonable to perform pooled OLS regression that ignores these effects.

Output 9.1.3 Airline Cost Data, Test for Fixed Effects

F Test for No Fixed Effects			
Num DF	Den DF	F Value	Pr > F
5	81	57.74	<.0001

The PRINTFIXED option in the MODEL statement provides estimates of the airline effects (which are not displayed by default). Looking at the parameters, only two airlines (3 and 4) are significantly different from average. Quantity and fuel price have positive effects on cost, but load factors negatively affect costs. Because cost, quantity, and fuel price are log-transformed, the coefficients for quantity and price are interpreted as elasticities of cost. The coefficient for (log) fuel price is 0.417, meaning that you would associate a 10% increase in fuel price with a 4.17% increase in costs.

Output 9.1.4 Airline Cost Data, Parameter Estimates

Parameter Estimates						
Variable	DF	Estimate	Standard		Pr > t	Label
			Error	t Value		
Intercept	1	9.713556	0.2296	42.30	<.0001	Intercept
IQ	1	0.919293	0.0299	30.76	<.0001	Log Transformation of Quantity
IPF	1	0.417492	0.0152	27.47	<.0001	Log Transformation of Price of Fuel
LF	1	-1.07044	0.2017	-5.31	<.0001	Load Factor (utilization index)
CS 1	1	-0.00759	0.0465	-0.16	0.8707	Cross-Sectional Effect: 1
CS 2	1	-0.04883	0.0390	-1.25	0.2144	Cross-Sectional Effect: 2
CS 3	1	-0.21651	0.0184	-11.77	<.0001	Cross-Sectional Effect: 3
CS 4	1	0.176971	0.0214	8.27	<.0001	Cross-Sectional Effect: 4
CS 5	1	0.016475	0.0378	0.44	0.6638	Cross-Sectional Effect: 5
CS 6	1	0.079484	0.0415	1.92	0.0588	Cross-Sectional Effect: 6

You suspect that there might be other factors at play, and so you augment your model to include time effects. The following statements fit a two-way model, a model with both airline and time effects:

```
proc cpanel data = mycas.airline;
  id Airline Year;
  model lC = lQ lPF LF / fixtwo printfixed;
run;
```

The F test and parameter estimates for the two-way model are provided in [Output 9.1.5](#).

Output 9.1.5 Airline Cost Data, Two-Way Fixed Effects

The CPANEL Procedure
Two-Way Fixed Effects (FixTwo)
Dependent Variable: IC (Log Transformation of Costs)

F Test for No Fixed Effects				
Num DF	Den DF	F Value	Pr > F	
19	67	23.10	<.0001	

Output 9.1.5 *continued*

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	12.66527	2.0810	6.09	<.0001	Intercept
IQ	1	0.817264	0.0318	25.66	<.0001	Log Transformation of Quantity
IPF	1	0.168732	0.1635	1.03	0.3057	Log Transformation of Price of Fuel
LF	1	-0.88267	0.2617	-3.37	0.0012	Load Factor (utilization index)
CS 1	1	0.128306	0.0460	2.79	0.0069	Cross-Sectional Effect: 1
CS 2	1	0.065481	0.0390	1.68	0.0975	Cross-Sectional Effect: 2
CS 3	1	-0.18948	0.0156	-12.14	<.0001	Cross-Sectional Effect: 3
CS 4	1	0.134259	0.0183	7.33	<.0001	Cross-Sectional Effect: 4
CS 5	1	-0.09264	0.0373	-2.48	0.0155	Cross-Sectional Effect: 5
CS 6	1	-0.04593	0.0416	-1.10	0.2736	Cross-Sectional Effect: 6
Time 1970	1	-0.37386	0.1919	-1.95	0.0555	Time Effect: 1970
Time 1971	1	-0.31916	0.1861	-1.72	0.0909	Time Effect: 1971
Time 1972	1	-0.27654	0.1833	-1.51	0.1362	Time Effect: 1972
Time 1973	1	-0.22292	0.1730	-1.29	0.2019	Time Effect: 1973
Time 1974	1	-0.15388	0.0864	-1.78	0.0796	Time Effect: 1974
Time 1975	1	-0.10805	0.0449	-2.41	0.0188	Time Effect: 1975
Time 1976	1	-0.07686	0.0319	-2.41	0.0188	Time Effect: 1976
Time 1977	1	-0.02072	0.0204	-1.01	0.3145	Time Effect: 1977
Time 1978	1	0.047197	0.0291	1.62	0.1093	Time Effect: 1978
Time 1979	1	0.091657	0.0811	1.13	0.2627	Time Effect: 1979
Time 1980	1	0.207197	0.1491	1.39	0.1693	Time Effect: 1980
Time 1981	1	0.28534	0.1756	1.62	0.1089	Time Effect: 1981
Time 1982	1	0.301255	0.1660	1.81	0.0741	Time Effect: 1982
Time 1983	1	0.300347	0.1536	1.96	0.0547	Time Effect: 1983
Time 1984	1	0.319001	0.1475	2.16	0.0341	Time Effect: 1984

Only a few of the time effects are significant, but there is an overall time trend of increasing costs. The time period of the data spans the OPEC oil embargoes and the dissolution of the Civil Aeronautics Board (CAB). These are two possible explanations for the rising costs.

A surprising result is that the fuel cost is not significant in the two-way model. If the time effects are proxies for the effect of the oil embargoes, then the effect of fuel price might be subsumed by the time effects. If the time dummy variables are proxies for the dissolution of the CAB, then the effect of load factors is not precisely estimated.

Example 9.2: Analyzing Demand for Liquid Assets: Random Effects

Feige (1964) provides data on the demand for liquid assets. The data are for six states and the District of Columbia (CA, DC, FL, IL, NY, TX, and WA) and were collected each year from 1949 to 1959. All variables are log-transformed.

The following statements create the `Assets` data table. These statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.Assets;
  length state $ 2;
  input state $ year d t s y rd rt rs;
  label d = 'Per Capita Demand Deposits'
        t = 'Per Capita Time Deposits'
        s = 'Per Capita S & L Association Shares'
        y = 'Permanent Per Capita Personal Income'
        rd = 'Service Charge on Demand Deposits'
        rt = 'Interest on Time Deposits'
        rs = 'Interest on S & L Association Shares';
datalines;
CA 1949 6.2785 6.1924 4.4998 7.2056 -1.0700 0.1080 1.0664
CA 1950 6.4019 6.2106 4.6821 7.2889 -1.0106 0.1501 1.0767
CA 1951 6.5058 6.2729 4.8598 7.3827 -1.0024 0.4008 1.1291
CA 1952 6.4785 6.2729 5.0039 7.4000 -0.9970 0.4492 1.1227
CA 1953 6.4118 6.2538 5.1761 7.4200 -0.8916 0.4662 1.2110
CA 1954 6.4520 6.2971 5.3613 7.4478 -0.6951 0.4756 1.1924

... more lines ...
```

The data contain per capita consumptions for three liquid assets: demand deposits such as checking, time deposits, and savings and loan (S&L) shares. You posit a linear model for per capita demand deposits, with random effects for states.

The following statements fit a one-way random-effects model:

```
proc cpanel data = mycas.Assets;
  id state year;
  model d = y rd rt rs / ranone;
run;
```

The regression results are provided in [Output 9.2.1](#).

The “Variance Component Estimates” table provides the estimated variances and standard deviations for the cross-sectional (state) effects in addition to the overall errors. A majority of the overall error variance can be attributed to differences between states, not differences within states.

The “Hausman Test for Random Effects” table shows the result of a Hausman specification test. The null hypothesis is that state effects can be treated as random (random-effects model) and that they do not need to be estimated directly (fixed-effects model). The test results favor the random-effects specification that is used to generate this output.

Output 9.2.1 Demand for Demand Deposits, One-Way Random-Effects Model

The CPANEL Procedure
One-Way Random Effects (RanOne)
Swamy and Arora Variance Components
Dependent Variable: d (Per Capita Demand Deposits)

Model Description					
Estimation Method	RanOne				
Variance Estimation	Model Based				
Data Set	ASSETS				
Number of Observations	77				
Number of Cross Sections	7				
Time Series Length	11				

Fit Statistics			
SSE	0.0939	DFE	72
MSE	0.0013	Root MSE	0.0361
R-Square	0.7651		

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046394	0.215393
Error	0.00134	0.036608

Hausman Test For Random Effects				
Coefficients	DF	m	Value	Pr > m
	4	4	1.52	0.8235

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	-1.64325	0.6841	-2.40	0.0189
y	1	1.136776	0.0998	11.39	<.0001
rd	1	-0.26428	0.0517	-5.11	<.0001
rt	1	0.032076	0.0291	1.10	0.2746
rs	1	-0.41213	0.1192	-3.46	0.0009

The parameter estimate for the variable Y is greater than 1, indicating that demand is elastic to income— income has a more than proportional positive association with the demand for demand deposits. The coefficient on the variable RD indicates that demand deposits increase significantly as the service charge is reduced.

The variables RT and RS represent positive aspects of competing products, and you would expect these variables to affect demand negatively. The coefficient for RS meets that expectation, but the coefficient for RT is not significant.

The previous analysis used the default Swamy-Arora method to estimate the variance components. The CPANEL procedure supports four other methods, and you might be interested in how the different methods affect the analysis.

The following statements fit the model by using all five methods and include a COMPARE statement to compare the results:

```
proc cpanel data = mycas.Assets;
  id state year;
  sa: model d = y rd rt rs / ranone vcomp = sa;
  wh: model d = y rd rt rs / ranone vcomp = wh;
  wk: model d = y rd rt rs / ranone vcomp = wk;
  fb: model d = y rd rt rs / ranone vcomp = fb;
  nl: model d = y rd rt rs / ranone vcomp = nl;
  compare / mstat(varcs varerr);
run;
```

The tables produced by the COMPARE statement are provided in [Output 9.2.2](#).

Output 9.2.2 Comparison of Variance-Component Methods, Assets Data

The CPANEL Procedure						
Model Comparison						
Model: NL						
Dependent Variable: d (Per Capita Demand Deposits)						
Comparison of Model Statistics						
Statistic		SA	WH	WK	FB	NL
		RanOne	RanOne	RanOne	RanOne	RanOne
Var due to Cross Sections		0.0464	0.0315	0.0315	0.0291	0.0327
Var due to Error		0.001340	0.000107	0.001340	0.001340	0.001149
Comparison of Model Parameter Estimates						
Variable		SA	WH	WK	FB	NL
		RanOne	RanOne	RanOne	RanOne	RanOne
Intercept	Estimate	-1.643246	-1.472425	-1.723092	-1.742581	-1.680406
	Std Err	0.684144	0.719067	0.681184	0.680541	0.682676
y	Estimate	1.136776	1.117252	1.145844	1.148051	1.141001
	Std Err	0.099816	0.099799	0.099776	0.099761	0.099802
rd	Estimate	-0.264276	-0.245861	-0.272995	-0.275135	-0.268325
	Std Err	0.051729	0.052260	0.051445	0.051372	0.051600
rt	Estimate	0.032076	0.029227	0.033397	0.033718	0.032692
	Std Err	0.029138	0.028570	0.029416	0.029485	0.029266
rs	Estimate	-0.412126	-0.414540	-0.410731	-0.410361	-0.411500
	Std Err	0.119181	0.117486	0.119968	0.120160	0.119548

You conclude that how you estimate variance components has little bearing on the regression results.

Example 9.3: Panel Study of Income Dynamics (PSID): Hausman-Taylor Models

Cornwell and Rupert (1988) analyze data from the Panel Study of Income Dynamics (PSID), an income study of 595 individuals over the seven-year period, 1976–1982 inclusive. Of particular interest is the effect of additional schooling on wages. The analysis here replicates that of Baltagi (2013, sec. 7.5), where it is surmised that covariate correlation with individual effects makes a standard random-effects model inadequate.

The following statements create the PSID data. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.psid;
  input id t lwage wks south smsa ms exp exp2 occ ind union fem blk ed;
  label id      = 'Person ID'
        t      = 'Time'
        lwage  = 'Log of wages'
        wks    = 'Weeks worked'
        south  = '1 if resides in the South'
        smsa   = '1 if resides in SMSA'
        ms     = '1 if married'
        exp    = 'Years full-time experience'
        exp2   = 'exp squared'
        occ    = '1 if blue-collar occupation'
        ind    = '1 if manufacturing'
        union  = '1 if union contract'
        fem    = '1 if female'
        blk    = '1 if black'
        ed     = 'Years of education';
datalines;
1   1  5.5606799126  32  1  0  1  3   9   0  0  0  0  0  0  9
1   2  5.7203102112  43  1  0  1  4  16   0  0  0  0  0  0  9
1   3  5.9964499474  40  1  0  1  5  25   0  0  0  0  0  0  9
1   4  5.9964499474  39  1  0  1  6  36   0  0  0  0  0  0  9
1   5  6.0614600182  42  1  0  1  7  49   0  1  0  0  0  0  9
1   6  6.1737899780  35  1  0  1  8  64   0  1  0  0  0  0  9
1   7  6.2441701889  32  1  0  1  9  81   0  1  0  0  0  0  9
2   1  6.1633100510  34  0  0  1  30  900  1  0  0  0  0  0  11
2   2  6.2146100998  27  0  0  1  31  961  1  0  0  0  0  0  11
2   3  6.2634000778  33  0  0  1  32 1024  1  1  1  0  0  0  11
2   4  6.5439100266  30  0  0  1  33 1089  1  1  0  0  0  0  11
2   5  6.6970300674  30  0  0  1  34 1156  1  1  0  0  0  0  11
2   6  6.7912201881  37  0  0  1  35 1225  1  1  0  0  0  0  11
2   7  6.8156399727  30  0  0  1  36 1296  1  1  0  0  0  0  11

... more lines ...
```

You begin by fitting a one-way random-effects model:

```
proc cpanel data = mycas.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
              ind union fem blk ed / ranone;
run;
```

The output is shown in [Output 9.3.1](#). The coefficient on the variable ED (which represents years of education) estimates that an additional year of schooling is associated with about a 10% increase in wages. However, the results of the Hausman test for random effects show a serious violation of the random-effects assumptions, namely that the regressors are independent of the individual effects.

Output 9.3.1 One-Way Random Effects Estimation

The CPANEL Procedure One-Way Random Effects (RanOne) Swamy and Arora Variance Components Dependent Variable: lwage (Log of wages)

Model Description						
Estimation Method	RanOne					
Variance Estimation	Model Based					
Data Set	PSID					
Number of Observations	4165					
Number of Cross Sections	595					
Time Series Length	7					

Variance Component Estimates			
Source	Variance	Standard Deviation	
Cross Sections	0.068972	0.262626	
Error	0.023102	0.151994	

Hausman Test For Random Effects				
Coefficients	DF	m Value	Pr > m	
9	9	5074.87	<.0001	

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	4.26382	0.0977	43.64	<.0001	Intercept
wks	1	0.001035	0.000773	1.34	0.1810	Weeks worked
south	1	-0.01662	0.0265	-0.63	0.5309	1 if resides in the South
smsa	1	-0.01381	0.0200	-0.69	0.4897	1 if resides in SMSA
ms	1	-0.07463	0.0230	-3.24	0.0012	1 if married
exp	1	0.082051	0.00285	28.81	<.0001	Years full-time experience
exp2	1	-0.00081	0.000063	-12.87	<.0001	exp squared
occ	1	-0.05007	0.0166	-3.01	0.0026	1 if blue-collar occupation
ind	1	0.003745	0.0173	0.22	0.8283	1 if manufacturing
union	1	0.063226	0.0171	3.70	0.0002	1 if union contract
fem	1	-0.33923	0.0513	-6.61	<.0001	1 if female
blk	1	-0.21027	0.0580	-3.63	0.0003	1 if black
ed	1	0.099653	0.00575	17.34	<.0001	Years of education

An alternative could be a fixed-effects (FIXONE) model, but that model would not permit estimation of the coefficient for ED, which does not vary within individuals. A compromise is the Hausman-Taylor model, for which you stipulate a set of covariates that are correlated with the individual effects (but uncorrelated with

the observation-level errors). You specify the correlated variables in the CORRELATED statement:

```
proc cpanel data = mycas.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
              ind union fem blk ed / htaylor;
  correlated wks ms exp exp2 union ed;
run;
```

The results are shown in [Output 9.3.2](#). The table of parameter estimates has an added column, Type, which identifies the regressors that are assumed to be correlated with individual effects (C) and the regressors that do not vary within cross sections (TI). It was stated previously that the Hausman-Taylor model is a compromise between fixed-effects and random-effects models, and you can think of the compromise this way: You want to fit a random-effects model, but the correlated (C) variables make that model invalid. So you fall back to the consistent fixed-effects model, but then the time-invariant (TI) variables are the problem because they would be dropped from that model. The solution is to use the Hausman-Taylor estimator.

The estimation results show that an additional year of schooling is now associated with a 13.8% increase in wages. Also presented is a Hausman test that compares this model to the fixed-effects model. As was the case previously when you fit the random-effects model, you can think of the Hausman test as a referendum on the assumptions you are making. For this estimation, it seems that your choice of variables to treat as correlated is adequate.

Output 9.3.2 Hausman-Taylor Estimation

The CPANEL Procedure Hausman and Taylor Model for Correlated Individual Effects Dependent Variable: lwage (Log of wages)

Variance Component Estimates			
Source	Variance	Standard Deviation	
Cross Sections	0.886993	0.941803	
Error	0.023044	0.151803	

Hausman Test vs. Fixed Effects				
Coefficients	DF	m Value	Pr > m	
9	3	5.26	0.1539	

Output 9.3.2 *continued*

Parameter Estimates							
Variable	Type	DF	Estimate	Standard	t Value	Pr > t	Label
				Error			
Intercept	TI	1	2.912726	0.2837	10.27	<.0001	Intercept
wks	C	1	0.000837	0.000600	1.40	0.1627	Weeks worked
south		1	0.00744	0.0320	0.23	0.8159	1 if resides in the South
smsa		1	-0.04183	0.0190	-2.21	0.0274	1 if resides in SMSA
ms	C	1	-0.02985	0.0190	-1.57	0.1159	1 if married
exp	C	1	0.113133	0.00247	45.79	<.0001	Years full-time experience
exp2	C	1	-0.00042	0.000055	-7.67	<.0001	exp squared
occ		1	-0.0207	0.0138	-1.50	0.1331	1 if blue-collar occupation
ind		1	0.013604	0.0152	0.89	0.3720	1 if manufacturing
union	C	1	0.032771	0.0149	2.20	0.0280	1 if union contract
fem	TI	1	-0.13092	0.1267	-1.03	0.3014	1 if female
blk	TI	1	-0.28575	0.1557	-1.84	0.0665	1 if black
ed	C TI	1	0.137944	0.0212	6.49	<.0001	Years of education

C: correlated with the individual effects

TI: constant (time-invariant) within cross sections

At its core, the Hausman-Taylor estimator is an instrumental variables regression, where the instruments are derived from regressors that are assumed to be uncorrelated with the individual effects. Technically, it is the cross-sectional means of these variables that need to be uncorrelated, not the variables themselves.

The Amemiya-MaCurdy model is a close relative of the Hausman-Taylor model. The only difference between the two is that the Amemiya-MaCurdy model makes the added assumption that the regressors (and not just their means) are uncorrelated with the individual effects. By making that assumption, the Amemiya-MaCurdy model can take advantage of a more efficient set of instrumental variables.

The following statements fit the Amemiya-MaCurdy model:

```
proc cpanel data = mycas.psid;
  id id t;
  model lwage = wks south smsa ms exp exp2 occ
               ind union fem blk ed / amacurdy;
  correlated wks ms exp exp2 union ed;
run;
```

The results are shown in [Output 9.3.3](#). Little is changed from the Hausman-Taylor model. The Hausman test compares the Amemiya-MaCurdy model to the Hausman-Taylor model (not the fixed-effects model as previously) and shows that the one additional assumption is acceptable. You even gained a bit of efficiency in the process; compare the standard deviations of the coefficient on the variable ED from both models.

Output 9.3.3 Amemiya-MaCurdy Estimation

The CPANEL Procedure
Amemiya and MaCurdy Model for Correlated Individual Effects
Dependent Variable: lwage (Log of wages)

Variance Component Estimates							
Source		Variance	Standard				
			Deviation				
Cross Sections		0.886993	0.941803				
Error		0.023044	0.151803				

Hausman Test vs. Hausman-Taylor				
Coefficients	DF	m	Value	Pr > m
13	13		14.67	0.3287

Parameter Estimates							
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	TI	1	2.927338	0.2751	10.64	<.0001	Intercept
wks	C	1	0.000838	0.000599	1.40	0.1622	Weeks worked
south		1	0.007282	0.0319	0.23	0.8197	1 if resides in the South
smsa		1	-0.04195	0.0189	-2.21	0.0269	1 if resides in SMSA
ms	C	1	-0.03009	0.0190	-1.59	0.1127	1 if married
exp	C	1	0.11297	0.00247	45.76	<.0001	Years full-time experience
exp2	C	1	-0.00042	0.000055	-7.72	<.0001	exp squared
occ		1	-0.02085	0.0138	-1.51	0.1299	1 if blue-collar occupation
ind		1	0.013629	0.0152	0.89	0.3709	1 if manufacturing
union	C	1	0.032475	0.0149	2.18	0.0293	1 if union contract
fem	TI	1	-0.13201	0.1266	-1.04	0.2972	1 if female
blk	TI	1	-0.2859	0.1555	-1.84	0.0660	1 if black
ed	C TI	1	0.137205	0.0206	6.67	<.0001	Years of education

C: correlated with the individual effects							
TI: constant (time-invariant) within cross sections							

Finally, you should realize that the Hausman-Taylor and Amemiya-MaCurdy estimators are not cure-alls for correlated individual effects. Estimation tacitly relies on the uncorrelated regressors being sufficient to predict the correlated regressors. Otherwise, you run into the problem of weak instruments. If you have weak instruments, you will obtain biased estimates that have very large standard errors. However, that does not seem to be the case here.

Example 9.4: Crime Rates in North Carolina: Instrumental Variables Regression

Cornwell and Trumbull (1994) examined data on crime rates from 90 counties in North Carolina and collected data each year from 1981 to 1987. The following statements create the Crime data table. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.Crime;
  input county year west central urban d82 d83 d84 d85 d86 d87
        lcrmte lprbarr lprbconv lprbpris lavgsen lpolpc
        ldensity ltaxpc lwcon lwtuc lwtrd lwfir lwser lwmfg
        lwfed lwsta lwloc lmix lpctymle lpctmin;
  label county      = 'county identifier'
        year        = '81 to 87'
        west        = '1 if in western N.C.'
        central     = '1 if in central N.C.'
        urban       = '1 if in SMSA'

  ... more lines ...
```

The variables are too numerous to describe individually, but they are fully labeled in the regression output and are summarized as follows:

- The panel ID variables are County and Year.
- The dependent variable, LCrMRate, is the log of the crime rate (number of crimes divided by total population).
- The variables West, Central, and Urban provide geographical background for each county.
- The variables D82 through D87 are dummy variables for year effects.
- The variables with the LPrb prefix are log-probabilities of police or legal actions. For example, LPrbArr is the log-probability of an arrest given that a crime was committed.
- The variables with the LW prefix are log weekly wages for various occupations and industries.
- Other variables that begin with the letter L are log-transformed. For example, the variable LPolPC is the log of police per capita.

Cornwell and Trumbull (1994) fit a fixed-effects model for the log of crime rate, with the log-probability of arrest (LPrbArr), and the log of police per capita (LPolPC) among the regressors. They noted that these two regressors were endogenous because of the following causality loops:

- Although police presence might decrease the crime rate because of deterrence (or increase the crime rate because of better reporting), an increase in crime might cause a desire for more police presence.
- Although the crime rate might fall if more crimes resulted in arrest, a rising crime rate could cause the probability of arrest to decrease because police resources would be spread thin.

To deal with the endogeneity, Cornwell and Trumbull (1994) fit a two-stage least squares (2SLS) model with county-level fixed effects, with two external instruments: LTaxPC (log tax revenue per capita) and LMix (log ratio of crimes that involve face-to-face contact). Their rationale for using these instruments is that counties with higher tax revenues have more police presence and that face-to-face crimes have a higher probability of arrest because victims could identify their offenders.

Baltagi (2006) proposed a similar 2SLS model, but with random effects instead of fixed effects for counties. The following statements recreate that estimation:

```
proc cpanel data = mycas.crime;
  id county year;
  model lcrmte = lprbarr lpolpc lprbconv lprbpris lavgsen ldensity
               lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta
               lwloc lpctymle lpctmin west central urban
               d82 d83 d84 d85 d86 d87 / ivranone;
  endogenous lprbarr lpolpc;
  instruments ltaxpc lmix;
run;
```

The results are shown in [Output 9.4.1](#). The output includes a “Structural Summary” table, which lists the endogenous variables and the instrumental variables. The list of instrumental variables includes both the specified external instruments (variables LTaxPC and LMix) and the exogenous regression variables.

Output 9.4.1 Crime Rates in North Carolina, IV Random Effects

The CPANEL Procedure EC2SLS One-Way Random Effects (IVRanOne) Dependent Variable: lcrmte (log crimes committed per person)

Model Description			
Estimation Method	IVRanOne		
Variance Estimation	Model Based		
Data Set	CRIME		
Number of Observations	630		
Number of Cross Sections	90		
Time Series Length	7		

Fit Statistics			
SSE	12.1250	DFE	603
MSE	0.0201	Root MSE	0.1418

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046036	0.21456
Error	0.022272	0.149239

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
22 22	22	19.50	0.6140

Output 9.4.1 continued

Structural Summary								
Endogenous Variables	lprbarr lpolpc							
Instruments	lprbconv lprbpris lavgsen ldensity lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta lwloc lpctymle lpctmin west central urban d82 d83 d84 d85 d86 d87 ltaxpc lmix							
Parameter Estimates								
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label	
Intercept		1	-0.9538	1.2840	-0.74	0.4579	Intercept	
lprbarr	EN	1	-0.41293	0.0974	-4.24	<.0001	log probability of arrest	
lpolpc	EN	1	0.434749	0.0897	4.85	<.0001	log police per capita	
lprbconv		1	-0.32289	0.0536	-6.03	<.0001	log probability of conviction	
lprbpris		1	-0.18632	0.0419	-4.44	<.0001	log probability of prison sentence	
lavgsen		1	-0.01018	0.0270	-0.38	0.7066	log avg. sentence in days	
ldensity		1	0.429028	0.0548	7.82	<.0001	log people per sq. mile	
lwcon		1	-0.00748	0.0396	-0.19	0.8503	log weekly wage, construction	
lwtuc		1	0.045445	0.0198	2.30	0.0220	log wkly wge, trns, util, commun	
lwtrd		1	-0.00814	0.0414	-0.20	0.8441	log wkly wge, whlesle, retail trade	
lwfir		1	-0.00364	0.0289	-0.13	0.8999	log wkly wge, fin, ins, real est	
lwser		1	0.00561	0.0201	0.28	0.7805	log wkly wge, service industry	
lwmfg		1	-0.20414	0.0804	-2.54	0.0114	log wkly wge, manufacturing	
lwfed		1	-0.16351	0.1594	-1.03	0.3056	log wkly wge, fed employees	
lwsta		1	-0.05405	0.1057	-0.51	0.6092	log wkly wge, state employees	
lwloc		1	0.163052	0.1196	1.36	0.1734	log wkly wge, local gov emps	
lpctymle		1	-0.10811	0.1397	-0.77	0.4393	log percent young male	
lpctmin		1	0.189037	0.0415	4.56	<.0001	log perc. minority, 1980	
west		1	-0.22684	0.0996	-2.28	0.0231	1 if in western N.C.	
central		1	-0.19404	0.0598	-3.24	0.0012	1 if in central N.C.	
urban		1	-0.22515	0.1156	-1.95	0.0520	1 if in SMSA	
d82		1	0.010745	0.0258	0.42	0.6772	1 if year == 82	
d83		1	-0.08379	0.0307	-2.73	0.0065	1 if year == 83	
d84		1	-0.1035	0.0371	-2.79	0.0054	1 if year == 84	
d85		1	-0.0957	0.0495	-1.94	0.0534	1 if year == 85	
d86		1	-0.0689	0.0596	-1.16	0.2481	1 if year == 86	
d87		1	-0.03141	0.0705	-0.45	0.6562	1 if year == 87	

EN: endogenous variable

EN: endogenous variable

Cornwell and Trumbull (1994) favored a fixed-effects approach because a Hausman specification test rejected the null hypothesis of random effects. Baltagi (2006) argued that the Hausman test used by Cornwell and Trumbull (1994) was inaccurate because it was based on standard methodology that did not account for endogeneity in the regressors.

Baltagi (2006) proposed a modified 2SLS Hausman test, and the results of that test are provided in the “Hausman Test For Random Effects” table in the output. The results favor the random-effects specification.

A higher probability of arrest is shown to diminish crime, and police presence increases the crime rate because more crimes are reported.

The following statements fit the model by two-step GMM with robust standard errors:

```
proc cpanel data = mycas.crime;
  id county year;
  model lcrmte = lprbarr lpolpc lprbconv lprbpris lavgsen ldensity
              lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta
              lwloc lpctymle lpctmin west central urban
              d82 d83 d84 d85 d86 d87 / ivranone gmm = twostep
              robust;
  endogenous lprbarr lpolpc;
  instruments ltaxpc lmix;
run;
```

The results are shown in [Output 9.4.2](#). The coefficients do not change much, but the standard errors are now more reliable. You can also obtain bias-corrected standard errors by using the `BIASCORRECTED` option according to the method of Windmeijer (2005).

Output 9.4.2 Crime Rates in North Carolina, IV Random Effects by Two-Step GMM

The CPANEL Procedure
EC2SLS One-Way Random Effects (IVRanOne)
Dependent Variable: lcrmte (log crimes committed per person)

Model Description	
Estimation Method	IVRanOne
Estimation Technique	Two-Step GMM
Variance Estimation	Robust
Data Set	CRIME
Number of Observations	630
Number of Cross Sections	90
Time Series Length	7

Fit Statistics			
SSE	12.3814	DFE	603
MSE	0.0205	Root MSE	0.1433

Variance Component Estimates		
Source	Variance	Standard Deviation
Cross Sections	0.046036	0.21456
Error	0.022272	0.149239

Hausman Test For Random Effects			
Coefficients	DF	m Value	Pr > m
22	22	15.78	0.8269

Structural Summary	
Endogenous Variables	lprbarr lpolpc
Instruments	lprbconv lprbpris lavgsen ldensity lwcon lwtuc lwtrd lwfir lwser lwmfg lwfed lwsta lwloc lpctymle lpctmin west central urban d82 d83 d84 d85 d86 d87 ltaxpc lmix

Output 9.4.2 continued

Parameter Estimates							
Variable	Type	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept		1	-1.66797	1.1204	-1.49	0.1371	Intercept
lprbarr	EN	1	-0.37858	0.1137	-3.33	0.0009	log probability of arrest
lpolpc	EN	1	0.426573	0.0858	4.97	<.0001	log police per capita
lprbconv		1	-0.31245	0.0587	-5.32	<.0001	log probability of conviction
lprbpris		1	-0.14817	0.0458	-3.23	0.0013	log probability of prison sentence
lavgsen		1	0.030359	0.0309	0.98	0.3255	log avg. sentence in days
ldensity		1	0.340629	0.0585	5.83	<.0001	log people per sq. mile
lwcon		1	-0.02737	0.0246	-1.11	0.2658	log weekly wage, construction
lwtuc		1	0.029904	0.0195	1.54	0.1248	log wkly wge, trns, util, commun
lwtrd		1	0.003937	0.0422	0.09	0.9258	log wkly wge, whlesle, retail trade
lwfir		1	-0.00248	0.0131	-0.19	0.8498	log wkly wge, fin, ins, real est
lwser		1	-0.00434	0.0130	-0.33	0.7379	log wkly wge, service industry
lwmfg		1	-0.16174	0.0835	-1.94	0.0532	log wkly wge, manufacturing
lwfed		1	-0.00709	0.1499	-0.05	0.9623	log wkly wge, fed employees
lwsta		1	-0.02507	0.0798	-0.31	0.7535	log wkly wge, state employees
lwloc		1	0.090092	0.1089	0.83	0.4086	log wkly wge, local gov emps
lpctymle		1	-0.07707	0.1143	-0.67	0.5002	log percent young male
lpctmin		1	0.181526	0.0478	3.79	0.0002	log perc. minority, 1980
west		1	-0.2151	0.1031	-2.09	0.0374	1 if in western N.C.
central		1	-0.16708	0.0537	-3.11	0.0020	1 if in central N.C.
urban		1	-0.12948	0.0816	-1.59	0.1129	1 if in SMSA
d82		1	0.01654	0.0242	0.68	0.4939	1 if year == 82
d83		1	-0.07028	0.0313	-2.25	0.0251	1 if year == 83
d84		1	-0.09766	0.0339	-2.88	0.0041	1 if year == 84
d85		1	-0.08787	0.0462	-1.90	0.0578	1 if year == 85
d86		1	-0.0586	0.0508	-1.15	0.2492	1 if year == 86
d87		1	-0.02047	0.0580	-0.35	0.7244	1 if year == 87

 EN: endogenous variable

Example 9.5: Cigarette Sales Data: Dynamic Panel Estimation

Consider a dynamic panel demand model for cigarette sales that illustrates the methods described in the section “[Dynamic Panel Estimation \(DYNDIFF and DYNSSYS Options\)](#)” on page 389. The data are from a panel of 46 American states over the period 1963–1992. The dependent variable is the logarithm of per capita cigarette sales (variable LSales). Other factors that were measured include the log of price (LPrice), the log of disposable income (LDisp), and the log of minimum price in adjoining states (LMin). For a full description of the data, see Baltagi (2013, sec. 8.9).

The following statements create the Cigar data set. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```

data mycas.Cigar;
  input State Year Price Pop Pop_16 Cpi Disp Sales Min;
  LSales = log(Sales);
  LPrice = log(Price);
  LDisp  = log(Disp);
  LMin   = log(Min);
  label
    State   = 'State abbreviation'
    Year    = 'Year'
    LSales  = 'Log cigarette sales in packs per capita'
    LPrice  = 'Log price per pack of cigarettes'
    LDisp   = 'Log per capita disposable income'
    LMin    = 'Log minimum price in adjoining states per pack of cigarettes';
datalines;
1 63 28.6 3383 2236.5 30.6 1558.3045298 93.9 26.1
1 64 29.8 3431 2276.7 31.0 1684.0732025 95.4 27.5
1 65 29.8 3486 2327.5 31.5 1809.8418752 98.5 28.9
1 66 31.5 3524 2369.7 32.4 1915.1603572 96.4 29.5
1 67 31.6 3533 2393.7 33.4 2023.5463678 95.5 29.6
1 68 35.6 3522 2405.2 34.8 2202.4855362 88.4 32
1 69 36.6 3531 2411.9 36.7 2377.3346665 90.1 32.8
1 70 39.6 3444 2394.6 38.8 2591.0391591 89.8 34.3
1 71 42.7 3481 2443.5 40.5 2785.3159706 95.4 35.8
... more lines ...

```

You propose a panel model for cigarette sales that contains fixed effects for states. Because you believe that the data are insufficient to explain all possible shocks in yearly sales, you include lagged sales in the model as a regressor. By construction, lagged sales are an endogenous regressor, and you thus specify dynamic panel estimation by using the DYNDIFF option. The following statements fit the model:

```

proc cpanel data = mycas.Cigar;
  id State Year;
  model LSales = LPrice LDisp LMin / dyndiff;
run;

```

The results are shown in [Output 9.5.1](#). Note that it was not necessary to explicitly include lagged sales on the right-hand side of the model; PROC CPANEL generates it for you. The coefficient on lagged sales is 0.732, indicating a high degree of autocorrelation in the dependent variable. When cigarette sales are unusually high or low because of unforeseen circumstances, the effects tend to linger for several years. The results also show that demand is highly elastic to price.

Output 9.5.1 Dynamic Panel Estimation for Cigarette Sales

The CPANEL Procedure
Dynamic-Panel Estimation using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description					
Estimation Method	DynDiff				
Estimation Technique	One-Step GMM				
Variance Estimation	Model Based				
Data Set	CIGAR				
Number of Observations	1380				
Number of Cross Sections	46				
Time Series Length	30				
GMM Bandwidth	30				

Fit Statistics			
SSE	3.1373	DFE	1283
MSE	0.0024	Root MSE	0.0494

Instruments		
Type	Variables	Number of Instruments
Difference Equations, Endogenous	LSales	406
Difference Equations, Standard	LPrice LDisp LMin	3
Level Equations, Standard	Intercept	1
Total		410

Sargan Test		
DF	Chi-Square	Pr > ChiSq
405	712.45	<.0001

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	0.769092	0.0658	11.69	<.0001
LSales (Lag 1)	1	0.732212	0.0178	41.07	<.0001
LPrice	1	-0.26328	0.0255	-10.31	<.0001
LDisp	1	0.166116	0.0105	15.88	<.0001
LMin	1	0.032726	0.0233	1.40	0.1604

AR(m) Tests		
Lag	Z	Pr > Z
1	-15.44	<.0001
2	2.47	0.0134

Included in [Output 9.5.1](#) are two diagnostic measures. The first, a Sargan test, is a test of the validity of the moment conditions that are conferred by the GMM instruments that were used. The p -value indicates that the moment conditions are not valid and that you should probably look for a set of instruments other than the default set that PROC CPANEL provides.

The second diagnostic test is the $AR(m)$ test for autocorrelation in the residuals. In well-fitting dynamic panel models, you expect to see some autocorrelation of lag 1, but any autocorrelation at higher lags indicates a poor fit. The autocorrelation at lag 2 is significant, leading you to seek a better-fitting alternative.

One possible explanation for the poor fit is that, by default, PROC CPANEL uses the one-step generalized method of moments (GMM). One-step GMM is known for being too reliant on the assumption that the residuals from the difference equations are not serially correlated. An alternative is two-step GMM, which instead uses a data-driven variance matrix for the differenced residuals.

The following statements fit the model by two-step GMM:

```
proc cpanel data = mycas.Cigar;
  id State Year;
  endogenous LSales;
  instruments LPrice LDisp LMin;
  model LSales = LPrice LDisp LMin / dyndiff gmm = twostep biascorrected;
run;
```

The code includes ENDOGENOUS and INSTRUMENTS statements that, for demonstration purposes, reproduce the default instrument set. That set includes the following:

- GMM-style instruments based on the endogenous dependent variable, LSales
- standard instruments for the exogenous regressors LPrice, LDisp, and LMin
- a column of ones in the instrument matrix for the level equations that corresponds to the model intercept

The code also includes the BIASCORRECTED option, which produces bias-corrected standard errors according to the method of Windmeijer (2005).

The results are shown in [Output 9.5.2](#). The coefficients do not change much, but the standard errors are now more reliable. The model diagnostic tests indicate a better fit, although you should use caution when interpreting Sargan test results. Sargan tests lack power when the number of instruments is large, and their distributional properties come into question under conditions that favor either robust or bias-corrected standard errors.

Output 9.5.2 Dynamic Panel Estimation by Two-Step GMM

The CPANEL Procedure
Dynamic-Panel Estimation using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description	
Estimation Method	DynDiff
Estimation Technique	Two-Step GMM
Variance Estimation	Windmeijer Bias Corrected
Data Set	CIGAR
Number of Observations	1380
Number of Cross Sections	46
Time Series Length	30
GMM Bandwidth	30

Output 9.5.2 *continued*

Fit Statistics						
SSE	3.1277	DFE	1283			
MSE	0.0024	Root MSE	0.0494			

Instruments			
Type	Variables	Number of Instruments	
Difference Equations, Endogenous	LSales	406	
Difference Equations, Standard	LPrice LDisp LMin	3	
Level Equations, Standard	Intercept	1	
Total		410	

Sargan Test		
DF	Chi-Square	Pr > ChiSq
41	45.20	0.3008

Parameter Estimates						
Variable	DF	Estimate	Standard Error	t Value	Pr > t	Label
Intercept	1	0.776914	0.1458	5.33	<.0001	Intercept
LSales (Lag 1)	1	0.728347	0.0492	14.79	<.0001	Lag 1: Log cigarette sales in packs per capita
LPrice	1	-0.25707	0.0421	-6.11	<.0001	Log price per pack of cigarettes
LDisp	1	0.167345	0.0248	6.75	<.0001	Log per capita disposable income
LMin	1	0.025064	0.0432	0.58	0.5616	Log minimum price in adjoining states per pack of cigarettes

AR(m) Tests		
Lag	Z	Pr > Z
1	-5.00	<.0001
2	1.90	0.0575

The previous estimation treats regressors such as LPrice as exogenous. If you believe that price is endogenous, you can create GMM-style instruments for LPrice to replace the default standard instruments.

The following statements fit the model by using GMM-style instruments for LPrice:

```
proc cpanel data = mycas.Cigar;
  id State Year;
  endogenous LSales LPrice;
  instruments LDisp LMin;
  model LSales = LPrice LDisp LMin / dyndiff gmm = twostep biascorrected;
run;
```

The results are shown in [Output 9.5.3](#). Treating LPrice as endogenous nearly doubles the number of instruments. Although this is not the case here, when the number of instruments is so large that it makes estimation infeasible, you can limit the number of instruments by specifying the MAXBAND= option in the MODEL statement.

Output 9.5.3 Dynamic Panel Estimation, Custom Instrument Set

The CPANEL Procedure
Dynamic-Panel Estimation using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

Model Description	
Estimation Method	DynDiff
Estimation Technique	Two-Step GMM
Variance Estimation	Windmeijer Bias Corrected
Data Set	CIGAR
Number of Observations	1380
Number of Cross Sections	46
Time Series Length	30
GMM Bandwidth	30

Fit Statistics			
SSE	3.3942	DFE	1283
MSE	0.0026	Root MSE	0.0514

Instruments		
Type	Variables	Number of Instruments
Difference Equations, Endogenous	LSales LPrice	812
Difference Equations, Standard	LDisp LMin	2
Level Equations, Standard	Intercept	1
Total		815

Sargan Test		
DF	Chi-Square	Pr > ChiSq
41	44.98	0.3088

Parameter Estimates					
Variable	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	0.531106	0.1271	4.18	<.0001
LSales (Lag 1)	1	0.799028	0.0412	19.38	<.0001
LPrice	1	-0.22491	0.0346	-6.50	<.0001
LDisp	1	0.139713	0.0202	6.92	<.0001
LMin	1	0.029462	0.0383	0.77	0.4416

AR(m) Tests		
Lag	Z	Pr > Z
1	-5.02	<.0001
2	1.94	0.0521

References

- Amemiya, T. (1971). "The Estimation of the Variances in a Variance-Component Model." *International Economic Review* 12:1–13.
- Amemiya, T., and MaCurdy, T. E. (1986). "Instrumental-Variable Estimation of an Error-Components Model." *Econometrica* 54:869–880.
- Andrews, D. W. K. (1991). "Heteroscedasticity and Autocorrelation Consistent Covariance Matrix Estimation." *Econometrica* 59:817–858.
- Arellano, M., and Bond, S. (1991). "Some Tests of Specification for Panel Data: Monte Carlo Evidence and an Application to Employment Equations." *Review of Economic Studies* 58:277–297.
- Baltagi, B. H. (1981). "Simultaneous Equations with Error Components." *Journal of Econometrics* 17:189–200.
- Baltagi, B. H. (2006). "Estimating an Economic Model of Crime Using Panel Data from North Carolina." *Journal of Applied Econometrics* 21:543–547.
- Baltagi, B. H. (2013). *Econometric Analysis of Panel Data*. 5th ed. Chichester, UK: John Wiley & Sons.
- Baltagi, B. H., and Chang, Y. (1994). "Incomplete Panels: A Comparative Study of Alternative Estimators for the Unbalanced One-Way Error Component Regression Model." *Journal of Econometrics* 62:67–89.
- Baltagi, B. H., Song, S. H., and Jung, B. C. (2002). "A Comparative Study of Alternative Estimators for the Unbalanced Two-Way Error Component Regression Model." *Econometrics Journal* 5:480–493.
- Blundell, R., and Bond, S. (1998). "Initial Conditions and Moment Restrictions in Dynamic Panel Data Models." *Journal of Econometrics* 87:115–143.
- Buse, A. (1973). "Goodness of Fit in Generalized Least Squares Estimation." *American Statistician* 27:106–108.
- Cornwell, C., and Rupert, P. (1988). "Efficient Estimation with Panel Data: An Empirical Comparison of Instrumental Variables Estimators." *Journal of Applied Econometrics* 3:149–155.
- Cornwell, C., and Trumbull, W. N. (1994). "Estimating the Economic Problem of Crime with Panel Data." *Review of Econometrics and Statistics* 76:360–366.
- Da Silva, J. G. C. (1975). "The Analysis of Cross-Sectional Time Series Data." Ph.D. diss., Department of Statistics, North Carolina State University.
- Davidson, R., and MacKinnon, J. G. (1993). *Estimation and Inference in Econometrics*. New York: Oxford University Press.
- Feige, E. L. (1964). *The Demand for Liquid Assets: A Temporal Cross-Section Analysis*. Englewood Cliffs, NJ: Prentice-Hall.
- Fuller, W. A., and Battese, G. E. (1974). "Estimation of Linear Models with Crossed-Error Structure." *Journal of Econometrics* 2:67–78.

- Gardner, R. (1998). "Unobservable Individual Effects in Unbalanced Panel Data." *Economics Letters* 58:39–42.
- Greene, W. H. (1990). *Econometric Analysis*. New York: Macmillan.
- Greene, W. H. (2000). *Econometric Analysis*. 4th ed. Upper Saddle River, NJ: Prentice-Hall.
- Hausman, J. A. (1978). "Specification Tests in Econometrics." *Econometrica* 46:1251–1271.
- Hausman, J. A., and Taylor, W. E. (1981). "Panel Data and Unobservable Individual Effects." *Econometrica* 49:1377–1398.
- Hausman, J. A., and Taylor, W. E. (1982). "A Generalized Specification Test." *Economics Letters* 8:239–245.
- Nerlove, M. (1971). "Further Evidence on the Estimation of Dynamic Relations from a Time Series of Cross Sections." *Econometrica* 39:359–382.
- Newey, W. K., and West, D. W. (1994). "Automatic Lag Selection in Covariance Matrix Estimation." *Review of Economic Studies* 61:631–653.
- Nickell, S. J. (1981). "Biases in Dynamic Models with Fixed Effects." *Econometrica* 49:1417–1426.
- Parks, R. W. (1967). "Efficient Estimation of a System of Regression Equations When Disturbances Are Both Serially and Contemporaneously Correlated." *Journal of the American Statistical Association* 62:500–509.
- Swamy, P. A. V. B., and Arora, S. S. (1972). "The Exact Finite Sample Properties of the Estimators of Coefficients in the Error Components Regression Models." *Econometrica* 40:261–275.
- Theil, H. (1961). *Economic Forecasts and Policy*. 2nd ed. Amsterdam: North-Holland.
- Wallace, T., and Hussain, A. (1969). "The Use of Error Components Model in Combining Cross Section with Time Series Data." *Econometrica* 37:55–72.
- Wansbeek, T., and Kapteyn, A. (1989). "Estimation of the Error-Components Model with Incomplete Panels." *Journal of Econometrics* 41:341–361.
- White, H. (1980). "A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity." *Econometrica* 48:817–838.
- Windmeijer, F. (2005). "A Finite Sample Correction for the Variance of Linear Efficient Two-Step GMM Estimators." *Journal of Econometrics* 126:25–51.
- Wooldridge, J. M. (2002). *Econometric Analysis of Cross Section and Panel Data*. Cambridge, MA: MIT Press.
- Wu, D. M. (1973). "Alternative Tests of Independence between Stochastic Regressors and Disturbances." *Econometrica* 41:733–750.

Chapter 10

The CQLIM Procedure

Contents

Overview: CQLIM Procedure	438
PROC CQLIM Features	438
PROC CQLIM Compared with Other SAS Procedures	439
Using CAS Sessions and CAS Engine Librefs	440
Getting Started: CQLIM Procedure	441
Syntax: CQLIM Procedure	443
Functional Summary	443
PROC CQLIM Statement	445
BOUNDS Statement	447
BY Statement	447
CLASS Statement	448
DISPLAY Statement	448
DISPLAYOUT Statement	449
ENDOGENOUS Statement	450
FREQ Statement	452
HETERO Statement	452
INIT Statement	453
MODEL Statement	453
OUTPUT Statement	455
RESTRICT Statement	457
TEST Statement	458
WEIGHT Statement	459
Details: CQLIM Procedure	460
Ordinal Discrete Choice Modeling	460
Limited Dependent Variable Models	461
Stochastic Frontier Production and Cost Models	461
Heteroscedasticity	463
Bivariate Censored Dependent Variable Modeling	464
Multivariate Limited Dependent Models	465
Tests on Parameters	465
Output to SAS Data Table	466
Naming	470
ODS Table Names	471
Example: The CQLIM Procedure	472
Example 10.1: Model with Censoring	472
Example 10.2: Bivariate Probit Analysis	473
References	475

Overview: CQLIM Procedure

The CQLIM procedure is similar to the QLIM procedure in SAS/ETS software; PROC CQLIM requires SAS Cloud Analytic Services (CAS) in order to run.

Both procedures analyze univariate and multivariate limited dependent variable models, in which dependent variables take discrete values or are observed within only a limited range of values. When the dependent variable takes discrete values, you can use either a logit model, a probit model, or an ordered logit or probit model, depending on the distribution of the dependent variable. In addition to the discrete variable models, the CQLIM procedure includes Tobit models, truncated regression models, and multivariate models. The multivariate model can contain discrete choice and limited dependent variables in addition to linear continuous dependent variables.

Another class of limited dependent variable models that the CQLIM procedure supports is stochastic frontier models. These models were first introduced by Aigner, Lovell, and Schmidt (1977) and Meeusen and van den Broeck (1977); they are capable of analyzing technical inefficiencies in production functions or cost functions.

Unlike the QLIM procedure, which can run only on a single workstation, the CQLIM procedure takes advantage of the distributed computing environment of CAS, which enables it to distribute the optimization task to one or more nodes. In addition, each node can use one or more threads to perform the optimization on its subset of the data. When several nodes are used and each node uses several threads to carry out its part of the work, the result is a highly parallel computation that can provide a dramatic gain in performance.

The CQLIM procedure can use the maximum likelihood method to perform the inference of the available models. By default, it uses multiple threads to perform computations.

PROC CQLIM Features

The CQLIM procedure supports the following models:

- linear regression models with heteroscedasticity
- probit models with heteroscedasticity
- logit models with heteroscedasticity
- Tobit models with heteroscedasticity
- truncated regression models with heteroscedasticity
- bivariate probit models
- bivariate Tobit models
- multivariate limited dependent variables
- stochastic frontier production and cost models

In linear regression models with heteroscedasticity, the assumption that error variance is constant across observations is relaxed. The CQLIM procedure allows for a number of different linear and nonlinear variance specifications.

The CQLIM procedure is also capable of modeling problems in which the dependent variable takes discrete values. For these problems, you can use either a logit model or a probit model. In both cases, the dependent variable is modeled by using a binary distribution (typically, $y = 0, 1$). The main difference is that the logit model is characterized by an underlying logistic distribution, whereas the probit model has a Gaussian distribution. When the dependent variable takes more than two levels, the binary assumption is no longer valid. In these cases, you can use the ordered logit regression and the ordered probit regression. Assumptions about the error variance can also be relaxed in order to estimate a probit or logit model with heteroscedasticity.

PROC CQLIM also offers a class of models in which the dependent variable is censored or truncated from below, above, or both. When a continuous dependent variable is observed within only a certain range, and values outside this range are not available, PROC CQLIM offers a class of models that adjust for truncation. In some cases, the dependent variable is continuous within only a certain range, and all values outside this range are reported as being on its boundary. For example, if it is not possible to observe negative values, the value of the dependent variable is reported as equal to 0. Because the data are censored, ordinary least squares (OLS) results are inconsistent, and you cannot be certain that the predicted values from the model will fall in the appropriate region.

Most of the models in the CQLIM procedure can be extended to accommodate bivariate and multivariate scenarios.

The stochastic frontier production and cost models available in PROC CQLIM allow for random shocks of the production or cost. These models include a systematic positive component in the error term that adjusts for technical or cost inefficiency.

The CQLIM procedure can use the maximum likelihood method for both univariate and multivariate models. Initial starting values for the nonlinear optimizations are usually calculated by OLS.

PROC CQLIM Compared with Other SAS Procedures

The CQLIM procedure provides limited dependent data modeling functionality comparable to that of the HPQLIM and QLIM procedures in SAS/ETS software.

PROC CQLIM Compared with the HPQLIM Procedure

The functionality of the CQLIM procedure closely resembles that of the HPQLIM procedure, which is a high-performance procedure. The CQLIM procedure is the next generation of the HPQLIM procedure, and it was developed specifically for SAS Viya. Both procedures are designed to run on a cluster of machines that distribute the data and the computations. Both procedures perform computations in multiple threads on multiple machines.

The CQLIM procedure supports maximum likelihood estimation of all the models available in PROC HPQLIM.

PROC CQLIM Compared with the QLIM Procedure

The CQLIM procedure is specifically designed to operate in SAS Viya and performs computations in multiple threads on multiple machines. The capability of the QLIM procedure to perform computations in a distributed environment is limited to the parallel implementation of MCMC sampling algorithms for Bayesian analysis.

This release of the CQLIM procedure contains some, but not all, of the functionality available in the QLIM procedure. PROC QLIM provides the following features, which are not available in PROC CQLIM:

- sample selection models
- variable selection
- Bayesian analysis

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CQLIM Procedure

This example illustrates the use of the CQLIM procedure. The data were originally published by Mroz (1987), and the following DATA steps load a subset of the data into CAS. The assumption here is that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```

title1 'Estimating a Tobit Model';

data subset;
  input Hours Yrs_Ed Yrs_Exp @@;
  if Hours eq 0 then Lower=.;
  else                Lower=Hours;
datalines;
0 8 9 0 8 12 0 9 10 0 10 15 0 11 4 0 11 6
1000 12 1 1960 12 29 0 13 3 2100 13 36
3686 14 11 1920 14 38 0 15 14 1728 16 3
1568 16 19 1316 17 7 0 17 15
;

data mycas.subset(copies=1 promote=yes);
  set subset;
run;

```

In these data, Hours is the number of hours that a wife worked outside the household in a particular year, Yrs_Ed is years of education, and Yrs_Exp is years of work experience.

From the nature of the data, it is clear that there are a number of women who committed some positive number of hours to outside work ($y_i > 0$ is observed). There are also a number of women who did not work outside the household at all ($y_i = 0$ is observed). This yields the following model,

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}$$

where $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$ and $\mathbf{x}_i$ denotes the set of explanatory variables. The following statements fit a Tobit model to the number of hours worked, with years of education and years of work experience as covariates:

```

/--- Tobit Model ---/
proc cqlim data=mycas.subset;
  model hours = yrs_ed yrs_exp;
  endogenous hours ~ censored(lb=0);
run;

```

The output of the CQLIM procedure is shown in [Output 10.1](#).

Figure 10.1 Tobit Analysis Results**Estimating a Tobit Model****The CQLIM Procedure**

Model Fit Summary	
Dependent Variable	Hours
Number of Observations	17
Data Set	SUBSET
Log Likelihood	-74.937
Maximum Absolute Gradient	6.748E-6
Number of Iterations	5
Optimization Method	Newton-Raphson
AIC	157.874
SBC	161.2069
Covariance Estimation	Hessian

Convergence criterion (ABSGCONV=0.00001) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	-5595.665232	27.645559	-202.41	<.0001
Yrs_Ed	1	372.981321	53.962540	6.91	<.0001
Yrs_Exp	1	63.319498	36.536554	1.73	0.0831
_Sigma	1	1582.240492	389.785175	4.06	<.0001

The “Parameter Estimates” table contains four rows. The first three rows correspond to the vector estimate of the regression coefficients β . The last row is called `_Sigma`, which corresponds to the estimate of the error variance σ .

Syntax: CQLIM Procedure

The following statements are available in the CQLIM procedure:

```

PROC CQLIM options ;
  BOUNDS bound1 < , bound2 ... > ;
  BY variables ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  FREQ variable ;
  ENDOGENOUS variables ~ options ;
  HETERO dependent-variables ~ exogenous-variables / options ;
  INIT initvalue1 < , initvalue2 ... > ;
  MODEL dependent-variables = regressors / options ;
  OUTPUT OUT=CAS-libref-data-table < output-options > ;
  RESTRICT restriction1 < , restriction2 ... > ;
  TEST options ;
  WEIGHT variable < / option > ;

```

At least one MODEL statement is required. If you specify more than one MODEL statement, the CQLIM procedure estimates a system of models. If you specify more than one FREQ or WEIGHT statement, only the variable that is specified in the first instance of the statement is used.

Functional Summary

Table 10.1 summarizes the statements and options available in the CQLIM procedure.

Table 10.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the input data table	PROC CQLIM	DATA=
Writes predictions to an output data table	OUTPUT	OUT=
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies classification variables	CLASS	
Specifies a frequency variable	FREQ	
Specifies a weight variable	WEIGHT	NONNORMALIZE
Printing Control Options		
Prints the correlation matrix of the estimates	PROC CQLIM	CORRB
Prints the covariance matrix of the estimates	PROC CQLIM	COVB
Prints a summary iteration listing	PROC CQLIM	ITPRINT

Table 10.1 *continued*

Description	Statement	Option
Optimization Process Control Options		
Specifies the maximum number of iterations	PROC CQLIM	MAXITER=
Selects the iterative minimization method to use	PROC CQLIM	METHOD=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the method to calculate parameter covariance	PROC CQLIM	COVEST=
Specifies the number of draws for Monte Carlo integration	PROC CQLIM	NDRAW=
Suppresses the intercept parameter	MODEL	NOINT
Specifies a seed for pseudorandom number generation	PROC CQLIM	SEED=
Endogenous Variable Options		
Specifies a discrete variable	ENDOGENOUS	DISCRETE()
Specifies a censored variable	ENDOGENOUS	CENSORED()
Specifies a truncated variable	ENDOGENOUS	TRUNCATED()
Specifies a stochastic frontier variable	ENDOGENOUS	FRONTIER()
Heteroscedasticity Model Options		
Specifies the function for heteroscedasticity models	HETERO	LINK=
Squares the function for heteroscedasticity models	HETERO	SQUARE
Specifies no constant for heteroscedasticity models	HETERO	NOCONST
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	
Outputs the predicted values	OUTPUT	PREDICTED
Outputs the structured part	OUTPUT	XBETA
Outputs the residuals	OUTPUT	RESIDUAL
Outputs the error standard deviation	OUTPUT	ERRSTD
Outputs the marginal effects	OUTPUT	MARGINAL
Outputs the probability for the current response	OUTPUT	PROB
Outputs the probability for all responses	OUTPUT	PROBALL
Outputs the expected value	OUTPUT	EXPECTED
Outputs the conditional expected value	OUTPUT	CONDITIONAL
Outputs the inverse Mills ratio	OUTPUT	MILLS
Outputs the technical efficiency measures	OUTPUT	TE1
	OUTPUT	TE2

Table 10.1 *continued*

Description	Statement	Option
Test Request Options		
Requests Wald, Lagrange multiplier, and likelihood ratio tests	TEST	ALL
Requests the Wald test	TEST	WALD
Requests the Lagrange multiplier test	TEST	LM
Requests the likelihood ratio test	TEST	LR

PROC CQLIM Statement

PROC CQLIM < *options* > ;

The PROC CQLIM statement invokes the CQLIM procedure. You can specify the following *options*.

Data Table Options

DATA=*CAS-libref.data-table*

names the input data table for PROC CQLIM to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 440.

data-table specifies the name of the input data table.

Printing Options

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

ITPRINT

prints the initial parameter estimates, convergence criteria, and all constraints of the optimization. At each iteration, the objective function value, step size, maximum gradient, and slope of search direction are also printed.

Model Estimation Options

COVEST=HESSIAN | OP | QML

specifies the method to use for calculating the covariance matrix of parameter estimates. You can specify the following covariance options:

HESSIAN	specifies the covariance from the inverse Hessian matrix.
OP	specifies the covariance from the outer product matrix.
QML	specifies the covariance from the outer product and Hessian matrices (the quasi-maximum likelihood estimates).

By default, COVEST=HESSIAN.

NDRAW=*value*

specifies the number of draws for Monte Carlo integration.

SEED=*value*

specifies a seed for pseudorandom number generation in Monte Carlo integration.

Optimization Control Options

PROC CQLIM uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options*:

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

METHOD=CONGRA | DBLDOG | NEWRAP | NONE | NRRIDG | QUANEW | TRUREG

specifies the iterative minimization method to use. You can specify the following methods:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

BOUNDS Statement

BOUNDS *bound1* < , *bound2* ... > ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. BOUNDS statement constraints refer to the parameters that are estimated by the CQLIM procedure. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameters, constants, and inequality operators. Parameters that are associated with regressor variables are referred to by the names of the corresponding regressor variables. Specify each bound as follows:

item operator item < *operator item* < *operator item* ... > >

Each *item* is a constant, the name of a parameter, or a list of parameter names. For more information about how parameters are named in the CQLIM procedure, see the section “[Naming of Parameters](#)” on page 470. Each *operator* is <, >, <=, or >=.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints; however, the BOUNDS statement provides a simpler syntax for specifying these types of constraints. For more information, see the section “[RESTRICT Statement](#)” on page 457.

The following BOUNDS statement constrains the estimates of the parameters that are associated with the variable *ttime* and the variables *x1* through *x10* to be between 0 and 1. The following example illustrates the use of parameter lists to specify boundary constraints:

```
bounds 0 < ttime x1-x10 < 1;
```

The following BOUNDS statement constrains the estimates of the correlation (*_RHO*) and sigma (*_SIGMA*) in the bivariate model:

```
bounds _rho >= 0, _sigma.y1 > 1, _sigma.y2 < 5;
```

BY Statement

BY *variables* ;

A BY statement can be used in PROC CQLIM to obtain separate analyses of observations in groups defined by the BY variables.

CLASS Statement

CLASS *variable* < (*options*) > . . . < *variable* < (*options*) > > < / *global-options* > ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You are not required to list the response variable for discrete choice models in the CLASS statement. Table 10.2 summarizes the values that you can use for either an *option* or a *global-option*. By default, REF=LAST. The options are fully documented in the section “CLASS Statement” on page 14 in Chapter 3, “Shared Concepts.”

Table 10.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “ODS Table Names” on page 471. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *Selection-Summary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

<i>key=value</i>	specifies <i>key</i> as the ODS table name, path, or partial pathname, and specifies <i>value</i> as the CAS output table name.
<i>key</i>	specifies <i>key</i> as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 471. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

ENDOGENOUS Statement

ENDOGENOUS *variables ~ options* ;

The ENDOGENOUS statement specifies the type of dependent variables that appear on the left-hand side of the equation. Endogenous variables listed refer to the dependent variables that appear on the left-hand side of the equation. You can specify the following options.

Discrete Variable Options

DISCRETE < (*discrete-options*) >

specifies that the endogenous variables in this statement be discrete. You can specify the following *discrete-options*:

DISTRIBUTION=LOGISTIC | NORMAL

DIST=LOGISTIC | NORMAL

D=LOGISTIC | NORMAL

specifies the cumulative distribution function to use to model the response probabilities. You can specify the following distribution types:

LOGISTIC specifies the logistic distribution for the logit model.

NORMAL specifies the normal distribution for the probit model.

By default, DISTRIBUTION=NORMAL.

If a multivariate model is specified, logistic distribution is not allowed. Only normal distribution is supported.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of the discrete variables that you specify in the ENDOGENOUS statement. This sort order determines which parameters in the model correspond to each level in the data. You can specify the following sort orders:

DATA sorts levels by order of appearance in the input data table.

FORMATTED sorts levels by formatted value. The sort order is machine-dependent.

FREQ sorts levels by descending frequency count; levels that have the most observations come first in the order.

INTERNAL sorts levels by unformatted value. The sort order is machine-dependent.

By default, ORDER=FORMATTED. For more information about sort order, see the chapter on the SORT procedure in the *Base SAS Procedures Guide*.

Censored Variable Options

CENSORED (*censored-options*)

censors the endogenous variables in this statement. You can specify the following *censored-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the censored variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the censored variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Truncated Variable Options

TRUNCATED (*truncated-options*)

truncates the endogenous variables in the statement. You can specify the following *truncated-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Stochastic Frontier Variable Options

FRONTIER <(*frontier-options*)>

specifies a stochastic frontier model. You can specify the following *frontier-options*:

COST

specifies that the estimated model be a cost function.

PRODUCTION

specifies that the estimated model be a production function.

TYPE=EXPONENTIAL | HALF | TRUNCATED

specifies the model type. You can specify the following types:

EXPONENTIAL specifies an exponential model.

HALF specifies a half-normal model.

TRUNCATED specifies a truncated normal model.

If neither PRODUCTION nor COST is specified, a production function is estimated by default.

FREQ Statement

FREQ *variable* ;

The FREQ statement identifies a variable that contains the frequency of occurrence of each observation. PROC CQLIM treats each observation as if it appeared n times, where n is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the model fitting. When the FREQ statement is not specified, each observation is assigned a frequency of 1. If you specify more than one FREQ statement, then only the first FREQ statement is used.

HETERO Statement

HETERO *dependent-variables ~ exogenous-variables* </ options > ;

The HETERO statement specifies variables that are related to the heteroscedasticity of the residuals and the way that these variables are used to model the error variance. PROC CQLIM supports the following heteroscedastic regression model:

$$y_i = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$\epsilon_i \sim N(0, \sigma_i^2)$$

For more information about the specification of functional forms, see the section “[Heteroscedasticity](#)” on page 463.

You can use the HETERO statement only for univariate models.

The following *options* specify the functional forms of heteroscedasticity:

LINK=EXP | LINEAR

specifies the functional form. You can specify the following values:

EXP specifies the exponential link function:

$$\sigma_i^2 = \sigma^2(1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma}))$$

LINEAR specifies the linear link function:

$$\sigma_i^2 = \sigma^2(1 + \mathbf{z}_i' \boldsymbol{\gamma})$$

By default, LINK=EXP.

NOCONST

specifies that no constant appear in the exponential heteroscedasticity model:

$$\sigma_i^2 = \sigma^2 \exp(\mathbf{z}_i' \boldsymbol{\gamma})$$

SQUARE

estimates the model by using the square of the linear heteroscedasticity function. For example, you can specify the heteroscedasticity function

$$\sigma_i^2 = \sigma^2(1 + (\mathbf{z}_i' \boldsymbol{\gamma})^2)$$

by using the following code:

```
model y = x1 x2 / censored(lb=0);
hetero y ~ z1 / link=linear square;
```

The SQUARE option does not apply to the exponential heteroscedasticity function, because the square of an exponential function of $\mathbf{z}_i' \boldsymbol{\gamma}$ is the same as the exponential of $2\mathbf{z}_i' \boldsymbol{\gamma}$. Hence, the only difference is that all $\boldsymbol{\gamma}$ estimates are divided by two.

This option is ignored if you do not specify the LINK= option.

INIT Statement

INIT *initvalue1* < , *initvalue2* ... > ;

The INIT statement sets initial values for parameters in the optimization. You can specify any number of INIT statements.

Each *initvalue* is written as a parameter or parameter list, followed by an optional equality operator (=), followed by a number:

parameter <=> *number*

MODEL Statement

MODEL *dependent-variables* < (*order-options*) > = *regressors* < / *options* > ;

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model.

You can specify the following *option* after a slash (/):

NOINT

suppresses the intercept parameter.

You can also specify the following endogenous variable options, which are the same as the options in the ENDOGENOUS statement. If you specify an endogenous variable option in both the MODEL statement and the ENDOGENOUS statement, only the option in the ENDOGENOUS statement is used.

Discrete Variable Options

DISCRETE < (*discrete-options*) >

specifies that the endogenous variables in the MODEL statement be discrete. You can specify the following *discrete-options*:

DISTRIBUTION=LOGISTIC | NORMAL

DIST=LOGISTIC | NORMAL

D=LOGISTIC | NORMAL

specifies the cumulative distribution function to use to model the response probabilities. You can specify the following distribution types:

LOGISTIC specifies the logistic distribution for the logit model.

NORMAL specifies the normal distribution for the probit model.

By default, DISTRIBUTION=NORMAL.

If a multivariate model is specified, logistic distribution is not allowed. Only normal distribution is supported.

ORDER=FORMATTED | FREQ | INTERNAL

specifies the sort order for the levels of the discrete variables that you specify in the ENDOGENOUS statement. The sort order determines which parameters in the model correspond to each level in the data. You can specify the following sort orders:

FORMATTED sorts levels by formatted value. The sort order is machine-dependent.

FREQ sorts levels by descending frequency count; levels that have the most observations come first in the order.

INTERNAL sorts levels by unformatted value. The sort order is machine-dependent.

By default, ORDER=FORMATTED. For more information about sort order, see the chapter on the SORT procedure in the *Base SAS Procedures Guide*.

Censored Variable Options

CENSORED < (*censored-options*) >

censors the endogenous variables in this statement. You can specify the following *censored-options*:

LB=value | variable

LOWERBOUND=value | variable

specifies the lower bound of the censored variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=value | variable

UPPERBOUND=value | variable

specifies the upper bound of the censored variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Truncated Variable Options

TRUNCATED <(*truncated-options*)>

truncates the endogenous variables in this statement. You can specify the following *truncated-options*:

LB=*value* | *variable*

LOWERBOUND=*value* | *variable*

specifies the lower bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no lower bound is set. By default, no lower bound is set.

UB=*value* | *variable*

UPPERBOUND=*value* | *variable*

specifies the upper bound of the truncated variables. If *value* is missing or the value of *variable* is missing, no upper bound is set. By default, no upper bound is set.

Stochastic Frontier Variable Options

FRONTIER <(*frontier-options*)>

specifies the stochastic frontier model. You can specify the following *frontier-options*:

COST

specifies that the estimated model be a cost function.

PRODUCTION

specifies that the estimated model be a production function.

TYPE=EXPONENTIAL | **HALF** | **TRUNCATED**

specifies the model type.

EXPONENTIAL specifies an exponential model.

HALF specifies a half-normal model.

TRUNCATED specifies a truncated normal model.

If neither PRODUCTION nor COST is specified, a production function is estimated by default.

OUTPUT Statement

OUTPUT OUT=CAS.libref.data-table < *output-options* > ;

The OUTPUT statement creates a new SAS data table to contain variables that you specify using the COPYVAR option and the following quantities of interest if they are specified as *output-options*: estimates of $x'\beta$, predicted value, residual, marginal effects, probability, standard deviation of the error, expected value, conditional expected value, technical efficiency measures, and inverse Mills ratio. When the response values are missing for an observation, all output estimates except the residual for that observation are still computed as long as none of the explanatory variables are missing. This enables you to compute these statistics for prediction. You can specify only one OUTPUT statement.

You must specify the OUT= option:

OUT=CAS-libref.data-table

names the output data table for PROC CQLIM to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>CAS-libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 440.
<i>data-table</i>	specifies the name of the output data table.

You can specify one or more of the following *output-options*:

CONDITIONAL

outputs estimates of conditional expected values of continuous endogenous variables.

COPYVAR=SAS-variable-names**COPYVARS=(SAS-variable-names)**

adds any SAS variables that appear in the input data table to the output data table.

ERRSTD

outputs estimates of σ_j , the standard deviation of the error term.

EXPECTED

outputs estimates of expected values of continuous endogenous variables.

MARGINAL

outputs marginal effects.

MILLS

outputs estimates of inverse Mills ratios of censored or truncated continuous variables and binary discrete variables.

PREDICTED

outputs estimates of predicted endogenous variables.

PROB

outputs estimates of probability of discrete endogenous variables that take the current observed responses.

PROBALL

outputs estimates of probability of discrete endogenous variables for all possible responses.

RESIDUAL

outputs estimates of residuals of continuous endogenous variables.

TE1

outputs estimates of technical efficiency for each producer in the stochastic frontier model that is suggested by Battese and Coelli (1988).

TE2

outputs estimates of technical efficiency for each producer in the stochastic frontier model that is suggested by Jondrow et al. (1982).

XBETA

outputs estimates of $\mathbf{x}'\boldsymbol{\beta}$.

RESTRICT Statement

RESTRICT *restriction1* <, *restriction2* ... > ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements, but the number of restrictions that are imposed is limited by the number of regressors.

Each *restriction* is written as an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression:

expression operator expression

The *operator* can be =, <, >, <=, or >=. The *operator* and second *expression* are optional.

Restriction expressions can be composed of parameter names; multiplication (*), addition (+), and subtraction (−) operators; and constants. Parameters that are named in restriction expressions must be among the parameters that are estimated by the model. Parameters that are associated with a regressor variable are referred to by the name of the corresponding regressor variable. The restriction expressions must be a linear function of the parameters.

The following statements illustrate the use of the RESTRICT statement:

```
proc cqlim data=mycas.dataset;
  model y = x1-x10 / censored(lb=0);
  restrict x1*x2 <= x2 + x3;
run;
```

The RESTRICT statement can also be used to impose cross-equation restrictions in multivariate models. The following RESTRICT statement imposes an equality restriction on coefficients of x_1 in equation y_1 and x_1 in equation y_2 :

```
proc cqlim data=mycas.dataset;
  model y1 = x1-x10 / discrete;
  model y2 = x1-x4 / discrete;
  restrict y1.x1=y2.x1;
run;
```

TEST Statement

`<label:> TEST <'string'> equation <, equation .../ options > ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the regression parameters in the preceding MODEL statement. Each equation specifies a linear hypothesis to be tested. All hypotheses in one TEST statement are tested jointly. Variable names in the equations must correspond to regressors in the preceding MODEL statement, and each name represents the coefficient of the corresponding regressor. Use the keyword INTERCEPT for a test that includes a constant.

You can specify the following *options* after the slash (/):

ALL

requests Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

The following statements illustrate the use of the TEST statement (note the use of the INTERCEPT keyword in the second TEST statement):

```
proc cqlim;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0;
  test_int: test intercept = 0, x3 = 0;
run;
```

The first TEST statement investigates the joint hypothesis that

$$\beta_1 = 0$$

and

$$0.5\beta_2 + 2\beta_3 = 0$$

In case there is more than one MODEL statement in one CQLIM procedure, then TEST statement is capable of testing cross equation restrictions. Each parameter reference should be preceded by the name of the dependent variable of the particular model and the dot sign. For example,

```
proc cqlim;
  model y1 = x1 x2 x3;
  model y2 = x3 x5 x6;
  test y1.x1 + y2.x6 = 1;
run;
```

This cross equation test investigates the null hypothesis that

$$\beta_{1,1} + \beta_{2,3} = 1$$

in the system of equations

$$y_{1,i} = \alpha_1 + \beta_{1,1}x_{1,i} + \beta_{1,2}x_{2,i} + \beta_{1,3}x_{3,i}$$

$$y_{2,i} = \alpha_2 + \beta_{2,1}x_{3,i} + \beta_{2,2}x_{5,i} + \beta_{2,3}x_{6,i}$$

Only linear equality restrictions and tests are permitted in PROC CQLIM. Test expressions can be composed only of algebraic operations that involve the addition symbol (+), subtraction symbol (–), and multiplication symbol (*).

The TEST statement accepts labels that are reproduced in the printed output. You can label a TEST statement in two ways: you can specify a label followed by a colon before the TEST keyword, or you can specify a string in quotation marks after the TEST keyword. If you specify both a label before the TEST keyword and a quoted string after the keyword, PROC CQLIM uses the label that precedes the colon. If no label or quoted string is specified, PROC CQLIM labels the test automatically.

WEIGHT Statement

WEIGHT *variable* *</option>* ;

The WEIGHT statement specifies a variable that supplies weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

You can add the following *option* after a slash (/):

NONNORMALIZE

specifies that the weights must be used as is. When you omit this option, the weights are normalized so that they add up to the actual sample size. Weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^n w_i}$, where n is the sample size.

Details: CQLIM Procedure

Ordinal Discrete Choice Modeling

Binary Probit and Logit Model

The binary choice model is

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where the value of the latent dependent variable, y_i^* , is observed only as follows:

$$\begin{aligned} y_i &= 1 && \text{if } y_i^* > 0 \\ &= 0 && \text{otherwise} \end{aligned}$$

The disturbance, ϵ_i , of the probit model has a standard normal distribution with the cumulative distribution function (CDF)

$$\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp(-t^2/2) dt$$

The disturbance of the logit model has a standard logistic distribution with the CDF

$$\Lambda(x) = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)}$$

The binary discrete choice model has the following probability that the event $\{y_i = 1\}$ occurs:

$$P(y_i = 1) = F(\mathbf{x}_i' \boldsymbol{\beta}) = \begin{cases} \Phi(\mathbf{x}_i' \boldsymbol{\beta}) & \text{(probit)} \\ \Lambda(\mathbf{x}_i' \boldsymbol{\beta}) & \text{(logit)} \end{cases}$$

For more information, see [SAS/ETS User's Guide](#).

Ordinal Probit/Logit

When the dependent variable is observed in sequence with M categories, binary discrete choice modeling is not appropriate for data analysis. McKelvey and Zavoina (1975) propose the ordinal (or ordered) probit model.

Consider the regression equation

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where error disturbances, ϵ_i , have the distribution function F . The unobserved continuous random variable, y_i^* , is identified as M categories. Suppose there are $M + 1$ real numbers, $\mu_0, \dots, \mu_M$, where $\mu_0 = -\infty$, $\mu_1 = 0$, $\mu_M = \infty$, and $\mu_0 \leq \mu_1 \leq \dots \leq \mu_M$. Define

$$R_{i,j} = \mu_j - \mathbf{x}_i' \boldsymbol{\beta}$$

The probability that the unobserved dependent variable is contained in the j th category can be written as

$$P[\mu_{j-1} < y_i^* \leq \mu_j] = F(R_{i,j}) - F(R_{i,j-1})$$

For more information, see [SAS/ETS User's Guide](#).

Limited Dependent Variable Models

Censored Regression Models

When the dependent variable is censored, all values within a certain range are transformed to a single value. For example, the standard Tobit model can be defined as

$$y_i^* = \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

$$y_i = \begin{cases} y_i^* & \text{if } y_i^* > 0 \\ 0 & \text{if } y_i^* \leq 0 \end{cases}$$

where $\epsilon_i \sim \text{iid}N(0, \sigma^2)$.

The Tobit model can be generalized to handle observation-by-observation censoring. The model that is censored on both the lower and upper limits can be defined as

$$y_i = \begin{cases} R_i & \text{if } y_i^* \geq R_i \\ y_i^* & \text{if } L_i < y_i^* < R_i \\ L_i & \text{if } y_i^* \leq L_i \end{cases}$$

For more information, see *SAS/ETS User's Guide*.

Truncated Regression Models

In a truncated model, the observed sample is a subset of the population in which the dependent variable falls within a certain range. For example, when neither a dependent variable nor exogenous variables are observed for $y_i^* \leq 0$, the truncated regression model can be specified as

$$\ell = \sum_{i \in \{y_i > 0\}} \left\{ -\ln \Phi(\mathbf{x}_i' \boldsymbol{\beta} / \sigma) + \ln \left[\frac{\phi((y_i - \mathbf{x}_i' \boldsymbol{\beta}) / \sigma)}{\sigma} \right] \right\}$$

For more information, see *SAS/ETS User's Guide*.

Stochastic Frontier Production and Cost Models

Stochastic frontier production models were first developed by Aigner, Lovell, and Schmidt (1977); Meeusen and van den Broeck (1977). Specification of these models allows for random shocks of the production or cost but also includes a term for technical or cost inefficiency. Assuming that the production function takes a log-linear Cobb-Douglas form, the stochastic frontier production model can be written as

$$\ln(y_i) = \beta_0 + \sum_n \beta_n \ln(x_{ni}) + \epsilon_i$$

where $\epsilon_i = v_i - u_i$. The v_i term represents the stochastic error component, and the u_i term represents the nonnegative, technical inefficiency error component. The v_i error component is assumed to be distributed iid normal and independent from u_i . If $u_i > 0$, the error term ϵ_i is negatively skewed and represents technical inefficiency. If $u_i < 0$, the error term ϵ_i is positively skewed and represents cost inefficiency. PROC CQLIM models the u_i error component as a half-normal, exponential, or truncated normal distribution.

The Normal-Half-Normal Model

When v_i is iid $N(0, \sigma_v^2)$ in a normal-half-normal model, u_i is iid $N^+(0, \sigma_u^2)$, with v_i and u_i independent of each other. Given the independence of error terms, the joint density of v and u can be written as

$$f(u, v) = \frac{2}{2\pi\sigma_u\sigma_v} \exp \left\{ -\frac{u^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2} \right\}$$

Substituting $v = \epsilon + u$ into the preceding equation and integrating u out gives

$$f(\epsilon) = \frac{2}{\sigma} \phi \left(\frac{\epsilon}{\sigma} \right) \Phi \left(-\frac{\epsilon\lambda}{\sigma} \right)$$

where $\lambda = \sigma_u/\sigma_v$ and $\sigma = \sqrt{\sigma_u^2 + \sigma_v^2}$.

In the case of a stochastic frontier cost model, $v = \epsilon - u$ and

$$f(\epsilon) = \frac{2}{\sigma} \phi \left(\frac{\epsilon}{\sigma} \right) \Phi \left(\frac{\epsilon\lambda}{\sigma} \right)$$

For more information, see *SAS/ETS User's Guide*.

The Normal-Exponential Model

Under the normal-exponential model, v_i is iid $N(0, \sigma_v^2)$ and u_i is iid exponential. Given the independence of the error term components u_i and v_i , the joint density of v and u can be written as

$$f(u, v) = \frac{1}{\sqrt{2\pi}\sigma_u\sigma_v} \exp \left\{ -\frac{u}{\sigma_u} - \frac{v^2}{2\sigma_v^2} \right\}$$

The marginal density function of ϵ for the production function is

$$f(\epsilon) = \left(\frac{1}{\sigma_u} \right) \Phi \left(-\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u} \right) \exp \left\{ \frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2} \right\}$$

The marginal density function for the cost function is equal to

$$f(\epsilon) = \left(\frac{1}{\sigma_u} \right) \Phi \left(\frac{\epsilon}{\sigma_v} - \frac{\sigma_v}{\sigma_u} \right) \exp \left\{ -\frac{\epsilon}{\sigma_u} + \frac{\sigma_v^2}{2\sigma_u^2} \right\}$$

For more information, see *SAS/ETS User's Guide*.

The Normal–Truncated Normal Model

The normal–truncated normal model is a generalization of the normal-half-normal model that allows the mean of u_i to differ from 0. Under the normal–truncated normal model, the error term component v_i is iid $N^+(0, \sigma_v^2)$, and the error term component u_i is iid $N(\mu, \sigma_u^2)$. The joint density of v_i and u_i can be written as

$$f(u, v) = \frac{1}{\sqrt{2\pi}\sigma_u\sigma_v\Phi(\mu/\sigma_u)} \exp\left\{-\frac{(u-\mu)^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2}\right\}$$

The marginal density function of ϵ for the production function is

$$f(\epsilon) = \frac{1}{\sigma}\phi\left(\frac{\epsilon + \mu}{\sigma}\right)\Phi\left(\frac{\mu}{\sigma\lambda} - \frac{\epsilon\lambda}{\sigma}\right)\left[\Phi\left(\frac{\mu}{\sigma_u}\right)\right]^{-1}$$

The marginal density function for the cost function is

$$f(\epsilon) = \frac{1}{\sigma}\phi\left(\frac{\epsilon - \mu}{\sigma}\right)\Phi\left(\frac{\mu}{\sigma\lambda} + \frac{\epsilon\lambda}{\sigma}\right)\left[\Phi\left(\frac{\mu}{\sigma_u}\right)\right]^{-1}$$

For more information, see *SAS/ETS User's Guide*.

For more information about normal-half-normal, normal-exponential, and normal–truncated normal models, see Kumbhakar and Lovell (2000); Coelli, Prasada Rao, and Battese (1998).

Heteroscedasticity

If the variance of regression disturbance ϵ_i is heteroscedastic, the variance can be specified as a function of variables

$$E(\epsilon_i^2) = \sigma_i^2 = f(\mathbf{z}_i'\boldsymbol{\gamma})$$

Table 10.3 shows various functional forms of heteroscedasticity and the corresponding options to request each model.

Table 10.3 Specification Summary for Modeling Heteroscedasticity

Number	Model	Options
1	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma}))$	LINK=EXP (default)
2	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2 \exp(\mathbf{z}_i'\boldsymbol{\gamma})$	LINK=EXP NOCONST
3	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + \sum_{l=1}^L \gamma_l z_{li})$	LINK=LINEAR
4	$f(\mathbf{z}_i'\boldsymbol{\gamma}) = \sigma^2(1 + (\sum_{l=1}^L \gamma_l z_{li})^2)$	LINK=LINEAR SQUARE

In model 3, variance of some observations might be negative. Although the CQLIM procedure assigns a large penalty to move the optimization away from such a region, the optimization might not be able to improve the objective function value and might become locked in the region. Signs of such an outcome include extremely small likelihood values or missing standard errors in the estimates. In model 2, variance is guaranteed to be greater than or equal to 0, but variances of some observations might be very close to 0. In these scenarios, standard errors might be missing. Models 1 and 4 do not have such problems. Variances in these models are always positive and never close to 0.

Bivariate Censored Dependent Variable Modeling

The generic form of a bivariate censored dependent variable model is

$$\begin{aligned} y_{1i}^* &= \mathbf{x}_{1i}'\boldsymbol{\beta}_1 + \epsilon_{1i} \\ y_{2i}^* &= \mathbf{x}_{2i}'\boldsymbol{\beta}_2 + \epsilon_{2i} \end{aligned}$$

where the disturbances, ϵ_{1i} and ϵ_{2i} , have a joint normal distribution with zero mean, standard deviations σ_1 and σ_2 , and correlation ρ . The variables y_1^* and y_2^* are latent variables. The dependent variables y_1 and y_2 might or might not be censored at the edges of the bivariate interval $\{[L_1, R_1], [L_2, R_2]\}$, depending on the behavior of the latent variables y_1^* and y_2^* :

$$y_{1i} = \begin{cases} R_1 & \text{if } R_1 < y_{1i}^* \\ y_{1i}^* & \text{if } L_1 \leq y_{1i}^* \leq R_1 \\ L_1 & \text{if } y_{1i}^* < L_1 \end{cases}$$

$$y_{2i} = \begin{cases} R_2 & \text{if } R_2 < y_{2i}^* \\ y_{2i}^* & \text{if } L_2 \leq y_{2i}^* \leq R_2 \\ L_2 & \text{if } y_{2i}^* < L_2 \end{cases}$$

There are three cases for the log likelihood of (y_{1i}, y_{2i}) . In the first case, $y_{1i} = y_{1i}^*$ and $y_{2i} = y_{2i}^*$. That is, both dependent variables are uncensored. The log likelihood is computed from a bivariate normal density,

$$\ell_i = \ln [\text{pdf}(y_{1i}^*, y_{2i}^*)] = \ln \left[\phi_2 \left(\frac{y_{1i} - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1}, \frac{y_{2i} - \mathbf{x}_{2i}'\boldsymbol{\beta}_2}{\sigma_2}, \rho \right) \right] - \ln \sigma_1 - \ln \sigma_2$$

where $\phi_2(u, v, \rho)$ is the probability density function (pdf) for a standardized bivariate normal distribution with correlation ρ ,

$$\phi_2(u, v, \rho) = \frac{e^{-(1/2)(u^2 + v^2 - 2\rho uv)/(1 - \rho^2)}}{2\pi(1 - \rho^2)^{1/2}}$$

In the second case, one dependent variable is censored and the other is not. For example, if $y_{1i} = y_{1i}^*$ and $y_{2i} = L_2$, then the log likelihood is computed as

$$\begin{aligned} \ell_i &= \ln \left[\int_{-\infty}^{L_2} \text{pdf}(y_{1i}^*, y_{2i}^*) dy_{2i}^* \right] = \ln \left[\int_{-\infty}^{L_2} \text{pdf}(y_{2i}^* | y_{1i}^*) \text{pdf}(y_{1i}^*) dy_{2i}^* \right] \\ &= \ln \left[\phi \left(\frac{y_{1i}^* - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1} \right) \right] - \ln \sigma_1 + \ln \left[\Phi \left(\frac{L_2 - \mathbf{x}_{2i}'\boldsymbol{\beta}_2 - \sigma_2 \rho \frac{y_{1i}^* - \mathbf{x}_{1i}'\boldsymbol{\beta}_1}{\sigma_1}}{\sigma_2 \sqrt{1 - \rho^2}} \right) \right] \end{aligned}$$

where ϕ and Φ are the probability density function and the cumulative probability function for a standardized univariate normal distribution, respectively.

In the third case, both dependent variables are censored. For example, if $y_{1i} = R_1$ and $y_{2i} = L_2$, then the log likelihood is

$$\ell_i = \ln \left[\int_{u=\frac{R_1 - \mathbf{x}_1' \boldsymbol{\beta}_1}{\sigma_1}}^{\infty} \int_{v=-\infty}^{\frac{L_2 - \mathbf{x}_2' \boldsymbol{\beta}_2}{\sigma_2}} \phi_2(u, v, \rho) du dv \right]$$

Multivariate Limited Dependent Models

Multivariate models are similar to bivariate models. The generic form of the multivariate limited dependent variable model is

$$\begin{aligned} y_{1i}^* &= \mathbf{x}_{1i}' \boldsymbol{\beta}_1 + \epsilon_{1i} \\ y_{2i}^* &= \mathbf{x}_{2i}' \boldsymbol{\beta}_2 + \epsilon_{2i} \\ &\vdots \\ y_{mi}^* &= \mathbf{x}_{mi}' \boldsymbol{\beta}_m + \epsilon_{mi} \end{aligned}$$

where m is the number of models to be estimated. The vector ϵ has a multivariate normal distribution with mean 0 and variance-covariance matrix Σ . As with bivariate models, calculating the likelihood can involve computing multivariate normal integrations. This is done using Monte Carlo integration. (See Genz 1992; Hajivassiliou and McFadden 1998.)

When the number of equations, N , increases in a system, the number of parameters increases at the rate of N^2 because of the correlation matrix. When the number of parameters is large, sometimes the optimization converges but some of the standard deviations are missing. This usually means that the model is overparameterized. The default method for computing the covariance is to use the inverse Hessian matrix. The Hessian is computed by finite differences, and in overparameterized cases, the inverse cannot be computed. It is recommended that you reduce the number of parameters in such cases. Sometimes it might also help to use the outer product covariance matrix (the COVEST=OP option in the PROC CQLIM statement).

Tests on Parameters

In general, the tested hypothesis can be written as

$$H_0 : \mathbf{h}(\theta) = 0$$

where $\mathbf{h}(\theta)$ is an $r \times 1$ vector-valued function of the parameters θ given by the r expressions that are specified in the TEST statement.

Let $\hat{V}$ be the estimate of the covariance matrix of $\hat{\theta}$. Let $\hat{\theta}$ be the unconstrained estimate of θ , and let $\tilde{\theta}$ be the constrained estimate of θ such that $\mathbf{h}(\tilde{\theta}) = 0$. Let

$$A(\theta) = \partial \mathbf{h}(\theta) / \partial \theta \big|_{\hat{\theta}}$$

Using this notation, the test statistics for the three types of tests are computed as follows:

- The Wald test statistic is defined as

$$W = h'(\hat{\theta}) \left(A(\hat{\theta}) \hat{V} A'(\hat{\theta}) \right)^{-1} h(\hat{\theta})$$

- The Lagrange multiplier test statistic is

$$LM = \lambda' A(\tilde{\theta}) \tilde{V} A'(\tilde{\theta}) \lambda$$

where λ is the vector of Lagrange multipliers from the computation of the restricted estimate $\tilde{\theta}$.

- The likelihood ratio test statistic is

$$LR = 2 \left(L(\hat{\theta}) - L(\tilde{\theta}) \right)$$

where $\tilde{\theta}$ represents the constrained estimate of θ and L is the concentrated log-likelihood value.

The following statements use the TEST statement to perform a likelihood ratio test:

```
proc cqlim;
  model y = x1 x2 x3;
  test x1 = 0, x2 * .5 + 2 * x3 = 0 /lr;
run;
```

For more information, see *SAS/ETS User's Guide*.

Output to SAS Data Table

XBeta, Predicted, and Residual

Xbeta is the structural part on the right-hand side of the model. The predicted value is the predicted dependent variable value. For censored variables, if the predicted value is outside the boundaries, it is reported as the closest boundary. For discrete variables, it is the level whose boundaries Xbeta falls between. The residual is defined only for continuous variables and is defined as

$$\text{Residual} = \text{Observed} - \text{Predicted}$$

Error Standard Deviation

The error standard deviation is σ_i in the model. It varies only when the HETERO statement is used.

Marginal Effects

A marginal effect is defined as a contribution of one control variable to the response variable. For a binary choice model with two response categories, $\mu_0 = -\infty$, $\mu_1 = 0$, and $\mu_2 = \infty$. For an ordinal response model with M response categories ($\mu_0, \dots, \mu_M$), define

$$R_{i,j} = \mu_j - \mathbf{x}_i' \boldsymbol{\beta}$$

The probability that the unobserved dependent variable is contained in the j th category can be written as

$$P[\mu_{j-1} < y_i^* \leq \mu_j] = F(R_{i,j}) - F(R_{i,j-1})$$

The marginal effect of changes in the regressors on the probability of $y_i = j$ is then

$$\frac{\partial \text{Prob}[y_i = j]}{\partial \mathbf{x}} = [f(\mu_{j-1} - \mathbf{x}'_i \boldsymbol{\beta}) - f(\mu_j - \mathbf{x}'_i \boldsymbol{\beta})] \boldsymbol{\beta}$$

where $f(x) = \frac{dF(x)}{dx}$. In particular,

$$f(x) = \frac{dF(x)}{dx} = \begin{cases} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} & (\text{probit}) \\ \frac{e^{-x}}{[1+e^{(-x)}]^2} & (\text{logit}) \end{cases}$$

The marginal effects in the truncated regression model are

$$\frac{\partial E[y_i | L_i < y_i^* < R_i]}{\partial \mathbf{x}} = \boldsymbol{\beta} \left[1 - \frac{(\phi(a_i) - \phi(b_i))^2}{(\Phi(b_i) - \Phi(a_i))^2} + \frac{a_i \phi(a_i) - b_i \phi(b_i)}{\Phi(b_i) - \Phi(a_i)} \right]$$

where $a_i = \frac{L_i - \mathbf{x}'_i \boldsymbol{\beta}}{\sigma_i}$ and $b_i = \frac{R_i - \mathbf{x}'_i \boldsymbol{\beta}}{\sigma_i}$.

The marginal effects in the censored regression model are

$$\frac{\partial E[y | \mathbf{x}_i]}{\partial \mathbf{x}} = \boldsymbol{\beta} \times \text{Prob}[L_i < y_i^* < R_i]$$

For models that involve higher-order or interaction variables, the marginal effects are calculated by treating these variables as independent explanatory variables. Therefore, the marginal effects of the higher-order or interaction variables ignore the polynomial nature of these variables.

Inverse Mills Ratio, Expected and Conditionally Expected Values

Expected and conditionally expected values are computed only for continuous variables. The inverse Mills ratio is computed for censored or truncated continuous, binary discrete, and selection endogenous variables.

Let L_i and R_i be the lower boundary and the upper boundary of y_i . Define $a_i = \frac{L_i - \mathbf{x}'_i \boldsymbol{\beta}}{\sigma_i}$ and $b_i = \frac{R_i - \mathbf{x}'_i \boldsymbol{\beta}}{\sigma_i}$. Then the inverse Mills ratio is defined as

$$\lambda = \frac{(\phi(a_i) - \phi(b_i))}{(\Phi(b_i) - \Phi(a_i))}$$

for a continuous variable and defined as

$$\lambda = \frac{\phi(\mathbf{x}'_i \boldsymbol{\beta})}{\Phi(\mathbf{x}'_i \boldsymbol{\beta})}$$

for a binary discrete variable.

The expected value is the unconditional expectation of the dependent variable. For a censored variable, it is

$$E[y_i] = \Phi(a_i)L_i + (\mathbf{x}'_i \boldsymbol{\beta} + \lambda \sigma_i)(\Phi(b_i) - \Phi(a_i)) + (1 - \Phi(b_i))R_i$$

For a left-censored variable ($R_i = \infty$), this formula is

$$E[y_i] = \Phi(a_i)L_i + (\mathbf{x}'_i \boldsymbol{\beta} + \lambda \sigma_i)(1 - \Phi(a_i))$$

where $\lambda = \frac{\phi(a_i)}{1 - \Phi(a_i)}$.

For a right-censored variable ($L_i = -\infty$), this formula is

$$E[y_i] = (\mathbf{x}_i' \boldsymbol{\beta} + \lambda \sigma_i) \Phi(b_i) + (1 - \Phi(b_i)) R_i$$

where $\lambda = -\frac{\phi(b_i)}{\Phi(b_i)}$.

For a noncensored variable, this formula is

$$E[y_i] = \mathbf{x}_i' \boldsymbol{\beta}$$

The conditional expected value is the expectation given that the variable is inside the boundaries:

$$E[y_i | L_i < y_i < R_i] = \mathbf{x}_i' \boldsymbol{\beta} + \lambda \sigma_i$$

Probability

Probability applies only to discrete responses. It is the marginal probability that the discrete response is taking the value of the observation. If you specify the **PROBALL** option, then the probability for all possible responses of the discrete variables is computed.

Technical Efficiency

Technical efficiency for each producer is computed only for stochastic frontier models.

In general, the stochastic production frontier can be written as

$$y_i = f(x_i; \beta) \exp\{v_i\} TE_i$$

where y_i denotes producer i 's actual output, $f(\cdot)$ is the deterministic part of the production frontier, $\exp\{v_i\}$ is a producer-specific error term, and TE_i is the technical efficiency coefficient, which can be written as

$$TE_i = \frac{y_i}{f(x_i; \beta) \exp\{v_i\}}$$

For a Cobb-Douglas production function, $TE_i = \exp\{-u_i\}$. For more information, see the section “**Stochastic Frontier Production and Cost Models**” on page 461.

The cost frontier can be written in general as

$$E_i = c(y_i, w_i; \beta) \exp\{v_i\} / CE_i$$

where w_i denotes producer i 's input prices, $c(\cdot)$ is the deterministic part of the cost frontier, $\exp\{v_i\}$ is a producer-specific error term, and CE_i is the cost efficiency coefficient, which can be written as

$$CE_i = \frac{c(x_i, w_i; \beta) \exp\{v_i\}}{E_i}$$

For a Cobb-Douglas cost function, $CE_i = \exp\{-u_i\}$. For more information, see the section “**Stochastic Frontier Production and Cost Models**” on page 461. Hence, both technical and cost efficiency coefficients are the same. The estimates of technical efficiency are provided in the following subsections.

Normal-Half-Normal Model

Define $\mu_* = -\epsilon\sigma_u^2/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as shown by Jondrow et al. (1982), conditional density is as follows:

$$f(u|\epsilon) = \frac{f(u, \epsilon)}{f(\epsilon)} = \frac{1}{\sqrt{2\pi}\sigma_*} \exp\left\{-\frac{(u - \mu_*)^2}{2\sigma_*^2}\right\} \bigg/ \left[1 - \Phi\left(-\frac{\mu_*}{\sigma_*}\right)\right]$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\mu_*, \sigma_*^2)$.

From this result, it follows that the estimate of technical efficiency (Battese and Coelli 1988) is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_* - \mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)}\right] \exp\left\{-\mu_{*i} + \frac{1}{2}\sigma_*^2\right\}$$

The second version of the estimate (Jondrow et al. 1982) is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \mu_{*i} + \sigma_* \left[\frac{\phi(-\mu_{*i}/\sigma_*)}{1 - \Phi(-\mu_{*i}/\sigma_*)} \right] = \sigma_* \left[\frac{\phi(\epsilon_i \lambda / \sigma)}{1 - \Phi(\epsilon_i \lambda / \sigma)} - \left(\frac{\epsilon_i \lambda}{\sigma} \right) \right]$$

Normal-Exponential Model

Define $A = -\tilde{\mu}/\sigma_v$ and $\tilde{\mu} = -\epsilon - \sigma_v^2/\sigma_u$. Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_v\Phi(-\tilde{\mu}/\sigma_v)} \exp\left\{-\frac{(u - \tilde{\mu})^2}{2\sigma_v^2}\right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_v^2)$.

From this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \left[\frac{1 - \Phi(\sigma_v - \tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)}\right] \exp\left\{-\tilde{\mu}_i + \frac{1}{2}\sigma_v^2\right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_v \left[\frac{\phi(-\tilde{\mu}_i/\sigma_v)}{1 - \Phi(-\tilde{\mu}_i/\sigma_v)} \right] = \sigma_v \left[\frac{\phi(A)}{\Phi(-A)} - A \right]$$

Normal-Truncated Normal Model

Define $\tilde{\mu} = (-\sigma_u^2\epsilon_i + \mu\sigma_v^2)/\sigma^2$ and $\sigma_*^2 = \sigma_u^2\sigma_v^2/\sigma^2$. Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi}\sigma_*[1 - \Phi(-\tilde{\mu}/\sigma_*)]} \exp\left\{-\frac{(u - \tilde{\mu})^2}{2\sigma_*^2}\right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\tilde{\mu}, \sigma_*^2)$.

From this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \frac{1 - \Phi(\sigma_* - \tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \exp\left\{-\tilde{\mu}_i + \frac{1}{2}\sigma_*^2\right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \tilde{\mu}_i + \sigma_* \left[\frac{\phi(\tilde{\mu}_i/\sigma_*)}{1 - \Phi(-\tilde{\mu}_i/\sigma_*)} \right]$$

Naming

Naming of Parameters

When the estimation contains only one equation, the parameters are named in the same way as in other SAS procedures, such as the REG and PROBIT procedures. The constant in the regression equation is named Intercept. The coefficients of independent variables are named by the independent variables. The standard deviation of the errors is called _Sigma. The limits of the discrete dependent variable are named _Limit*i*. _Limit1 is set to 0 and the limit parameters start from $i = 2$. If you also specify the HETERO statement, the coefficients of the independent variables in the HETERO statement are called _H.x, where x is the name of the independent variable.

When the estimation contains multiple equations, the parameters in the main equation are named in the format of y.x, where y is the name of the dependent variable and x is the name of the independent variable. The standard deviation of the errors is named _Sigma.y. The correlation of the errors is named _Rho for the bivariate model. For the model that contains three variables, it is _Rho.y1.y2, _Rho.y1.y3, and _Rho.y2.y3. The construction of correlation names for models that contain four or more variables is analogous.

Naming of Output Variables

Table 10.4 shows the *options* in the OUTPUT statement, along with the corresponding variable names and their explanations.

Table 10.4 OUTPUT Statement Options

<i>output-option</i>	Variable Name	Explanation
CONDITIONAL	CEXPCT_y	Conditional expected value of y , conditioned on the truncation
ERRSTD	ERRSTD_y	Standard deviation of error term
EXPECTED	EXPCT_y	Unconditional expected value of y
MARGINAL	MEFF_x	Marginal effect of x on y ($\frac{\partial y}{\partial x}$) with single equation
	MEFF_y_x	Marginal effect of x on y ($\frac{\partial y}{\partial x}$) with multiple equations

Table 10.4 *continued*

<i>output-option</i>	Variable Name	Explanation
	MEFF_Pi_x	Marginal effect of x on y ($\frac{\partial \text{Prob}(y=i)}{\partial x}$) with single equation and discrete y
	MEFF_Pi_y_x	Marginal effect of x on y ($\frac{\partial \text{Prob}(y=i)}{\partial x}$) with multiple equations and discrete y
PREDICTED	P_y	Predicted value of y
RESIDUAL	RESID_y	Residual of y , ($y - \text{PredictedY}$)
PROB	PROB_y	Probability that y is taking the observed value in this observation (discrete y only)
PROBALL	PROB_i_y	Probability that y is taking the i th value (discrete y only)
MILLS	MILLS_y	Inverse Mills ratio for y
TE1	TE1	Technical efficiency estimate for each producer proposed by Battese and Coelli (1988)
TE2	TE2	Technical efficiency estimate for each producer proposed by Jondrow et al. (1982)
XBETA	XBETA_y	Structure part ($\mathbf{x}'\boldsymbol{\beta}$) of y equation

ODS Table Names

PROC CQLIM assigns a name to each table that it creates. You can use these names to refer to the tables when you use the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in Table 10.5.

Table 10.5 ODS Tables Produced by PROC CQLIM

Table Name	Description	Statement	Option
ClassInfo	Level information for classification variables	CLASS	Default
Correlation	Correlation of parameter estimates	PROC CQLIM	CORRB
Covariance	Covariance of parameter estimates	PROC CQLIM	COVB
DiscreteRespProfile	Response profile	ENDOGENOUS or MODEL	DISCRETE
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
GoodnessOfFit	Pseudo-R-square measures	MODEL	Default
InitialParameterEstimates	Optimization start	PROC CQLIM	ITPRINT
IterationHistory	Iteration history of nonlinear estimation	PROC CQLIM	ITPRINT
OptimizationResults	Optimization results	PROC CQLIM	ITPRINT
ParameterEstimates	Parameter estimates	MODEL	Default
SummaryContResponse	Summary of continuous response	MODEL	Default
TestResults	Hypothesis test statistics	TEST	Default

Example: The CQLIM Procedure

Example 10.1: Model with Censoring

This example uses the CQLIM procedure to process a large data table in the distributed computing environment.

The following DATA step generates a data table that contains 5 million observations from a censored model. The model contains seven variables.

```
data simulate;
  call streaminit(12345);
  array vars x1-x7;
  array parms{7} (3 4 2 4 -3 -5 -3);

  intercept=2;

  do i=1 to 5000000;
    sum_xb=0;
    do j=1 to 7;
      vars[j]=rand('NORMAL',0,1);
      sum_xb=sum_xb+parms[j]*vars[j];
    end;
    y=intercept+sum_xb+100*rand('NORMAL',0,1);
    if y>400 then y=400;
    if y<0 then y=0;
    output;
  end;
  keep y x1-x7;
run;

data mycas.simulate(copies=1 promote=yes);
  set simulate;
run;
```

The following statements estimate a censored model. To run these statements successfully, you need to promote the data table to your CAS engine libref. The DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc cqlim data=mycas.simulate ;
  model y=x1-x7 /censored(lb=0 ub=400);
run;
```

Output 10.1.1 shows the estimation results for the censored model. The “Model Fit Summary” table shows detailed information about the model. All parameter estimates in the “Parameter Estimates” table are highly significant and correspond to their theoretical values that were set during the data generating process.

Output 10.1.1 Censored Model: Summary**The CQLIM Procedure**

Summary Statistics of Continuous Responses							
Variable	Mean	Standard Error	Type	Lower Bound	Upper Bound	N Obs Lower Bound	N Obs Upper Bound
y	41.11168	59.338269	Censored	0	400	246E4	189

Model Fit Summary	
Dependent Variable	y
Number of Observations	5000000
Data Set	SIMULATE
Log Likelihood	-1.702E7
Maximum Absolute Gradient	1.806E-9
Number of Iterations	7
Optimization Method	Newton-Raphson
AIC	34043697
SBC	34043817
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-11) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	2.045243	0.054689	37.40	<.0001
x1	1	3.021825	0.049348	61.23	<.0001
x2	1	4.006987	0.049345	81.20	<.0001
x3	1	1.955760	0.049327	39.65	<.0001
x4	1	4.045403	0.049338	81.99	<.0001
x5	1	-3.020339	0.049310	-61.25	<.0001
x6	1	-5.041650	0.049351	-102.16	<.0001
x7	1	-2.950510	0.049362	-59.77	<.0001
_Sigma	1	100.015057	0.049335	2027.25	<.0001

Example 10.2: Bivariate Probit Analysis

This example shows how to estimate a bivariate probit model by using a large data table in the distributed computing environment.

The following DATA step generates a data table that contains 1 million observations from a bivariate probit model:

```
data bivariate_probit;
keep y1 y2 x1 x2;
do i = 1 to 1000000;
  x1 = rannor( 19283 );
  x2 = rannor( 19283 );
  u1 = rannor( 19283 );
  u2 = rannor( 19283 );
  y1l = 1 + 2 * x1 + 3 * x2 + u1;
  y2l = 3 + 4 * x1 - 2 * x2 + u1*.2 + u2;
  if ( y1l > 0 ) then y1 = 1;
  else y1 = 0;
```

```

        if ( y21 > 0 ) then y2 = 1;
        else                y2 = 0;
        output;
    end;
run;

data mycas.bivariate_probit(copies=1 promote=yes);
    set bivariate_probit;
run;

```

The following statements estimate a bivariate probit model. To run these statements successfully, you need to promote the data table to your CAS engine libref. The DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

Note that the INIT statement in the SAS program sets the initial values for some parameters in the optimization. The METHOD=QN option requests the quasi-Newton optimization algorithm.

```

proc cqlim data=mycas.bivariate_probit method=qn;
    init y1.x1=2.8, y1.x2=2.1, _rho=.1;
    model y1 = x1 x2 / discrete;
    model y2 = x1 x2 / discrete;
run;

```

Output 10.2.1 shows the estimation results for the bivariate probit model. The “Discrete Response Profile” tables show the response profile for the variables y_1 and y_2 .

Output 10.2.1 Bivariate Probit Analysis Results

The CQLIM Procedure

Discrete Response Profile of y1		
Index	Value	Total Frequency
1	0	394464
2	1	605536

The CQLIM Procedure

Discrete Response Profile of y2		
Index	Value	Total Frequency
1	0	256423
2	1	743577

Output 10.2.1 *continued*

Model Fit Summary	
Dependent Variable	y1 y2
Number of Observations	1000000
Data Set	BIVARIATE_PROBIT
Log Likelihood	-314562
Maximum Absolute Gradient	0.000497
Number of Iterations	18
Optimization Method	Quasi-Newton
AIC	629137.1
SBC	629219.8
Covariance Estimation	Hessian

Convergence criterion (FCONV=1E-11) satisfied.

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
y1.Intercept	1	0.996487	0.003065	325.15	<.0001
y1.x1	1	1.994780	0.004784	416.97	<.0001
y1.x2	1	2.992850	0.006689	447.43	<.0001
y2.Intercept	1	2.931870	0.007657	382.88	<.0001
y2.x1	1	3.904453	0.010185	383.37	<.0001
y2.x2	1	-1.954357	0.005639	-346.58	<.0001
_Rho	1	0.193618	0.006250	30.98	<.0001

References

- Aigner, C., Lovell, C. A. K., and Schmidt, P. (1977). "Formulation and Estimation of Stochastic Frontier Production Function Models." *Journal of Econometrics* 6:21–37.
- Battese, G. E., and Coelli, T. J. (1988). "Prediction of Firm-Level Technical Efficiencies with a Generalized Frontier Production Function and Panel Data." *Journal of Econometrics* 38:387–399.
- Christensen, L. R., and Greene, W. H. (1976). "Economies of Scale in U.S. Electric Power Generation." *Journal of Political Economy* 84:655–676.
- Coelli, T. J., Prasada Rao, D. S., and Battese, G. E. (1998). *An Introduction to Efficiency and Productivity Analysis*. London: Kluwer Academic.
- Genz, A. (1992). "Numerical Computation of Multivariate Normal Probabilities." *Journal of Computational and Graphical Statistics* 1:141–150.
- Hajivassiliou, V. A., and McFadden, D. L. (1998). "The Method of Simulated Scores for the Estimation of LDV Models." *Econometrica* 66:863–896.
- Jondrow, J., Lovell, C. A. K., Materov, I. S., and Schmidt, P. (1982). "On the Estimation of Technical Efficiency in the Stochastic Frontier Production Function Model." *Journal of Econometrics* 19:233–238.

- Kumbhakar, S. C., and Lovell, C. A. K. (2000). *Stochastic Frontier Analysis*. New York: Cambridge University Press.
- McKelvey, R. D., and Zavoina, W. (1975). “A Statistical Model for the Analysis of Ordinal Level Dependent Variables.” *Journal of Mathematical Sociology* 4:103–120.
- Meeusen, W., and van den Broeck, J. (1977). “Efficiency Estimation from Cobb-Douglas Production Functions with Composed Error.” *International Economic Review* 18:435–444.
- Mroz, T. A. (1987). “The Sensitivity of an Empirical Model of Married Women’s Work to Economic and Statistical Assumptions.” *Econometrica* 55:765–799.
- Wooldridge, J. M. (2002). *Econometric Analysis of Cross Section and Panel Data*. Cambridge, MA: MIT Press.

Chapter 11

The CSPATIALREG Procedure

Contents

Overview: CSPATIALREG Procedure	478
PROC CSPATIALREG Features	479
PROC CSPATIALREG Compared with the SPATIALREG Procedure	480
Using CAS Sessions and CAS Engine Librefs	480
Getting Started: CSPATIALREG Procedure	481
Syntax: CSPATIALREG Procedure	485
Functional Summary	485
PROC CSPATIALREG Statement	487
BOUNDS Statement	491
BY Statement	492
CLASS Statement	492
DISPLAY Statement	493
DISPLAYOUT Statement	494
INIT Statement	495
MODEL Statement	495
OUTPUT Statement	496
RESTRICT Statement	497
SPATIALEFFECTS Statement	498
SPATIALID Statement	498
TEST Statement	499
Details: CSPATIALREG Procedure	500
Missing Values	500
Spatial Autoregressive Models	500
Spatial Durbin Models	501
Spatial Error Models	502
Spatial Durbin Error Models	503
Spatial Moving Average Models	504
Spatial Durbin Moving Average Models	504
Spatial Autoregressive Moving Average Models	505
Spatial Durbin Autoregressive Moving Average Models	506
Spatial Autoregressive Confused Models	507
Spatial Durbin Autoregressive Confused Models	507
Linear Regression Models	508
Spatial Lag of X Models	509
Specifying the Spatial Weights Matrix	510
Compact Representation of the Spatial Weights Matrix	511

Spatial ID Matching	513
Parameter Space of Autoregressive Parameters	514
Approximations to the Jacobian	515
Impact Estimation	516
Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements	518
Computational Resources	521
Covariance Matrix Types	522
Displayed Output	522
OUTPUT OUT= Data Table	523
ODS Table Names	524
Examples: CSPATIALREG Procedure	524
Example 11.1: Columbus Crime Data	524
Example 11.2: Simulated Data Example	533
Example 11.3: Compact Representation of a Spatial Weights Matrix	536
Example 11.4: Taylor and Chebyshev Approximations	538
Example 11.5: Impact Estimation	545
References	548

Overview: CSPATIALREG Procedure

The CSPATIALREG (spatial regression) procedure, in SAS Viya, analyzes a class of linear spatial econometric models for cross-sectional data whose observations are spatially referenced or georeferenced. For example, housing price data that are collected from the 48 contiguous US states fall into the category of spatially referenced data. Compared to nonspatial regression models, spatial econometric models are capable of handling spatial interaction and spatial heterogeneity in a regression setting (Anselin 2001).

Spatial econometric models have been widely used in economics, political science, sociology, and other fields. For example, LeSage and Pace (2009) provide a detailed introduction to commonly used spatial econometric models from both frequentist and Bayesian perspectives. A brief introduction to spatial econometric models is also provided by Elhorst (2013).

PROC CSPATIALREG Features

The CSPATIALREG procedure fits spatial data regression models by maximum likelihood techniques.

PROC CSPATIALREG supports the following models:

- linear model
- linear model with spatial lag of X (SLX) effects
- spatial autoregressive (SAR) model
- spatial Durbin model (SDM)
- spatial error model (SEM)
- spatial Durbin error model (SDEM)
- spatial moving average (SMA) model
- spatial Durbin moving average (SDMA) model
- spatial autoregressive moving average (SARMA) model
- spatial Durbin autoregressive moving average (SDARMA) model
- spatial autoregressive confused (SAC) model
- spatial Durbin autoregressive confused (SDAC) model

The following list summarizes some basic features of the CSPATIALREG procedure:

- uses model-building syntax that includes CLASS and effect-based MODEL statements that are familiar from other SAS Econometrics analytic procedures
- enables you to fit multiple models by using maximum likelihood estimation
- provides a [TEST](#) statement for performing linear tests of hypotheses
- provides a [RESTRICT](#) statement for imposing linear parameter constraints
- provides an [OUTPUT](#) statement for producing a data table that contains predictions and residuals

Because the CSPATIALREG procedure runs on SAS Cloud Analytic Services (CAS), it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode
- exploits all the available cores and concurrent threads. For information about how PROC CSPATIALREG uses threads, see the section “[Multithreading](#)” on page 65 in Chapter 3, “[Shared Concepts](#).”

PROC CSPATIALREG Compared with the SPATIALREG Procedure

The CSPATIALREG procedure is specifically designed to operate on SAS Viya and performs computations in multiple threads on multiple machines. In comparison, the SPATIALREG procedure executes in multiple threads on a single machine. The CSPATIALREG procedure models cross-sectional spatial data in a way that is similar to how the SPATIALREG procedure in SAS/ETS software models them.

This release of the CSPATIALREG procedure supports all the models that are supported by PROC SPATIALREG. In addition, the CSPATIALREG procedure supports impact estimation for the following models: linear, SLX, SAR, SDM, SEM, SDEM, SMA, SDMA, SARMA, SDARMA, SAC, and SDAC models. The SPATIALREG procedure does not provide this support.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: CSPATIALREG Procedure

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 12 and “Loading a SAS Data Set onto a CAS Server” on page 13 in Chapter 3, “Shared Concepts.”

The CSPATIALREG procedure is similar to other SAS regression model procedures for nonspatial data, except that in PROC CSPATIALREG you usually need to provide a secondary data table (in the WMAT= option), which provides information about spatial weights matrix. The spatial weights matrix defines all pairwise spatial relationships and is a vital component of a spatial econometric model. For more information about how to create spatial weights matrix, see the section “Specifying the Spatial Weights Matrix” on page 510.

The following statements fit a SAR model:

```
proc cspatialreg data=mycas.One Wmat=mycas.W;
  model y = x1 x2 / type=SAR;
  spatialid sid;
run;
```

The response variable *y* (a continuous variable) and the explanatory variables *x1* and *x2* are contained in the data table *mycas.One*. The spatial weights matrix that you specify using the WMAT= option defines neighbor relationships among all spatial units in the data. The SPATIALID statement specifies a spatial ID variable that identifies spatial units in two data tables that you provide in the DATA= and WMAT= options. You specify the TYPE=SAR option in the MODEL statement to request a SAR model.

The following example illustrates PROC CSPATIALREG by using a real-world data set. The data set *CrimeOH* is taken from Anselin (1988). This data set contains variables such as *Income* (household income, measured in \$1,000), *HValue* (housing value by \$1,000), and *Crime* (number of crimes, including residential burglaries and vehicle thefts, measured per 1,000 households) in 49 neighborhoods in Columbus, Ohio. You want to examine how household income and housing value affect the number of crimes in these neighborhoods.

The first 10 observations in the *mycas.CrimeOH* data set are shown in Figure 11.1.

Figure 11.1 Columbus Crime Data

Obs	crime	income	hvalue	lat	lon	sid
1	18.802	21.232	44.567	35.62	42.38	s1
2	0.178	8.438	75.000	33.36	38.41	s4
3	50.732	11.252	23.225	40.01	38.00	s7
4	34.001	13.598	96.400	47.61	36.42	s10
5	19.146	18.942	40.300	50.11	29.91	s13
6	16.241	29.833	61.950	48.44	27.93	s16
7	33.705	11.709	30.450	43.37	33.46	s19
8	41.968	9.918	23.600	44.10	30.40	s22
9	25.962	16.961	33.500	43.23	27.31	s25
10	29.028	14.135	27.733	39.32	25.85	s28

The following SAS statements fit a linear regression model to the CrimeOH data set:

```
/*-- Linear Model --*/
proc cspatialreg data=mycas.CrimeOH;
    model crime = income hvalue / type=LINEAR;
run;
```

The “Model Fit Summary” table, shown in Figure 11.2, lists several fit summary statistics for the model. By default, the CSPATIALREG procedure uses the Newton-Raphson optimization technique. The maximum log-likelihood value is shown, in addition to two information measures, Akaike’s information criterion (AIC) and Schwarz’s Bayesian information criterion (SBC). AIC or SBC can be used for model selection. For a set of candidate models, the model that has the smallest AIC or SBC is often preferred. In addition, the method that is used for covariance estimation is shown.

Figure 11.2 Fit Summary Statistics for a Linear Model
The CSPATIALREG Procedure

Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Model	LINEAR
Log Likelihood	-187.377
Maximum Absolute Gradient	7.599E-7
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	382.7542
SBC	390.3215
Covariance Estimation	Hessian

The parameter estimates of the model and their standard errors are shown in Figure 11.3. Based on the p -values, both Income and HValue are significant at the 0.05 level.

Figure 11.3 Parameter Estimates of the Linear Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	68.618863	4.588210	14.96	<.0001
income	1	-1.597304	0.323739	-4.93	<.0001
hvalue	1	-0.273931	0.099989	-2.74	0.0062
_sigma2	1	122.751696	24.799493	4.95	<.0001

The following statements fit a SAR model to the CrimeOH data set:

```
/*-- SAR Model --*/
proc cspatialreg data=mycas.CrimeOH wmat=mycas.CrimeWmat NONORMALIZE;
    model crime = income hvalue / type=SAR;
    spatialid sid;
run;
```

The NONNORMALIZE option requests that the spatial weights matrix that is specified in the mycas.CrimeWmat data table be used “as is” rather than be row-standardized. The “Model Fit Summary” table, shown in Figure 11.4, lists several fit summary statistics for the SAR model. For this model, the value of AIC is about 374.78—smaller than 382.75, which is the AIC value for the preceding linear model. Based on AIC, the SAR model is preferred.

Figure 11.4 Fit Summary Statistics for the SAR Model

The CSPATIALREG Procedure

Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Spatial Weights	CRIMEWMAT
Model	SAR
Spatial ID Variable	sid
Log Likelihood	-182.389
Maximum Absolute Gradient	2.787E-7
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	374.7772
SBC	384.2363
Covariance Estimation	Hessian

The parameter estimates of the SAR model and their standard errors are shown in Figure 11.5. According to the p -values, both Income and HValue are significant at the 0.05 level. In addition, the spatial autoregressive coefficient ρ is estimated to be about 0.431, with a p -value of 0.0005.

Figure 11.5 Parameter Estimates of the SAR Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	45.077070	7.870590	5.73	<.0001
income	1	-1.031531	0.328403	-3.14	0.0017
hvalue	1	-0.265924	0.088218	-3.01	0.0026
_rho	1	0.431020	0.123594	3.49	0.0005
_sigma2	1	95.487066	19.506312	4.90	<.0001

Unlike the previous SAR model, a spatial Durbin model (SDM) accounts for exogenous interaction effects by introducing additional spatially lagged regressors into the model. To fit an SDM, you need to combine the MODEL statement (with the TYPE=SAR option) and the SPATIALEFFECTS statement. The following statements fit an SDM :

```

/*-- SDM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
    model crime = income hvalue / type=SAR;
    spatialeffects income hvalue;
    spatialid sid;
run;

```

The fit summary statistics for the SDM are shown in Figure 11.6. Parameter estimates are provided in Figure 11.7.

Figure 11.6 Fit Summary Statistics for the SDM

The CSPATIALREG Procedure

Model Fit Summary	
Dependent Variable	crime
Number of Observations	49
Data Set	CRIMEOH
Spatial Weights	CRIMEWMAT
Model	SDM
Spatial ID Variable	sid
Log Likelihood	-181.391
Maximum Absolute Gradient	5.448E-8
Number of Iterations	16
Optimization Method	Newton-Raphson
AIC	376.7828
SBC	390.0256
Covariance Estimation	Hessian

Figure 11.7 Parameter Estimates for the SDM

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	42.803457	13.924487	3.07	0.0021
income	1	-0.914206	0.336439	-2.72	0.0066
hvalue	1	-0.293745	0.088857	-3.31	0.0009
W_income	1	-0.519640	0.594772	-0.87	0.3823
W_hvalue	1	0.245716	0.176854	1.39	0.1647
_rho	1	0.426492	0.167492	2.55	0.0109
_sigma2	1	91.779519	18.909222	4.85	<.0001

The spatial autoregressive coefficient ρ is estimated to be 0.426 with a p -value of 0.0109 based on an asymptotic t test. This result seems to suggest that there is a significantly positive spatial dependence in the number of crimes.

In the CSPATIALREG procedure, the null hypothesis $H_0: \rho = 0$ can also be tested against the alternative $H_a: \rho \neq 0$ by using the likelihood ratio (LR) test, Lagrange multiplier (LM) test, and Wald test. For the LR test, the test statistic is equal to $-2(\mathcal{L}_{\text{linear}} - \mathcal{L}_{\text{SAR}}) = -2(-187.38 + 182.39) = 9.98$, where $\mathcal{L}_{\text{linear}}$ and $\mathcal{L}_{\text{SAR}}$ are the log likelihoods for the linear regression model and SAR model, respectively. The likelihood ratio test is significant at the 0.05 level, providing strong evidence of spatial dependence in the data.

Syntax: CSPATIALREG Procedure

The following statements are available in the CSPATIALREG procedure. Items within angle brackets (< >) or square brackets ([]) are optional.

```

PROC CSPATIALREG < options > ;
  BOUNDS bound1 [ , bound2 ... ] ;
  BY variables ;
  CLASS variable < (options) > ... < variable < (options) > > < / global-options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  INIT initialization1 < , initialization2 ... > ;
  MODEL dependent-variable = regressors < / options > ;
  OUTPUT OUT=CAS-libref.data-table < options > < keyword < =name > > ... < keyword < =name > > ;
  RESTRICT restriction1 [ , restriction2 ... ] ;
  TEST equation1 < , equation2 ... > < / test-options > ;
  SPATIALEFFECTS < model-spatial-effect-regressors > ;
  SPATIALID variable ;

```

You can specify only one MODEL statement. If you specify a BOUNDS, INIT, OUTPUT, RESTRICT, SPATIALEFFECTS, or TEST statement, it must appear after the MODEL statement. You can specify no more than one SPATIALEFFECTS statement. However, multiple BOUNDS, INIT, RESTRICT, and TEST statements are allowed.

Functional Summary

Table 11.1 summarizes the statements and options available in the CSPATIALREG procedure. The sections that follow provide detailed syntax information about each of these statements, beginning with the PROC CSPATIALREG statement. The remaining statements are covered in alphabetical order.

Table 11.1 Functional Summary

Description	Statement	Option
Data Table Options		
Specifies the CAS input data table	PROC CSPATIALREG	DATA=
Specifies the CAS spatial weights input data table	PROC CSPATIALREG	WMAT=
Suppresses normalization of the spatial weights	PROC CSPATIALREG	NONORMALIZE
Writes estimates of $\mathbf{x}'_i\boldsymbol{\beta}$, predicted values, and residuals to an output data table	OUTPUT	OUT=
Approximation Control Options		
Specifies the approximation-related options	PROC CSPATIALREG	APPROXIMATION=
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies classification variables	CLASS	

Table 11.1 *continued*

Description	Statement	Option
Specifies a spatial ID variable	SPATIALID	
Impact Estimation Control Option		
Specifies options that are related to impact estimation	PROC CSPATIALREG	IMPACT
Declaring the Role of Variables		
Specifies BY-group processing	BY	
Specifies classification variables	CLASS	
Specifies a spatial ID variable	SPATIALID	
Printing Control Options		
Prints the correlation matrix of the estimates	MODEL	CORRB
Prints the covariance matrix of the estimates	MODEL	COVB
Prints a summary iteration listing	MODEL	ITPRINT
Suppresses the normal printed output	PROC CSPATIALREG	NOPRINT
Prints all available output	MODEL	PRINTALL
Prints timing information	PROC CSPATIALREG	PRINTTIMING
Prints the names used internally for the parameters	PROC CSPATIALREG	PRINTINTERNALNAMES
Options to Control the Optimization Process		
Specifies the maximum number of iterations	MODEL	MAXITER=
Selects the iterative minimization method to use	PROC CSPATIALREG	METHOD=
Specifies the maximum number of iterations	PROC CSPATIALREG	MAXITER=
Specifies the maximum number of function calls	PROC CSPATIALREG	MAXFUNC=
Specifies the upper limit of CPU time in seconds	PROC CSPATIALREG	MAXTIME=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values of parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the spatial lag of covariate effect	SPATIALEFFECTS	
Specifies the type of model	MODEL	TYPE=
	PROC CSPATIALREG	TYPE=
Specifies the type of covariance matrix	MODEL	COVEST=
Suppresses the intercept parameter	MODEL	NOINT
Output Control Options		
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	
Outputs SAS variables to the output data table	OUTPUT	COPYVAR=
Outputs the residual	OUTPUT	RESID=

Table 11.1 *continued*

Description	Statement	Option
Outputs the expected value of the response variable	OUTPUT	PRED=
Outputs estimates of $\mathbf{x}_i' \boldsymbol{\beta}$	OUTPUT	XBETA=

PROC CSPATIALREG Statement

PROC CSPATIALREG *options* ;

You can specify the following *options* in the PROC CSPATIALREG statement.

Data Table Options

You must specify the following *option*:

DATA=*CAS-libref.data-table*

names the input data table for PROC CSPATIALREG to use. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 480.

data-table specifies the name of the input data table.

This is the primary input data table, which contains dependent variables, explanatory variables, and so on.

For all models except a purely linear model, you must also specify the following *option*:

WMAT=*CAS-libref.data-table*

specifies the secondary spatial weights data table, which you can use to construct the spatial weights matrix $\mathbf{W}$. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 480.

Loosely speaking, the entries of $\mathbf{W}$, $w(s_i, s_j)$, define the amount of influence that a unit s_j has over a unit s_i . The entries $w(s_i, s_j)$ must be nonnegative and have zeros on the diagonal; that is, $w(s_i, s_j) \geq 0$ and $w(s_i, s_i) = 0$, where i and $j = 1, 2, \dots, n$ and n is the total number of spatial units in the data. Any nonzero diagonal elements $w(s_i, s_i)$ are replaced with 0. The spatial weights matrix can be asymmetric; that is, it is not necessary that $w(s_i, s_j) = w(s_j, s_i)$. For information about missing spatial weights in $\mathbf{W}$, see the NONNORMALIZE option.

The $\mathbf{W}$ matrix can take two different forms:

- You can provide a full spatial weights matrix. In this case, the data table that you specify in the WMAT= option has n rows and $n + 1$ columns and must have a column for spatial ID variable.
- You can specify the spatial weights matrix by using a compact form when appropriate. In this form, the number of observations in the data table that you specify in the WMAT= option should match the number of nonzero elements in the spatial weights matrix. Moreover, the number of columns in this data table should be three. The first two columns contain the row and column indices for nonzero entries in the spatial weights matrix, and the third column contains the nonzero entries in the spatial weights matrix. If you use the compact form for the spatial weights matrix, you must include a SPATIALID statement to match observations in the data tables that you specify in the DATA= and WMAT= options. For more information about the SPATIALID statement, see the section “[SPATIALID Statement](#)” on page 498. For more information about the compact representation of the spatial weights matrix, see the section “[Compact Representation of the Spatial Weights Matrix](#)” on page 511.

For all models except a purely linear model, you can also specify the following *option*:

NONORMALIZE

suppresses the row standardization of the spatial weights matrix that you specify in the WMAT= option. By default, the spatial weights matrix is row-standardized; that is, the spatial weights matrix has unit row sum. If you specify the NONORMALIZE option, spatial weights are used “as is” except for $w(s_i, s_i)$, which is always treated as 0. This implies that an entry $w(s_i, s_j)$ in the **W** matrix cannot be missing for $i \neq j$ if you specify this option. If you do not specify this option, missing spatial weights are replaced with zeros.

Approximation Control Option

For the SAR, SDM, SEM, and SDEM models, you can specify the following *options*:

APPROXIMATION=(*approx-options*)

specifies options that are related to approximating the Jacobian, as described in the section “[Approximations to the Jacobian](#)” on page 515. To invoke approximation, you must specify one or more of the following *approx-options*:

CHEBYSHEV | TAYLOR

specifies the approximation method. By default, Chebyshev approximation is used. The Taylor approximation is used only if you specify the TAYLOR option.

NMC=number

specifies a positive integer as the number of standard random normal draws for Monte Carlo simulation. By default, NMC=100.

ORDER=number

specifies a positive integer as the order of series in Taylor or Chebyshev approximation. If you specify Taylor approximation, ORDER=50 by default. If you specify Chebyshev approximation, ORDER=5 by default.

SEED=number

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator that is used for Monte Carlo simulation. Specifying a seed enables you to reproduce your analysis. By default, SEED=1.

Impact Estimation Control Option

For impact estimation, you can specify the following *options*:

IMPACT(*impactest-option*)

specifies options that are related to impact estimation, as described in the section “[Impact Estimation](#)” on page 516. To invoke impact estimation, you must specify one or more of the following *impactest-options*:

NMC=number

specifies a positive integer as the number of iterations for Monte Carlo simulation. By default, NMC=1000.

ORDER=number

specifies a positive integer as the highest order of the Neumann series for approximation. By default, ORDER=50.

SEED=number

specifies an integer seed in the range 1 to $2^{31} - 1$ for the random number generator that is used for Monte Carlo simulation. Specifying a seed enables you to reproduce your analysis. By default, SEED=1.

Printing Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

PRINTINTERNALNAMES

prints internal names that are assigned to parameters.

PRINTTIMING

prints a timing report.

Estimation Control Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement:

COVEST=HESSIAN | OP | QML

specifies the type of covariance matrix for the parameter estimates. You can specify the following types:

HESSIAN	specifies the covariance from the Hessian matrix.
OP	specifies the covariance from the outer product matrix.
QML	specifies the covariance from the outer product and Hessian matrices.

By default, COVEST=HESSIAN. The quasi-maximum-likelihood estimates are computed using COVEST=QML. For all models except the linear and SLX models, only COVEST=HESSIAN is supported.

TYPE=LINEAR | SAC | SAR | SARMA | SEM | SMA

specifies the type of model to be fitted. You can specify the following model types:

LINEAR	fits a linear model.
SAC	fits a spatial autoregressive confused model.
SAR	fits a spatial autoregressive model.
SARMA	fits a spatial autoregressive moving average model.
SEM	fits a spatial error model.
SMA	fits a spatial moving average model.

By default, TYPE=SAR.

Optimization Control Options

PROC CSPATIALREG uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options* in either the PROC CSPATIALREG statement or the MODEL statement:

MAXFUNC=*i***MAXFU=*i***

specifies the maximum number of function calls in the optimization process. By default, MAXFUNC=1000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually used can exceed the number of calls that you specify in this option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number of iterations in the optimization process. By default, MAXITER=200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation available on your computer. The time that you specify in this option is checked only once at the end of each iteration. Therefore, the actual run time can be much longer than *r*. The actual run time includes the remaining time that is needed to finish the iteration and the time that is needed to generate the output of the results.

METHOD=CONGRA | DBLDOG | NEWRAP | NONE | NRRIDG | QUANEW | TRUREG

specifies the iterative minimization method to use. You can specify the following methods:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double-dogleg method.
NEWRAP	specifies the Newton-Raphson method.
NONE	specifies that optimization not be performed.
NRRIDG	specifies the Newton-Raphson ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

By default, METHOD=NEWRAP.

BOUNDS Statement

BOUNDS *bound1* [, *bound2* ...] ;

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item [*operator item* [*operator item* ...]]

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the BOUNDS statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. For more information about the RESTRICT statement, see the section “[RESTRICT Statement](#)” on page 497.

The following BOUNDS statement constrains the estimates of the parameter for z to be negative, the parameters for x_1 through x_{10} to be between 0 and 1, and the parameter for the spatial lag of x_1 to be less than 1:

```
bounds z < 0,
       0 < x1-x10 < 1,
       W_x1 < 1;
```

BY Statement

BY *variables* ;

You can use a BY statement in PROC CSPATIALREG to obtain separate analyses of observations in groups that are defined by the BY variables. When you use a BY statement, the primary input data table (specified in the DATA= option) should be sorted by the BY variables.

BY-statement processing is not supported when the CSPATIALREG procedure runs alongside the database or alongside the Hadoop Distributed File System (HDFS). These modes are used if the input data are stored in a database or HDFS and the grid host is the appliance that houses the data.

CLASS Statement

CLASS *variable* <(options)>... <*variable* <(options)>> </global-options> ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. Table 11.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 14 in Chapter 3, “Shared Concepts.”

Table 11.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 524. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 524. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

INIT Statement

INIT *initialization1* < , *initialization2* ... > ;

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter <=> *number*

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the INIT statement, see the section “[Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 518.

MODEL Statement

MODEL *dependent-variable* = *regressors* </ *options* > ;

The MODEL statement specifies the *dependent-variable* and independent covariates (*regressors*) for the regression model. If you specify no *regressors*, PROC CSPATIALREG fits a model that contains only an intercept. The *dependent-variable* is treated as a continuous variable in the primary input data table (specified in the DATA= option). Models in PROC CSPATIALREG do not allow missing values. If there are missing values, an error message is returned.

You can specify only one MODEL statement. You can specify the following *options* after a slash (/):

NOINT

suppresses the intercept parameter.

TYPE=LINEAR | SAC | SAR | SARMA | SEM | SMA

specifies the type of model to be fitted. If you specify this option in both the MODEL statement and the PROC CSPATIALREG statement, the MODEL statement overrides the PROC CSPATIALREG statement. You can specify the following model types:

LINEAR	fits a linear model.
SAC	fits a spatial autoregressive confused model.
SAR	fits a spatial autoregressive model.
SARMA	fits a spatial autoregressive moving average model.
SEM	fits a spatial error model.
SMA	fits a spatial moving average model.

By default, TYPE=SAR.

Printing Options

You can specify the following options in either the PROC CSPATIALREG statement or the MODEL statement. If you specify these options in both the PROC CSPATIALREG and MODEL statements, the MODEL statement overrides the PROC CSPATIALREG statement.

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

NOPRINT

suppresses all printed output.

PRINTALL

prints all available output.

OUTPUT Statement

OUTPUT *OUT=CAS-libref.data-table*

< options >

< keyword <=name> >... < keyword <=name> > ;

The OUTPUT statement creates a data table that contains observationwise statistics that PROC CSPATIALREG computes after fitting the model. These observationwise statistics include the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the residual. In order to avoid data duplication for large data tables, the variables in the input data table are *not* included in the output data table unless you specify them in the **COPYVAR=** option.

You can specify only one OUTPUT statement. You must specify the following option:

OUT=CAS-libref.data-table

names the output data table for PROC CSPATIALREG to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>CAS-libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 480.
-------------------	---

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

You can also specify the following *options*:

COPYVAR=*variable*

COPYVARS=*variables*

transfers one or more variables from the primary input data table to the output data table.

PRED=*name*

MEAN=*name*

assigns a name to the variable that contains the predicted value of the response variable.

RESID=*name*

RESIDUAL=*name*

assigns a name to the variable that contains the residuals (that is, the difference between the observed and predicted values of the response variable).

XBETA=*name*

names the variable that contains estimates of $\mathbf{x}_i' \boldsymbol{\beta}$.

RESTRICT Statement

RESTRICT *restriction1* [, *restriction2* ...] ;

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written as an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

Restriction expressions can be composed of parameter names; constants; and the operators times (*), plus (+), and minus (−). The restriction expressions must be a linear function of the parameters.

Parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the RESTRICT statement, see the section [“Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements”](#) on page 518.

Lagrange multipliers are reported in the “Parameter Estimates” table for all the active linear constraints. They are identified by the names Restrict1, Restrict2, and so on. The *p*-values of these Lagrange multipliers are computed using a beta distribution (LaMotte 1994). Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

For example, the following RESTRICT statement constrains the spatial autoregressive coefficient ρ to 0, which removes endogenous interaction effects:

```
restrict _rho = 0;
```

SPATIALEFFECTS Statement

SPATIALEFFECTS < *model-spatial-effect-regressors* > < /options > ;

The SPATIALEFFECTS statement enables you to specify covariates (such as X) whose spatial lag, WX , is to be added to the MODEL statement.

PROC CSPATIALREG adds the spatially weighted *model-spatial-effect-regressors* to regressors that you specify in the MODEL statement. For example, if you specify q variables $z_1, \dots, z_q$ in the SPATIALEFFECTS statement, then each of q spatially weighted variables, as represented by each column of WZ , has a parameter to be included in the regression. Here, WZ denotes the matrix product of W and Z . In addition, Z is the design matrix that is formed by the q variables $z_1, \dots, z_q$. The spatial weights matrix W comes from the data table that you specify in the WMAT= option. The “Parameter Estimates” table in the displayed output shows the estimates for spatially weighted *model-spatial-effect-regressors*; they are labeled with the prefix “W_”. For example, if you specify z (a variable in your primary data table) as a *model-spatial-effect-regressor*, then the “Parameter Estimates” table labels the corresponding parameter estimate “W_z”.

SPATIALID Statement

SPATIALID *variable* ;

The SPATIALID statement specifies a variable that identifies a spatial unit for each observation in the two data tables that you specify in the DATA= and WMAT= options in the PROC CSPATIALREG statement. The variable that you specify in the SPATIALID statement is also used to match the rows and columns in the spatial weights matrix. Only one SPATIALID statement and one spatial ID *variable* are allowed. For all models except a purely linear regression model without an SLX effect, you need a SPATIALID statement in order to specify a spatial ID *variable* for the purpose of matching observations in two data tables that you specify in the DATA= and WMAT= options. The values of the spatial ID *variable* in either the DATA= or WMAT= data table cannot be missing.

The *variable* in the SPATIALID statement can be either numeric or character. However, the type of spatial ID *variable* in the two data tables that you specify in the DATA= and WMAT= options must be the same. When the spatial ID *variable* is numeric, it needs to be integer-valued. If you specify a number that is not an integer, PROC CSPATIALREG uses the integer part of that number for matching.

TEST Statement

`<label:> TEST <'string'> equation1 <, equation2... > /<test-options> ;`

The TEST statement performs Wald, Lagrange multiplier, and likelihood ratio tests of linear hypotheses about the parameters that you specify in the preceding MODEL statement.

You can add a label (which is printed in the output) to a TEST statement in two ways: add an unquoted *label* followed by a colon before the TEST keyword, or add a quoted *string* after the TEST keyword. The unquoted *label* cannot contain spaces. If you include both an unquoted *label* and a quoted *string*, PROC CSPATIALREG uses the unquoted *label*. If you specify neither an unquoted *label* nor a quoted *string*, PROC CSPATIALREG automatically labels the tests.

Each *equation* specifies a linear hypothesis to be tested and consists of regression parameter names and relational operators. The regression parameter names are as shown in the Parameter column of the “Parameter Estimates” table. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)’), then you must use the internal name of the parameter to refer to that parameter in the TEST statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

For more information about how parameters are named in the TEST statement, see the section “[Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements](#)” on page 518. Only linear equality restrictions and tests are permitted in PROC CSPATIALREG. Test *equations* can consist only of algebraic operations that involve the addition symbol (+), subtraction symbol (-), and multiplication symbol (*).

All hypotheses in a TEST statement are tested jointly.

You can specify the following options after the slash (/):

ALL

requests the Wald, Lagrange multiplier, and likelihood ratio tests.

LM

requests the Lagrange multiplier test.

LR

requests the likelihood ratio test.

WALD

requests the Wald test.

The following statements illustrate the use of the TEST statement:

```
proc cspatialreg data=mycas.dat;
  model y = x1 x2 x3/type=LINEAR;
  test x1 = 0, x2 * .5 + 2 * x3 = 0/ALL;
  test_int: test intercept = 0, x3 = 0/LR;
run;
```

The first test investigates the joint hypothesis that $\beta_1 = 0$ and $0.5\beta_2 + 2\beta_3 = 0$.

The TEST statement accepts labels that are reproduced in the printed output. TEST statements can be labeled in two ways: a TEST statement can be preceded by a label followed by a colon, or the keyword TEST can be followed by a quoted string. If both are present, PROC CSPATIALREG uses the label preceding the colon. If no label is specified, PROC CSPATIALREG automatically labels the tests.

Details: CSPATIALREG Procedure

Missing Values

Missing values can occur in both the primary data table (which you specify in the DATA= option) and the secondary spatial weights data table (which you specify in the WMAT= option). The CSPATIALREG procedure does not allow missing values in the primary input data table. If any observation in the primary input data table has a missing value for one or more of the regressors or the dependent variable, PROC CSPATIALREG will not fit the model. If any observation in the primary data table has a missing value for the spatial ID variable when you specify the SPATIALID statement, PROC CSPATIALREG will not fit the model. In such cases, you get an error message.

For the secondary spatial weights data table, a missing value for the spatial ID variable is not allowed when you specify the SPATIALID statement. Moreover, missing spatial weights are not allowed if you specify the NONNORMALIZE option. In these cases, PROC CSPATIALREG issues an error message and skips the model fitting.

Spatial Autoregressive Models

The spatial autoregressive (SAR) model is useful for incorporating spatial dependence in the dependent variable—that is, the endogenous interaction effect. Let y_i denote the observation that is associated with a spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let an $n \times n$ matrix $\mathbf{W}$ with nonnegative elements be a spatial weights matrix and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded for the spatial unit s_i . You can formulate the SAR model as

$$y_i = \rho \sum_{j=1}^n W_{ij} y_j + \mathbf{x}_i' \boldsymbol{\beta} + \epsilon_i$$

where $i = 1, 2, \dots, n$, ρ is the spatial autoregressive coefficient, $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector, W_{ij} is the (i, j) th element of the matrix $\mathbf{W}$ subject to $W_{ii} = 0$, and the error term ϵ_i that is related to the spatial unit s_i is assumed to follow $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$ for $i = 1, 2, \dots, n$.

The SAR model is often described in vector form as

$$\mathbf{y} = \rho \mathbf{W} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, and $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$.

The standard estimator for the SAR model is the maximum likelihood estimator (MLE). For the SAR model, the log-likelihood function is (Anselin 2001)

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^2} + \ln |\mathbf{A}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho\mathbf{W}$, $\mathbf{I}_n$ is an $n \times n$ identity matrix, and $|\mathbf{A}|$ denotes the determinant of $\mathbf{A}$.

You can derive the gradients as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{\mathbf{X}'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{1}{\sigma^2} \mathbf{y}'\mathbf{W}'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \text{tr}(\mathbf{A}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^4}$$

For the $n \times n$ matrix $\mathbf{A}$, $\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii}$, where a_{ii} is the i th diagonal element of $\mathbf{A}$.

A SAR model does not account for exogenous interaction effects. In practice, the value of the dependent variable y for a spatial unit might also be affected by some independent exploratory variables of other spatial units. In such a case, you can use the spatial Durbin model instead.

Spatial Durbin Models

Unlike a SAR model, a spatial Durbin model (SDM) can account for exogenous interaction effects in addition to the endogenous interaction effects. Let y_i denote the observation that is associated with a spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix of your choice. Further assume that $\mathbf{x}_i$ is a $p \times 1$ vector that denotes values of p regressors that are recorded for the spatial unit s_i and assume that $\mathbf{z}_i$ is a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDM can be described in vector form as (LeSage and Pace 2009)

$$\mathbf{y} = \rho\mathbf{W}\mathbf{y} + \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are $p \times 1$ and $q \times 1$ parameter vectors, respectively.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SDM as

$$\mathbf{y} = \rho\mathbf{W}\mathbf{y} + \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$$

The log-likelihood function for the SDM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^2} + \ln |\mathbf{A}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho\mathbf{W}$.

For the SDM, the gradients are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} &= \frac{\widetilde{\mathbf{X}}'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \rho} &= \frac{1}{\sigma^2} \mathbf{y}'\mathbf{W}'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}}) - \text{tr}(\mathbf{A}^{-1}\mathbf{W}) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^4}\end{aligned}$$

Both the SDM and the SAR model account for endogenous interaction effects. However, in some cases there might be an interaction among error terms. In such cases, you might consider a spatial error model, which addresses spatial interaction among error terms.

Spatial Error Models

The spatial error model (SEM) accounts for spatial dependence in the error terms rather than in the dependent variable. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SEM can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}\mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SEM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}$.

For the SEM, the gradients are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \frac{(\mathbf{B}\mathbf{X})' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \frac{1}{\sigma^2} [\mathbf{W}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] - \text{tr}(\mathbf{B}^{-1}\mathbf{W}) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}\end{aligned}$$

In addition to the interaction effects among error terms, you might also want to include exogenous interaction effects in the model. In such cases, you need to consider a spatial Durbin error model.

Spatial Durbin Error Models

The spatial Durbin error model (SDEM) accounts for spatial dependence among the error terms and the exogenous interaction effect. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDEM can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}\mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are $p \times 1$ and $q \times 1$ parameter vectors, respectively.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SDEM as

$$\mathbf{y} = \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \mathbf{B}^{-1}\boldsymbol{\epsilon}$$

where $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}$.

The log-likelihood function for the SDEM is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{B}|$$

For the SDEM, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}\widetilde{\mathbf{X}})' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] - \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Spatial Moving Average Models

The spatial moving average (SMA) model accounts for spatial dependence among the error terms; thus it is similar to the SEM model but with a different autocorrelation structure. The SMA model is used for modeling local autocorrelation. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SMA model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W})\boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} - \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}$.

For the SMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}^{-1}\mathbf{X})' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}\mathbf{W}] [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] + \text{tr}(\mathbf{B}^{-1}\mathbf{W})$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

Spatial Durbin Moving Average Models

The term spatial Durbin moving average (SDMA) model is used to refer to the SMA model that accommodates exogenous interaction effects. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be an $n \times n$ spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q covariates that are measured at unit s_i .

The SDMA model can be described in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{W}\mathbf{X}\boldsymbol{\theta} + (\mathbf{I}_n - \lambda \mathbf{W})\boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$; $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$; $\mathbf{X}$ and $\mathbf{Z}$ are $n \times p$ and $n \times q$ matrices, where each row consists of $\mathbf{x}_i'$ and $\mathbf{z}_i'$, respectively; $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector; and $\boldsymbol{\theta}$ is a $q \times 1$ parameter vector.

By letting $\tilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{W}\mathbf{Z}]$ and $\tilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDMA model can be written as

$$\mathbf{y} = \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}} + \mathbf{B}\boldsymbol{\epsilon}$$

The log-likelihood function for the SDMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]}{2\sigma^2} - \ln |\mathbf{B}|$$

where $\mathbf{B} = \mathbf{I}_n - \lambda\mathbf{W}$ and $|\mathbf{B}|$ denotes the determinant of matrix $\mathbf{B}$.

For the SDMA model, the gradients are

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}}} &= \frac{(\mathbf{B}^{-1}\tilde{\mathbf{X}})' [\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= -\frac{1}{\sigma^2} [\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}\mathbf{W}] [\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})] + \text{tr}(\mathbf{B}^{-1}\mathbf{W}) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1}(\mathbf{y} - \tilde{\mathbf{X}}\tilde{\boldsymbol{\beta}})]}{2\sigma^4} \end{aligned}$$

Spatial Autoregressive Moving Average Models

The spatial autoregressive moving average (SARMA) model, like the SMA model, can account for spatial dependence among the error terms. In addition, the SARMA model enables you to account for spatial dependence in the dependent variable, as the SAR model does. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let the $n \times n$ matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices that contain nonnegative elements. In practice, $\mathbf{W}_1$ and $\mathbf{W}_2$ can be identical. Further, it is assumed that $\mathbf{x}_i$ is a $p \times 1$ vector that denotes values of p covariates that are recorded at unit s_i .

The SARMA model can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \rho\mathbf{W}_1\mathbf{y} + \mathbf{X}\boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda\mathbf{W}_2)\boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} \mathbf{N}(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that consists of $\mathbf{x}_i'$ in each row, $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector, and $\mathbf{I}_n$ is an $n \times n$ identity matrix.

The log-likelihood function for the SARMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{A}| - \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho\mathbf{W}_1$, $\mathbf{B} = \mathbf{I}_n - \lambda\mathbf{W}_2$, and $|\cdot|$ denotes the matrix determinant operator.

For the SARMA model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}^{-1}\mathbf{X})' [\mathbf{B}^{-1}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \rho} &= \frac{(\mathbf{B}^{-1} \mathbf{W}_1 \mathbf{y})' \mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1) \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= -\frac{1}{\sigma^2} [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})]' [\mathbf{B}^{-1} \mathbf{W}_2] [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})] + \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})]' [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \mathbf{X} \boldsymbol{\beta})]}{2\sigma^4}\end{aligned}$$

Spatial Durbin Autoregressive Moving Average Models

You can also accommodate exogenous interaction effects in the SARMA model. The term spatial Durbin autoregressive moving average (SDARMA) model is used to refer to such an extension of the SARMA model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDARMA model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W}_1 \mathbf{Z} \boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W}_2) \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\tilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{W}_1 \mathbf{Z}]$ and $\tilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDARMA model can be written as

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}} + (\mathbf{I}_n - \lambda \mathbf{W}_2) \boldsymbol{\epsilon}$$

The log-likelihood function for the SDARMA model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{A}| - \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}_1$ and $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}_2$.

For the SDARMA model, the gradients are

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}}} &= \frac{(\mathbf{B}^{-1} \tilde{\mathbf{X}})' [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]}{\sigma^2} \\ \frac{\partial \mathcal{L}}{\partial \rho} &= \frac{(\mathbf{B}^{-1} \mathbf{W}_1 \mathbf{y})' \mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1) \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= -\frac{1}{\sigma^2} [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1} \mathbf{W}_2] [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})] + \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2) \\ \frac{\partial \mathcal{L}}{\partial \sigma^2} &= -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]' [\mathbf{B}^{-1} (\mathbf{A} \mathbf{y} - \tilde{\mathbf{X}} \tilde{\boldsymbol{\beta}})]}{2\sigma^4}\end{aligned}$$

Spatial Autoregressive Confused Models

The spatial autoregressive confused (SAC) model, like the SARMA model, can accommodate spatial dependence in both the dependent variable and error terms. However, the covariance structure for the error terms in a SAC model is different from that of the SARMA model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i .

The SAC model can be described in vector form by using the following two-stage formulation (LeSage and Pace 2009),

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{u}$$

$$\mathbf{u} = \lambda \mathbf{W}_2 \mathbf{u} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SAC model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^2} + \ln |\mathbf{A}| + \ln |\mathbf{B}|$$

where $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}_1$ and $\mathbf{B} = \mathbf{I}_n - \lambda \mathbf{W}_2$.

For the SAC model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{(\mathbf{B}\mathbf{X})' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}\mathbf{W}_1 \mathbf{y})' \mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}_2(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})] - \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \mathbf{X}\boldsymbol{\beta})]}{2\sigma^4}$$

Spatial Durbin Autoregressive Confused Models

The SAC model can be extended to account for exogenous interaction effects. The term spatial Durbin autoregressive confused (SDAC) model is used to refer to such an extension of the SAC model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}_1$ and $\mathbf{W}_2$ be two spatial weights matrices. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors that are recorded at unit s_i , and assume that $\mathbf{z}_i$ is a $q \times 1$ vector that denotes values of q regressors that are measured at unit s_i .

The SDAC model can be described in vector form by using the following two-stage formulation,

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W}_1 \mathbf{Z} \boldsymbol{\theta} + \mathbf{u}$$

$$\mathbf{u} = (\mathbf{I}_n - \lambda \mathbf{W}_2)^{-1} \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix that has $\mathbf{x}_i'$ in each row, $\mathbf{Z}$ is an $n \times q$ matrix that has $\mathbf{z}_i'$ in each row, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{W}_1 \mathbf{Z}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, the SDAC model can be rewritten as

$$\mathbf{y} = \rho \mathbf{W}_1 \mathbf{y} + \widetilde{\mathbf{X}} \widetilde{\boldsymbol{\beta}} + (\mathbf{I}_n - \lambda \mathbf{W}_2)^{-1} \boldsymbol{\epsilon}$$

The log-likelihood function for the SDAC model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^2} + \ln |\mathbf{A}| + \ln |\mathbf{B}|$$

For the SDAC model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\mathbf{B}\widetilde{\mathbf{X}})' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(\mathbf{B}\mathbf{W}_1 \mathbf{y})' \mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2} - \text{tr}(\mathbf{A}^{-1} \mathbf{W}_1)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [\mathbf{W}_2(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})] - \text{tr}(\mathbf{B}^{-1} \mathbf{W}_2)$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]' [\mathbf{B}(\mathbf{A}\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})]}{2\sigma^4}$$

Linear Regression Models

You can also fit a linear regression model in PROC CSPATIALREG. In this case, let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$, and let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors recorded at unit s_i .

You can describe the linear regression model in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, and $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$.

The log-likelihood function for the linear regression model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^2}$$

For the linear regression model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \frac{\mathbf{X}'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{2\sigma^4}$$

The Hessians take the following forms:

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'} = -\frac{\mathbf{X}'\mathbf{X}}{\sigma^2}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta} \partial \sigma^2} = -\frac{\mathbf{X}'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^4}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})}{\sigma^6}$$

Spatial Lag of X Models

The spatial lag of X (SLX) model assumes no endogenous interaction effects or spatial dependence in the error terms. Instead, it incorporates only exogenous interaction effects into the linear regression model. Let y_i denote the observation that is associated with the spatial unit s_i for $i = 1, 2, \dots, n$. For these spatial units, let $\mathbf{W}$ be a spatial weights matrix. Further, let $\mathbf{x}_i$ be a $p \times 1$ vector that denotes values of p regressors recorded at unit s_i , and let $\mathbf{z}_i$ be a $q \times 1$ vector that denotes values of q regressors measured at unit s_i .

The SLX model can be described in vector form as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{WZ}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)'$, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots, \epsilon_n)'$ with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, $\mathbf{X}$ is an $n \times p$ matrix where each row consists of $\mathbf{x}_i'$, $\mathbf{Z}$ is an $n \times q$ matrix where each row consists of $\mathbf{z}_i'$, and $\boldsymbol{\beta}$ is a $p \times 1$ parameter vector.

By letting $\widetilde{\mathbf{X}} = [\mathbf{X} \ \mathbf{WZ}]$ and $\widetilde{\boldsymbol{\beta}} = (\boldsymbol{\beta}' \ \boldsymbol{\theta}')'$, you can rewrite the SLX model as

$$\mathbf{y} = \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}} + \boldsymbol{\epsilon}$$

The log-likelihood function for the SLX model is

$$\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^2}$$

For the SLX model, the gradients are

$$\frac{\partial \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}}} = \frac{(\widetilde{\mathbf{X}})'(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{2\sigma^4}$$

The Hessians take the following forms:

$$\frac{\partial^2 \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}} \partial \widetilde{\boldsymbol{\beta}}'} = -\frac{\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}}}{\sigma^2}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \widetilde{\boldsymbol{\beta}} \partial \sigma^2} = -\frac{\widetilde{\mathbf{X}}'(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^4}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})'(\mathbf{y} - \widetilde{\mathbf{X}}\widetilde{\boldsymbol{\beta}})}{\sigma^6}$$

Specifying the Spatial Weights Matrix

The spatial weights matrix $\mathbf{W}$ plays a vital role in spatial econometric modeling. If you fit a purely linear model without SLX effects, you do not need a $\mathbf{W}$ matrix. For other types of models in PROC CSPATIALREG, you need to provide a spatial weights matrix to fit the model. Although how you create the $\mathbf{W}$ matrix is often problem-specific, there are some general guidelines to consider. Two common ways to create the $\mathbf{W}$ matrix are k -order binary contiguity matrices and k -nearest-neighbor matrices (Elhorst 2013).

k -Order Binary Contiguity Matrices

To create the $\mathbf{W}$ matrix by using k -order binary contiguity matrices, you start with the spatial contiguity matrix $\mathbf{C}$. In the case of the first-order neighbors ($k = 1$), a value of 1 for the (i, j) th entry in $\mathbf{C}$ indicates that the two units i and j are neighbors to each other, and 0 indicates otherwise. The neighbor relationship is often defined based on sharing of a common boundary. To generalize this, a k -order neighbor ($k \geq 2$) of a unit i can be any units whose neighbors are $(k - 1)$ -order neighbors of unit i . In this sense, the two units i and j that are not first-order neighbors can still be second-order neighbors if unit j is the neighbor to a first-order neighbor of unit i .

As an example, a first-order binary contiguity matrix might look like the following:

$$\mathbf{C} = \begin{pmatrix} \text{SID} & \text{L1} & \text{L2} & \text{L3} & \text{L4} \\ \text{L1} & 0 & 1 & 0 & 1 \\ \text{L2} & 1 & 0 & 0 & 0 \\ \text{L3} & 0 & 0 & 0 & 1 \\ \text{L4} & 1 & 0 & 1 & 0 \end{pmatrix}$$

The diagonal elements of $\mathbf{C}$ are zeros because, in general, a unit is not considered to be a neighbor of itself. Moreover, the two units L2 and L4 are neighbors of L1; L2 has L1 as its only neighbor; L3 has L4 as its only neighbor; and L4 has L1 and L3 as its neighbors. You can create the spatial weights matrix $\mathbf{W}$ by row-standardizing the contiguity matrix $\mathbf{C}$. To do so, you divide entries in each row of $\mathbf{C}$ by the sum of that row. The spatial weights matrix $\mathbf{W}$, which is the row-standardized version of $\mathbf{C}$, is as follows:

$$\mathbf{W} = \begin{pmatrix} \text{SID} & \text{L1} & \text{L2} & \text{L3} & \text{L4} \\ \text{L1} & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \text{L2} & 1 & 0 & 0 & 0 \\ \text{L3} & 0 & 0 & 0 & 1 \\ \text{L4} & \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

k -Nearest-Neighbor Matrices

To create the $\mathbf{W}$ matrix by using k -nearest-neighbor matrices, you can start by creating a spatial contiguity matrix based on a distance metric. Let d_{ij} denote the distance between the two units i and j , which might be the Euclidean distance between centroids of the two spatial units. Let $(\text{lon}_i, \text{lat}_i)$ and $(\text{lon}_j, \text{lat}_j)$ be the centroids of units i and j , where $1 \leq i, j \leq n$, and lon and lat denote the longitude and latitude, respectively. Under the Euclidean distance metric, the distance d_{ij} between units i and j is

$$d_{ij} = \sqrt{(\text{lat}_i - \text{lat}_j)^2 + (\text{lon}_i - \text{lon}_j)^2}$$

After computing the distance between the unit i and other units under a certain metric, you sort d_{ij} in ascending order; for example, $d_{ij_1} \leq d_{ij_2} \leq \dots \leq d_{ij_k} \leq \dots \leq d_{ij_{n-1}}$. For a given k , let $N_k(i) = \{j_1, j_2, \dots, j_k\}$ be the set that contains the indices of k -nearest neighbors of unit i ; then the (i, j) th entries of the contiguity matrix $\mathbf{C}$ are defined as

$$C_{ij} = \begin{cases} 1 & \text{if } j \in N_k(i) \\ 0 & \text{otherwise} \end{cases}$$

The (i, j) th entry of the corresponding row-standardized matrix $\mathbf{W}$ is $W_{ij} = C_{ij} \left\{ \sum_{j \in N_k(i)} C_{ij} \right\}^{-1}$.

Unlike the k -order binary contiguity matrix, which is often symmetric by construction, k -nearest-neighbor matrices can be asymmetric. To obtain symmetric k -nearest neighbor matrices, you can define the (i, j) th entries of the contiguity matrix $\mathbf{C}$ as follows:

$$C_{ij} = \begin{cases} 1 & \text{if } j \in N_k(i) \text{ or } i \in N_k(j) \\ 0 & \text{otherwise} \end{cases}$$

In addition to the Euclidean distance measure, you can use other distance metrics as appropriate. A variant of k -nearest-neighbor matrices $\mathbf{C}^*$ that is used in some empirical studies defines its (i, j) th entries as

$$C_{ij}^* = \begin{cases} 1 & \text{if } d_{ij} \leq d_{\text{cutoff}} \\ 0 & \text{otherwise} \end{cases}$$

where d_{cutoff} is a prespecified threshold distance.

In addition to the two constructions of spatial weights matrices that are presented earlier, see Elhorst (2013) and the references therein for more information about other ways to create a spatial weights matrix. In practice, you can define the neighbor relation that is problem-specific. For example, you can define two spatial units that are far apart to be neighbors because they share some attributes (such as population sizes larger than 500,000).

The data table that you specify in the `WMAT=` option is row-standardized by default to create a spatial weights matrix. This means that if you specify `WMAT=C`, `PROC CSPATIALREG` row-standardizes the spatial contiguity matrix to create a spatial weights matrix. If you want to suppress row standardization, you must specify the `NONNORMALIZE` option in the `PROC CSPATIALREG` statement.

Compact Representation of the Spatial Weights Matrix

When the number of spatial units n increases, the amount of memory that it takes to store n^2 entries of the spatial contiguity matrix $\mathbf{C}$ or the spatial weights matrix $\mathbf{W}$ increases dramatically. To circumvent the storage issue, `PROC CSPATIALREG` enables you to provide a compact representation of $\mathbf{W}$ (or $\mathbf{C}$) when appropriate. For the compact matrix representation, you provide a data table that contains three variables by using the `WMAT=` option. The first two variables identify the row r and column c of $\mathbf{W}$ (or $\mathbf{C}$), and (r, c) can be expressed either as numerical indices or as values of the variable that you specify in the `SPATIALID` statement. The third variable contains the nonzero value of $\mathbf{W}$ (or $\mathbf{C}$) for row r and column c . In this compact representation, the number of observations in the data table that you specify in the `WMAT=` option equals the total number of nonzero entries in $\mathbf{W}$ (or $\mathbf{C}$).

You must use a `SPATIALID` statement if you want to use the compact representation of the spatial contiguity or spatial weights matrix. In the compact representation, the first two variables of the data table that you specify in the `WMAT=` option must be of the same type. First, the first two columns in that data table can

be the row and column index for each nonzero entry in **W** (or **C**). In this case, the SPATIALID variable is numeric type. Alternatively, the first two columns in the WMAT= data table can be characters that are the names of two neighboring spatial units in **W** (or **C**). In this second case, the SPATIALID variable is character type.

For example, the compact representation of the spatial weights matrix **W**,

$$\mathbf{W} = \begin{pmatrix} 0 & 0.5 & 0 & 0.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0.5 & 0 \end{pmatrix}$$

would look like the following:

```
data Ws;
  input SID cSID Weight;
  datalines;
  1 2 0.5
  1 4 0.5
  2 1 1.0
  3 4 1.0
  4 1 0.5
  4 3 0.5
  ;
run;
```

For the spatial contiguity matrix **C**,

$$\mathbf{C} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

the compact representation would look like the following:

```
data Cs;
  input SID cSID Weight;
  datalines;
  1 2 1.0
  1 4 1.0
  2 1 1.0
  3 4 1.0
  4 1 1.0
  4 3 1.0
  ;
run;
```

If the spatial weights matrix is the same as matrix **W** in the section “*k*-Order Binary Contiguity Matrices” on page 510, its compact representation would be as follows:

```
data Ws2;
  input SID $ cSID $ Weight;
  datalines;
  L1 L2 0.5
```

```

L1 L4 0.5
L2 L1 1.0
L3 L4 1.0
L4 L1 0.5
L4 L3 0.5
;
run;

```

If the spatial contiguity matrix is the same as matrix **C** in the section “*k*-Order Binary Contiguity Matrices” on page 510, its compact representation can be given in the data set **Cs2** as follows:

```

data Cs2;
  input SID $ cSID $ Weight;
  datalines;
L1 L2 1.0
L1 L4 1.0
L2 L1 1.0
L3 L4 1.0
L4 L1 1.0
L4 L3 1.0
;
run;

```

Spatial ID Matching

Depending on the type of model that you use in PROC CSPATIALREG, you might need to specify two data tables: one in the **DATA=** option and the other in the **WMAT=** option. In such cases, you must use a **SPATIALID** statement to specify a spatial ID variable in order to match observations in these two data tables.

As an example, assume that the data table that you specify in the **DATA=** option looks like the following:

```

data mycas.Example;
  input SID $ x1 x2 y;
  datalines;
L1 0.3 0.5 0.9
L3 -0.7 0.8 -0.4
L2 0.4 -1.2 0.6
L8 -1.7 1.2 -0.5
L4 1.4 0.9 0.3
L5 2.3 1.5 1.9
L7 -0.9 -0.8 -1.3
L6 1.4 -1.6 -2.0
;
run;

```

Suppose that the spatial contiguity matrix that you specify in the **WMAT=** option looks like the following:

```

data mycas.Cmat;
  input SID $ L1 L8 L3 L4 L7 L6 L5 L2;
  datalines;
L1 0 1 0 1 0 1 0 1
L2 1 0 0 0 1 0 0 0
L6 1 0 1 0 0 0 0 0

```

```

L4  1 0 0 0 1 0 0 0
L3  0 0 0 0 1 1 0 0
L7  0 0 1 1 0 0 0 1
L5  0 1 0 0 0 0 0 0
L8  1 0 0 0 0 0 1 0
;
run;

```

As you can see, rows in the two data tables `mycas.Example` and `mycas.Cmat` do not share identically sorted SID values. The second row in the `Example` data table contains the observation for a spatial unit L3, and its neighbor information is given in the fifth row of the `mycas.Cmat` data table. Moreover, the rows and columns of the spatial weights data table `mycas.Cmat` are not in the same order. The following SAS statements fit a SAR model to these data:

```

proc cspatialreg data=mycas.Example wmat=mycas.Cmat;
  model y=x1 x2/type=SAR;
  spatialid SID;
run;

```

The `SPATIALID` statement enables you to match rows and columns of the `mycas.Cmat` data table in addition to rows of the `mycas.Example` and `mycas.Cmat` data tables.

Parameter Space of Autoregressive Parameters

All models except linear regression models require that the parameters ρ and λ satisfy some assumptions to ensure consistency of the maximum likelihood estimator (Elhorst 2013). For SDMs and SAR models, the Jacobian term involves the log-determinant of $\mathbf{I} - \rho\mathbf{W}_1$, and the parameter space of ρ is specified such that $\mathbf{I} - \rho\mathbf{W}_1$ is nonsingular. For the SEM, SDEM, SMA, and SDMA models, the Jacobian term involves the log-determinant of $\mathbf{I} - \lambda\mathbf{W}_1$, and the parameter space of λ is specified such that $\mathbf{I} - \lambda\mathbf{W}_1$ is nonsingular. For the SAC, SDAC, SARMA, and SDARMA models, both $\mathbf{I} - \rho\mathbf{W}_1$ and $\mathbf{I} - \lambda\mathbf{W}_2$ must be nonsingular. As a result, the parameter space of ρ and λ is often specified such that both $\mathbf{I} - \rho\mathbf{W}_1$ and/or $\mathbf{I} - \lambda\mathbf{W}_1$ are nonsingular.

In the `CSPATIALREG` procedure, the parameter space of the parameters ρ and λ depends on the spatial weights matrix $\mathbf{W}$ that you choose. For $\mathbf{W}$, the parameter space of the parameters ρ and λ in `PROC CSPATIALREG` is determined as follows:

1. For a symmetric $\mathbf{W}$, the nonsingularity condition requires $\rho \in (\omega_{\min}^{-1}, \omega_{\max}^{-1})$ and $\lambda \in (\omega_{\min}^{-1}, \omega_{\max}^{-1})$. Here $\omega_{\min}$ and $\omega_{\max}$ denote the smallest (that is, most negative) and largest real eigenvalues of $\mathbf{W}$, respectively.
2. If $\mathbf{W}$ is symmetric and subsequently row-standardized, the nonsingularity condition requires $\rho \in (\omega_{\min}^{*-1}, 1)$ and $\lambda \in (\omega_{\min}^{*-1}, 1)$. Here $\omega_{\min}^*$ denotes the smallest purely real eigenvalue of the row-standardized $\mathbf{W}$.
3. If $\mathbf{W}$ is asymmetric and subsequently row-standardized, the nonsingularity condition requires $\rho \in (r_{\min}^{*-1}, 1)$ and $\lambda \in (r_{\min}^{*-1}, 1)$. Here $r_{\min}^*$ denotes the smallest purely real eigenvalue of the row-standardized $\mathbf{W}$.

4. When Taylor or Chebyshev approximation is used for SDMs and SAR models, $\mathbf{W}$ is required to be row-standardized. In these cases, the restriction on the spatial autoregressive coefficient ρ is $\rho \in (-1, 1)$.
5. When Taylor or Chebyshev approximation is used for SEMs and SDEMs, $\mathbf{W}$ is required to be row-standardized. In these cases, the restriction on the spatial autocorrelation coefficient λ is $\lambda \in (-1, 1)$.

Approximations to the Jacobian

To obtain maximum likelihood estimates for all models except linear regression models, the Jacobian term must be computed, because it appears in the log-likelihood function. The Jacobian term is $\ln |\mathbf{I}_n - \rho \mathbf{W}|$ for SDMs and SAR models and $\ln |\mathbf{I}_n - \lambda \mathbf{W}|$ for SEMs and SDEMs, where n is the number of observations and $\mathbf{W}$ is the spatial weights matrix. When n is not large, the Jacobian is computed as follows,

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| = \sum_{i=1}^n \ln |1 - \rho \omega_i|$$

where ω_i are the eigenvalues of $\mathbf{W}$. Such a method requires you to precompute all eigenvalues of $\mathbf{W}$. This works fine for small data sets, but when n is large, computing the Jacobian term by using the eigenvalue method can be computationally infeasible. Instead, you can use approximations to the Jacobian.

The CSPATIALREG procedure supports two different approximations to the Jacobian. For SDMs, SEMs, SDEMs, and SAR models, you can approximate the Jacobian by using either Taylor or Chebyshev approximation. Using SDMs and SAR models as an example, the two approximations for the Jacobian term can be described as follows (for more information, see LeSage and Pace 2009, and the references therein):

- Taylor approximation uses finite, lower-order series to approximate the log-determinant as

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| \approx - \sum_{k=1}^q \frac{\rho^k \text{tr}(\mathbf{W}^k)}{k}$$

- Chebyshev approximation uses finite, lower-order Chebyshev polynomials of the first kind to approximate the log-determinant as

$$\ln |\mathbf{I}_n - \rho \mathbf{W}| \approx \sum_{k=0}^q c_k(\rho) \text{tr}(T_k(\mathbf{W}))$$

where $T_0(\mathbf{W}) = \mathbf{I}_n$, $T_1(\mathbf{W}) = \mathbf{W}$, and $T_{k+1}(\mathbf{W}) = 2\mathbf{W}T_k(\mathbf{W}) - T_{k-1}(\mathbf{W})$ for $k = 1, 2, \dots, q$. The coefficients $c_k(\rho)$ are defined as

$$c_k(\rho) = \begin{cases} \frac{1}{q+1} \sum_{j=0}^q \ln(1 - \rho \cos \theta_j) \cos(k\theta_j) & \text{if } k = 0 \\ \frac{2}{q+1} \sum_{j=0}^q \ln(1 - \rho \cos \theta_j) \cos(k\theta_j) & \text{if } k > 0 \end{cases}$$

with $\theta_j = (j+1)\pi/(q+1)$ for $j = 0, 1, \dots, q$.

The traces of powers of $\mathbf{W}$ can be computed exactly or approximated using Monte Carlo simulation. The Monte Carlo simulation is done as follows,

$$\text{tr}(\mathbf{W}^k) \approx \frac{1}{M} \sum_{l=1}^M \frac{n}{\mathbf{u}_l' \mathbf{u}_l} \mathbf{u}_l' \mathbf{W}^k \mathbf{u}_l$$

where $\mathbf{u}_l \stackrel{\text{iid}}{\sim} N(\mathbf{0}, \mathbf{I}_n)$ and M is the total number of Monte Carlo samples.

When you apply these two approximations, it is often assumed that the maximum eigenvalue of $\mathbf{W}$ equals 1 and the minimum eigenvalue of $\mathbf{W}$ is greater than or equal to -1 (see LeSage and Pace 2009, and the references therein). One way to satisfy this assumption is to use a row-standardized spatial weights matrix that is similar to a symmetric matrix. If the spatial weights matrix is not symmetric or similar to a symmetric matrix, it becomes more difficult to apply Chebyshev approximation and thus requires extra care (LeSage and Pace 2009).

When you request an approximation to the Jacobian, the choices that you need to make might include the approximation method to use (that is, Taylor or Chebyshev approximation), the order of series q , and the number of Monte Carlo samples (that is, M). Your choice can be accommodated through the `APPROXIMATION=` option in the PROC CSPATIALREG statement. For the approximation method, you can use the keyword TAYLOR in the `APPROXIMATION=` option to request Taylor approximation. Otherwise, the approximation method defaults to Chebyshev approximation. You specify `ORDER=q` in the `APPROXIMATION=` option to request a series of order q when approximating the log-determinant. In addition, you specify `NMC=M` in the `APPROXIMATION=` option to request M Monte Carlo samples to be drawn when approximating the traces of powers of $\mathbf{W}$. In addition, you can use the `SEED=` suboption of the `APPROXIMATION=` option to specify an integer seed for a random number generator to replicate your analysis.

Impact Estimation

For models in PROC CSPATIALREG that include endogenous interactions or exogenous interactions, the interpretation of marginal effects is not as straightforward as it is in the classical linear regression model. For example, consider the following SDM model:

$$\mathbf{y} = \rho \mathbf{W} \mathbf{y} + \mathbf{X} \boldsymbol{\beta} + \mathbf{W} \mathbf{X} \boldsymbol{\theta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$.

For an explanatory variable x_p in the model,

$$\frac{\partial y_i}{\partial x_{jp}} = [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ij}$$

where $\mathbf{A} = \mathbf{I} - \rho \mathbf{W}$ and $[\mathbf{B}]_{ij}$ denotes the (i, j) th element of the matrix $\mathbf{B}$ for $i, j = 1, 2, \dots, n$.

According to $\frac{\partial y_i}{\partial x_{jp}}$, you can see that the impact of changes in x_p at a location j on the dependent variable differ from one location to another. To quantify the marginal effects of an explanatory variable x_p in the model, LeSage and Pace (2009) propose the average direct impacts ($\bar{M}(p)_{\text{direct}}$), the average total impacts ($\bar{M}(p)_{\text{total}}$), and the average indirect impacts ($\bar{M}(p)_{\text{indirect}}$) as follows:

$$\bar{M}(p)_{\text{direct}} = \frac{1}{n} \sum_{i=1}^n [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ii}$$

$$\bar{M}(p)_{\text{total}} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n [\mathbf{A}^{-1}(\beta_p \mathbf{I}_n + \theta_p \mathbf{W})]_{ij}$$

$$\bar{M}(p)_{\text{indirect}} = \bar{M}(p)_{\text{total}} - \bar{M}(p)_{\text{direct}}$$

In addition, LeSage and Pace (2009) describe how to use Monte Carlo simulation to quantify uncertainty that is associated with the three impact estimates. First, the empirical distribution for parameters (the regression coefficients and, if applicable, the spatial dependence parameter ρ) is assumed to be a multivariate normal distribution where the mean vector contains the point estimates and the covariance matrix is estimated based on the maximum likelihood estimation. Second, M samples are drawn from the empirical distribution and for each sample being drawn, $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ are computed. Based on these random samples, the sample mean and standard deviation for each of three impact estimates can be computed.

When $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ are computed, the inverse of $\mathbf{A} = \mathbf{I}_n - \rho \mathbf{W}$ needs to be computed for each random sample drawn. Since the exact inversion of $\mathbf{A}$ can be computationally intensive, approximation can be helpful when it is applicable. When the spectral radius of $\rho \mathbf{W}$ is less than 1, $\mathbf{A}^{-1}$ can be approximated by the partial sums of Neumann series as

$$(\mathbf{I}_n - \rho \mathbf{W})^{-1} = \sum_{o=0}^{+\infty} (\rho \mathbf{W})^o \approx \mathbf{I}_n + \rho \mathbf{W} + \cdots + \rho^O \mathbf{W}^O$$

The IMPACT option in the PROC CSPATIALREG statement computes $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ by using the Monte Carlo simulation that is proposed by LeSage and Pace (2009). In particular, you specify NMC= M in the IMPACT option to request that M samples be drawn from the empirical distribution of the relevant model parameters. You use the ORDER= O suboption of the IMPACT option to specify the highest order of Neumann series to approximate $\mathbf{A}^{-1}$. You use the SEED= suboption of the IMPACT option to specify an integer seed for a random number generator to replicate your results.

Impact estimation is not supported for models that include any of the following: a classification variable, interactions between regressions, a BOUNDS statement, or a RESTRICT statement. In addition, Monte Carlo simulation is not needed to compute summary statistics for $\bar{M}(p)_{\text{direct}}$, $\bar{M}(p)_{\text{total}}$, and $\bar{M}(p)_{\text{indirect}}$ for linear, SLX, SEM, SDEM, SMA, and SDMA models because a closed-form solution is available ($\bar{M}(p)_{\text{direct}} = \beta_p$, $\bar{M}(p)_{\text{total}} = \beta_p + \theta_p$, and $\bar{M}(p)_{\text{indirect}} = \theta_p$). In these models, any NMC=, ORDER=, and SEED= suboptions that you specify in the IMPACT option are ignored. When you specify the IMPACT option for SAR, SDM, SAC, SDAC, SARMA, and SDARMA models, the spatial weights matrix $\mathbf{W}$ must be row-standardized. Moreover, the restriction $-1 < \rho < 1$ is imposed for SAR and SDM models and the restrictions $-1 < \rho < 1$ and $-1 < \lambda < 1$ are imposed for SAC, SDAC, SARMA, and SDARMA models. With the restriction $-1 < \rho < 1$, the average total impacts $\bar{M}(p)_{\text{total}}$ can be simplified as

$$\bar{M}(p)_{\text{total}} = \frac{1}{1 - \rho} (\beta_p + \theta_p)$$

Parameter Naming Conventions for RESTRICT, TEST, BOUNDS, and INIT Statements

This section describes how you can refer to the parameters in the MODEL and SPATIALEFFECTS statements when you use the RESTRICT, TEST, BOUNDS, or INIT statement. The following examples use the RESTRICT statement, but the same remarks apply to naming parameters when you use the TEST, BOUNDS, or INIT statement. The names of the parameters are displayed in the “Parameter Estimates” table if you specify the option. If a parameter name contains a blank or some other special character (such as ‘*’, ‘-’, ‘(’, or ‘)'), then you must use the internal name of the parameter to refer to that parameter in the RESTRICT, TEST, BOUNDS, or INIT statement. You can specify the PRINTINTERNALNAMES option in the PROC CSPATIALREG statement if you want to see the internal names of the parameters. When you specify the PRINTINTERNALNAMES option, the “Parameter Estimates” table includes an additional column that shows the internal name of each parameter.

To impose a restriction on a parameter that is related to a regressor in the MODEL statement, you simply use the name of the regressor itself. Suppose that your model is

```
model y = x1-x3 / type=SAR;
```

where x1–x3 are continuous variables. Suppose you want to restrict the parameter that is associated with the regressor x3 to be greater than 1.7. You should provide the following statement:

```
RESTRICT x3 > 1.7;
```

To impose a restriction on a parameter that is associated with a regressor in the SPATIALEFFECTS statement, you can form the name of the parameter by adding the prefix W_ to the name of the regressor. Suppose that your MODEL and SPATIALEFFECTS statements are as follows:

```
model y = x1-x3 / type=SAR;  
spatialeffects x1 x2 x3;
```

Suppose you want to restrict the parameter that is related to the x3 regressor in the SPATIALEFFECTS statement to be less than 1.0. You should refer to the parameter as W_x3 and provide the following statement:

```
RESTRICT W_x3 < 1.0;
```

Even though the regressor x3 appears in both the MODEL and SPATIALEFFECTS statements, the parameter that is associated with x3 in the MODEL statement is, of course, different from the parameter that is associated with x3 in the SPATIALEFFECTS statement. Thus, when you use the name of a regressor in a RESTRICT statement without any prefix, it refers to the parameter that is associated with that regressor in the MODEL statement. Meanwhile, when you use the name of a regressor in a RESTRICT statement with the prefix W_, it refers to the parameter that is associated with that regressor in the SPATIALEFFECTS statement. Note that the intercept is not included in the SPATIALEFFECTS statement.

Referring to Class Level Parameters

When your MODEL includes a classification variable, you can impose restrictions on the parameters associated with each of the levels that are related to the classification variable as follows.

Suppose that your classification variable is named *C* and has three levels: 0, 1, and 2. Suppose that your model is the following:

```
class C;
model y = x1 x2 C;
```

Adding a classification variable as a regressor to your model introduces additional parameters to your model, each of which is associated with one of the levels of the classification variable. You can form the name of the parameter that is associated with a particular level of your classification variable by inserting the underscore character between the name of the classification variable and the value of the level. Thus, to restrict the parameter that is associated with level 0 of the classification variable *C* to always be greater than 0.7, you refer to the parameter as *C_0* and provide the following statement:

```
RESTRICT C_0 > 0.7;
```

Referring to Parameters Associated with Interactions between Regressors

When a regressor in your model involves an interaction between other regressors, you can impose restrictions on the parameters that are associated with the interaction as described in this section.

Suppose you have the following model:

```
model y = x1 x2 x3*x4;
```

You can form the name of the parameter that is associated with the interaction regressor *x3*x4* by replacing the multiplication sign with an underscore. Thus, *x3_x4* refers to the parameter that is associated with the interaction regressor *x3*x4*.

You refer to interactions between regressors and CLASS variables in exactly the same way. Suppose you have a CLASS variable named *C* that has three levels (0, 1, and 2), and your model is the following:

```
class C;
model y = x1 x2 C*x3;
```

The interaction between the continuous variable *x3* and the CLASS variable *C* introduces three additional parameters, which are named *x3_C_0*, *x3_C_1*, and *x3_C_2*. Note that, although the order of the terms in the interaction is *C* followed by *x3*, the name of the parameter that is associated with the interaction is formed by placing the name of the continuous variable *x3* first, followed by an underscore, followed by the name of the CLASS variable *C*, then another underscore, and then the level value. Once again, depending on the parameterization that you specify in your CLASS statement, for each interaction in your model that involves a CLASS variable, one of the parameters that are associated with that interaction can be dropped from your model before optimization.

The name of a parameter that is associated with a nested interaction is formed in a slightly different way. Suppose you have a CLASS variable named *C* that has three levels (0, 1, and 2) and your model is the following:

```
class C;
model y = x1 x2 x3(C);
```

The nested interaction between the continuous variable `x3` and the CLASS variable `C` introduces three additional parameters, which are named `x3_C__0`, `x3_C__1`, and `x3_C__2`. Note how the name in each case was formed from the name of the regressor by replacing the left and right parentheses with underscores and then appending another underscore, followed by the level value.

Referring to Class Level Parameters with Negative Values

When the value of a level is a negative number, you must replace the minus sign with an underscore when you form the name of the parameter that is associated with that particular level of the classification variable. For example, suppose your classification variable is named `D` and has four levels: `-1`, `0`, `1`, and `2`. Suppose that your model is the following:

```
class D;
model y = x1 x2 D;
```

To restrict the parameter that is associated with level `-1` of the classification variable `D` to always be less than `0.4`, you refer to the parameter as `D__1` (note that there are two underscores in this parameter name: one to connect the name of the classification variable to its value and the other to replace the minus sign in the value itself) and provide the following statement:

```
RESTRICT D__1 < 0.4;
```

Dropping a Class Level Parameter to Avoid Collinearity

Depending on the parameterization that you impose on your classification variable, one of the parameters that are associated with its levels might be dropped from your model before optimization in order to avoid collinearity. For example, when the default parameterization GLM is imposed, the parameter that is associated with the last level of your classification variable is dropped before optimization. If you attempt to impose a restriction on a dropped parameter by using the `RESTRICT` statement, PROC CSPATIALREG issues an error message in the log.

For example, suppose again that your classification variable is named `C` and has three levels: `0`, `1`, and `2`. Suppose that your model is the following:

```
class C;
model y = x1 x2 C;
```

Because no additional options are specified in the CLASS statement, GLM parameterization is assumed. This means that the parameter named `C_2` (which is the parameter associated with the last level of your classification variable) is dropped from your model before the optimizer is invoked. Therefore, you get an error if you attempt to restrict the `C_2` parameter in any way by referring to it in a `RESTRICT` statement. For example, the following `RESTRICT` statement will generate an error:

```
RESTRICT C_2 < 0.3;
```

Referring to Implicit Parameters

For all models in PROC CSPATIALREG, one or more implicit parameters are added to your model before optimization. You can impose restrictions on these implicit parameters as follows.

If you have a linear model or an SLX model, the `_sigma2` parameter is added to your model. For the SDM or SAR model, the `_rho` and `_sigma2` parameters are added to your model. For the SEM or SDEM, the `_lambda` and `_sigma2` parameters are added to your model.

Whenever your model type dictates the addition of one or more of these implicit parameters, you can impose restrictions on the implicit parameters by referring to them by name. For example, assuming that your model type implies the existence of the `_rho` parameter, you can restrict `_rho` to be greater than 0 as follows:

```
RESTRICT _rho > 0.0;
```

Computational Resources

The time and memory that PROC CSPATIALREG requires are proportional to the number of parameters in the model and the number of observations in the data that you are analyzing. Less time and memory are required for smaller models and fewer observations. When you run PROC CSPATIALREG in the high-performance distributed environment, the amount of time that is required is also affected by the number of nodes and the number of threads per node when the optimization task is distributed.

The method that you use to calculate the variance-covariance matrix and the optimization method also affect the time and memory resources. All optimization methods available through the `METHOD=` option have similar memory use requirements. The processing time might differ among methods, depending on the number of iterations and functional calls that are needed. The data is read into memory to save processing time. If not enough memory is available to hold the data, the CSPATIALREG procedure stores the data in a utility file on disk and rereads the data as needed from this file, substantially increasing the execution time of the procedure. The gradient and the variance-covariance matrix must be held in memory. If the model has p parameters, including the intercept, then you need at least $8 * (p + p * (p + 1)/2)$ bytes of memory. The processing time is also a function of the number of iterations that are needed to converge to a solution for the model parameters. The number of iterations that are needed cannot be known in advance. You can use the `MAXITER=` option to limit the number of iterations that PROC CSPATIALREG performs. You can alter the convergence criteria by using the nonlinear optimization options available in the PROC CSPATIALREG statement. For a list of all the nonlinear optimization options, see the section “[Optimization Control Options](#)” on page 490.

Covariance Matrix Types

The CSPATIALREG procedure enables you to specify the estimation method for the covariance matrix. You have three choices, which you can specify in either the PROC CSPATIALREG or MODEL statement: the COVEST=HESSIAN option estimates the covariance matrix based on the inverse of the Hessian matrix, the COVEST=OP option uses the outer product of gradients, and the COVEST=QML option produces the covariance matrix based on both the Hessian and outer product matrices. By default, COVEST=HESSIAN.

Although all three methods produce asymptotically equivalent results, they differ in computational intensity and produce results that might differ in finite samples. The COVEST=OP option provides the covariance matrix that is usually the easiest to compute. In some cases, the OP approximation is considered more efficient than the Hessian or QML approximation because it contains fewer random elements. The QML approximation is computationally the most complex, because it requires both the outer product of gradients and the Hessian matrix. In most cases, the OP or Hessian approximation is preferred to the QML approximation. You need to use the QML approximation in some cases when the model is misspecified and the information matrix equality does not hold.

For SDMs, SEMs, SDEMs, and SAR models, only COVEST=HESSIAN is supported.

Displayed Output

PROC CSPATIALREG produces the following displayed output.

Class Level Information

If you specify the CLASS statement, the CSPATIALREG procedure displays a table that contains the following information:

- CLASS variable name
- number of levels of the CLASS variable
- list of values of the CLASS variable

Model Fit Summary

The “Model Fit Summary” table contains the following information:

- dependent variable name
- number of observations used
- data table name
- name of the spatial weights data table (specified by the WMAT= option)
- spatial ID variable
- type of model that was fit

- log-likelihood value at solution
- maximum absolute gradient at solution
- number of iterations
- AIC value at the solution (a smaller value indicates a better fit)
- SBC value at the solution (a smaller value indicates a better fit)

Below the “Model Fit Summary” table is a statement about whether the algorithm successfully converged.

Parameter Estimates

The “Parameter Estimates” table displays the estimates of the model parameters. In the SAR model, estimates are also displayed for the spatial autoregressive coefficient ρ and the variance of the error terms σ^2 . For the SEMs and SDEMs, estimates are displayed for the spatial autocorrelation coefficient λ and the variance of the error terms σ^2 . For the linear and SLX models, estimates are displayed for the variance of the error terms σ^2 .

The internal name of the spatial autoregressive coefficient ρ in the SDMs and SAR models is “_rho”. The t statistic that is displayed for “_rho” is a test of the spatial autoregressive coefficient. In addition, “_lambda” is the internal name of the spatial autocorrelation coefficient λ in the SEMs and SDEMs, and “_sigma2” is the internal name of the variance parameter σ^2 .

Covariance of Parameter Estimates

If you specify the COVB option in the MODEL or PROC CSPATIALREG statement, the CSPATIALREG procedure displays the estimated covariance matrix, defined as the inverse of the information matrix, that is evaluated at the final iteration.

Correlation of Parameter Estimates

If you specify the CORRB option in the MODEL or PROC CSPATIALREG statement, the CSPATIALREG procedure displays the estimated correlation matrix. It is based on the Hessian matrix that is used in the final iteration.

OUTPUT OUT= Data Table

The OUTPUT statement creates a new data table that contains various estimates that you specify. You can request that the output data table contain the estimates of $\mathbf{x}_i' \boldsymbol{\beta}$, the expected value of the response variable, and the residual.

Because of potential space limitations on the client workstation, the data table that the OUTPUT statement creates does not contain the variables in the input data table.

ODS Table Names

PROC CSPATIALREG assigns a name to each table that it creates. You can use these names to denote the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in [Table 11.3](#).

Table 11.3 ODS Tables Produced by PROC CSPATIALREG

Table Name	Description	Statement	Option
ClassInfo	Level information from the CLASS statement	CLASS	Default
ConvergenceStatus	Convergence status	MODEL	Default
CorrB	Correlation of parameter estimates	MODEL	CORRB
CovB	Covariance of parameter estimates	MODEL	COVB
FitModelSummary	Summary of nonlinear estimation	MODEL	Default
ParameterEstimates	Parameter estimates	MODEL	Default
TestResults	Hypothesis test statistics	TEST	Default
TimingDetails	Detailed summary of time used for all phases of execution	PROC CSPATIALREG	PRINTTIMING=(DETAILS)
TimingSummary	Summary of time used for main phases of execution	PROC CSPATIALREG	PRINTTIMING
ImpactEstimates	Summary of direct, indirect, and total impact estimates	PROC CSPATIALREG	IMPACT

Examples: CSPATIALREG Procedure

Example 11.1: Columbus Crime Data

Data Description and Objective

The data set CrimeOH contains data from Columbus, Ohio, about numbers of crimes (including residential burglaries and vehicle thefts) and possible determinants of these crimes. This data set comes from Anselin (1988).

The variable Crime represents the number of crimes in 49 Columbus neighborhoods. Additional variables in the data set that you want to evaluate as determinants of these crimes include Income (household income by \$1,000) and HValue (housing value by \$1,000). Summary statistics for these variables are computed by the following statements and presented in [Output 11.1.1](#):

```
proc means data=mycas.CrimeOH;
    var crime income hvalue;
run;
```

Output 11.1.1 Summary Statistics**The MEANS Procedure**

Variable	N	Mean	Std Dev	Minimum	Maximum
crime	49	35.1288367	16.7320385	0.1780000	68.8920000
income	49	14.3749388	5.7033781	4.4770000	31.0700000
hvalue	49	38.4362245	18.4660693	17.9000000	96.4000000

The spatial relationships among the 49 neighborhoods are summarized using the first-order neighbor contiguity matrix, contained in the CrimeWmat data set. This data set also comes from Anselin (1988).

Spatial Autoregressive (SAR) Model

The following statements fit a spatial autoregressive (SAR) model to the data by using the regressors Income and HValue:

```

/*-- SAR --*/
proc cspatialreg data=mycas.CrimeOH wmat=mycas.CrimeWmat NONORMALIZE;
    model crime=income hvalue / type=SAR;
    spatialid sid;
run;

```

In this example, the TYPE=SAR option in the MODEL statement specifies a SAR model. The NONORMALIZE option indicates that the spatial weights data table mycas.CrimeWmat should be used “as is” rather than be row-standardized. The parameter estimates for this model are shown in [Output 11.1.2](#). According to the results, the spatial autoregressive coefficient ρ is positive and significant at the 0.05 level based on the t-statistic. This indicates a positive spatial dependence in the data.

Output 11.1.2 Parameter Estimates of SAR Model**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	45.077070	7.870590	5.73	<.0001
income	1	-1.031531	0.328403	-3.14	0.0017
hvalue	1	-0.265924	0.088218	-3.01	0.0026
_rho	1	0.431020	0.123594	3.49	0.0005
_sigma2	1	95.487066	19.506312	4.90	<.0001

Spatial Durbin Model (SDM)

To fit a spatial Durbin model (SDM), you specify the SPATIALEFFECTS statement and specify the TYPE=SAR option in the MODEL statement or the PROC CSPATIALREG statement. In this example, the spatial lags of the regressors Income and HValue are considered in the SDM.

The following statements fit an SDM to the CrimeOH data:

```

/*-- SDM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SAR;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

The parameter estimates are shown in [Output 11.1.3](#). As in the SAR model, the spatial autoregressive coefficient ρ in the SDM is positive and significant at the 0.05 level based on the t-statistic, indicating a positive spatial dependence in the data.

Output 11.1.3 Parameter Estimates of SDM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	42.803457	13.924487	3.07	0.0021
income	1	-0.914206	0.336439	-2.72	0.0066
hvalue	1	-0.293745	0.088857	-3.31	0.0009
W_income	1	-0.519640	0.594772	-0.87	0.3823
W_hvalue	1	0.245716	0.176854	1.39	0.1647
_rho	1	0.426492	0.167492	2.55	0.0109
_sigma2	1	91.779519	18.909222	4.85	<.0001

To avoid potential collinearity with the intercept term in the MODEL statement, the SPATIALEFFECTS statement always excludes the intercept term. This means that only variables that you explicitly specify in the SPATIALEFFECTS statement are used to construct spatial lag of covariate effects.

Spatial Error Model (SEM)

To fit a spatial error model (SEM), specify the TYPE=SEM option in the MODEL statement or the PROC CSPATIALREG statement.

The following statements fit an SEM to the CrimeOH data:

```

/*-- SEM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SEM;
  spatialid sid;
  restrict _lambda<0.99;
run;

```

The parameter estimates are shown in [Output 11.1.4](#). According to this output, the p -value for the spatial autocorrelation parameter λ is 0.0002 based on the t-statistic. The results indicate a significant positive dependence in the error term.

Output 11.1.4 Parameter Estimates of SEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.891839	5.884056	10.18	<.0001
income	1	-0.941297	0.370263	-2.54	0.0110
hvalue	1	-0.302253	0.090551	-3.34	0.0008
_lambda	1	0.561785	0.152411	3.69	0.0002
_sigma2	1	95.570140	20.036592	4.77	<.0001

Spatial Durbin Error Model (SDEM)

To fit a spatial Durbin error model (SDEM), use the SPATIALEFFECTS statement and specify the TYPE=SEM option in the MODEL statement or the PROC CSPATIALREG statement. In this example, the spatial lags of the regressors Income and HValue are considered in the SDEM.

The following statements fit an SDEM to the CrimeOH data:

```

/*-- SDEM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SEM;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

The parameter estimates are shown in [Output 11.1.5](#).

Output 11.1.5 Parameter Estimates of SDEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	73.540584	8.860968	8.30	<.0001
income	1	-1.051699	0.322436	-3.26	0.0011
hvalue	1	-0.275607	0.091154	-3.02	0.0025
W_income	1	-1.156553	0.592915	-1.95	0.0511
W_hvalue	1	0.111754	0.202366	0.55	0.5808
_lambda	1	0.425397	0.173831	2.45	0.0144
_sigma2	1	92.533614	19.090022	4.85	<.0001

Spatial Moving Average (SMA) Model

To fit an SMA model, specify the TYPE=SMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit an SMA model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SMA;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 11.1.6](#).

Output 11.1.6 Parameter Estimates of SMA Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.252971	5.934861	9.98	<.0001
income	1	-0.921806	0.363482	-2.54	0.0112
hvalue	1	-0.287393	0.086880	-3.31	0.0009
_lambda	1	-0.799089	0.277861	-2.88	0.0040
_sigma2	1	117.731990	26.373322	4.46	<.0001

Spatial Durbin Moving Average (SDMA) Model

To fit an SDMA model, use the SPATIALEFFECTS statement and specify the TYPE=SMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lags of the regressors INCOME and HVALUE and fit an SDMA model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SMA;
  spatialeffects income hvalue;
  spatialid sid;
run;
```

Partial output is shown in [Output 11.1.7](#).

Output 11.1.7 Parameter Estimates of an SDMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	73.944211	9.083977	8.14	<.0001
income	1	-1.065635	0.312045	-3.42	0.0006
hvalue	1	-0.266840	0.092400	-2.89	0.0039
W_income	1	-1.074757	0.584955	-1.84	0.0662
W_hvalue	1	0.067568	0.209867	0.32	0.7475
_lambda	1	-0.642124	0.296638	-2.16	0.0304
_sigma2	1	103.502516	22.487027	4.60	<.0001

Spatial Autoregressive Confused (SAC) Model

To fit an SAC model, specify the TYPE=SAC option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit the SAC model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SAC;
  spatialid sid;
  restrict _lambda<0.99;
run;
```

The parameter estimates are shown in [Output 11.1.8](#).

Output 11.1.8 Parameter Estimates of SAC Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	47.778869	9.278442	5.15	<.0001
income	1	-1.025839	0.334006	-3.07	0.0021
hvalue	1	-0.281636	0.093366	-3.02	0.0026
_rho	1	0.368144	0.181118	2.03	0.0421
_lambda	1	0.166525	0.298114	0.56	0.5764
_sigma2	1	95.597489	19.474422	4.91	<.0001

Spatial Durbin Autoregressive Confused (SDAC) Model

To fit an SDAC model, use the SPATIALEFFECTS statement and specify the TYPE=SAC option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lags of the regressors INCOME and HVALUE and fit an SDAC model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SAC;
  spatialeffects income hvalue;
  spatialid sid;
  restrict _lambda<0.99;
run;
```

The parameter estimates are shown in [Output 11.1.9](#).

Output 11.1.9 Parameter Estimates of SDAC Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	50.827256	31.089621	1.63	0.1021
income	1	-0.950352	0.353961	-2.68	0.0073
hvalue	1	-0.286559	0.091261	-3.14	0.0017
W_income	1	-0.690471	0.839980	-0.82	0.4111
W_hvalue	1	0.208936	0.222585	0.94	0.3479
_rho	1	0.316760	0.414771	0.76	0.4450
_lambda	1	0.152884	0.475512	0.32	0.7478
_sigma2	1	93.133958	19.187743	4.85	<.0001

Spatial Autoregressive Moving Average (SARMA) Model

To fit a SARMA model, specify the TYPE=SARMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements fit a SARMA model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SARMA;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 11.1.10](#).

Output 11.1.10 Parameter Estimates of SARMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	48.973247	9.602039	5.10	<.0001
income	1	-1.016359	0.337215	-3.01	0.0026
hvalue	1	-0.287458	0.093079	-3.09	0.0020
_rho	1	0.336281	0.204317	1.65	0.0998
_lambda	1	-0.271945	0.426840	-0.64	0.5241
_sigma2	1	97.992936	21.253768	4.61	<.0001

Spatial Durbin Autoregressive Moving Average (SDARMA) Model

To fit an SDARMA model, use the SPATIALEFFECTS statement and specify the TYPE=SARMA option in the PROC CSPATIALREG statement or the MODEL statement.

The following statements consider the spatial lag of the regressor HVALUE and fit an SDARMA model to the CRIMEOH data:

```
proc cspatialreg data=mycas.CrimeOH wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=SARMA;
  spatialeffects hvalue;
  spatialid sid;
run;
```

The parameter estimates are shown in [Output 11.1.11](#).

Output 11.1.11 Parameter Estimates of SDARMA Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	30.280526	16.779263	1.80	0.0711
income	1	-0.960400	0.332845	-2.89	0.0039
hvalue	1	-0.293353	0.090767	-3.23	0.0012
W_hvalue	1	0.239748	0.206784	1.16	0.2463
_rho	1	0.590709	0.234325	2.52	0.0117
_lambda	1	0.191790	0.455187	0.42	0.6735
_sigma2	1	89.039641	18.819474	4.73	<.0001

Linear Regression Model

To fit a linear model, specify the TYPE=LINEAR option in the MODEL statement or the PROC CSPATIAL-REG statement.

The following statements fit a linear model to the CrimeOH data:

```
/*-- LINEAR --*/
proc cspatialreg data=mycas.CrimeOH;
  model crime=income hvalue / type=LINEAR;
run;
```

Partial output is shown in [Output 11.1.12](#).

Output 11.1.12 Parameter Estimates of Linear Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	68.618863	4.588210	14.96	<.0001
income	1	-1.597304	0.323739	-4.93	<.0001
hvalue	1	-0.273931	0.099989	-2.74	0.0062
_sigma2	1	122.751696	24.799493	4.95	<.0001

Spatial Lag of X (SLX) Model

To fit a spatial lag of X (SLX) model, use the SPATIALEFFECTS statement and specify the TYPE=LINEAR option in the MODEL statement or the PROC CSPATIALREG statement.

The following statements consider the spatial lags of the regressors Income and HValue and fit an SLX model to the CrimeOH data:

```

/*-- SLX --*/
proc cspatialreg data=mycas.CrimeOH wmat=mycas.CrimeWmat NONORMALIZE;
  model crime=income hvalue / type=LINEAR;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

The parameter estimates are shown in [Output 11.1.13](#).

Output 11.1.13 Parameter Estimates of SLX Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	75.028184	6.279950	11.95	<.0001
income	1	-1.109020	0.354232	-3.13	0.0017
hvalue	1	-0.289734	0.096058	-3.02	0.0026
W_income	1	-1.370866	0.531889	-2.58	0.0100
W_hvalue	1	0.191785	0.189841	1.01	0.3124
_sigma2	1	107.292373	21.676329	4.95	<.0001

Example 11.2: Simulated Data Example

Data Description and Objective

This example uses two simulated data sets, SimData and SimW, to illustrate model fitting in PROC CSPATIALREG and to facilitate subsequent discussion.

The SimData data set contains 50 observations and five variables. The variable SID identifies each spatial unit in the data. The three explanatory variables are x1, x2, and x3. The dependent variable is y. The SimW data set defines the spatial contiguity for all 50 spatial units. The first column, SID, in the SimW data set identifies each spatial unit. The remaining entries in the SimW data set are binary and define whether two spatial units are neighbors. A value of 1 indicates that two spatial units are neighbors, and 0 indicates otherwise.

Summary statistics for all variables except SID in the SimData data set are computed by the following statements and presented in [Output 11.2.1](#):

```
proc means data=mycas.SimData;
  var x1 x2 x3 y;
run;
```

Output 11.2.1 Summary Statistics

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
x1	50	-0.0076329	1.0989504	-2.4523193	1.6539456
x2	50	-0.0829941	0.9671181	-2.5725767	2.4034547
x3	50	-0.0894387	0.9975304	-2.4470049	2.6720533
y	50	1.1569199	1.5687060	-1.9399423	4.7136835

The following statements fit a spatial autoregressive (SAR) model to the data by using three regressors, x1, x2, and x3:

```
/*-- SAR --*/
proc cspatialreg data=mycas.SimData wmat=mycas.SimW;
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.2.2](#).

Output 11.2.2 Parameter Estimates of SAR Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

To fit a spatial Durbin model (SDM) that includes exogenous interaction effects of x_1 , x_2 , and x_3 , submit the following statements:

```
/*-- SDM --*/
proc cspatialreg data=mycas.SimData wmat=mycas.SimW;
  model y=x1-x3 / type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.2.3](#).

Output 11.2.3 Parameter Estimates of SDM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

If you want to fit another type of model, you need to change the value of the TYPE= option in the MODEL statement. As an example, to fit a spatial error model (SEM) instead of a SAR model to the data, you can use the following statements:

```
/*-- SEM --*/
proc cspatialreg data=mycas.SimData wmat=mycas.SimW;
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.2.4](#).

Output 11.2.4 Parameter Estimates of SEM
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166289	0.029514	39.52	<.0001
x1	1	0.487975	0.049086	9.94	<.0001
x2	1	0.634442	0.061776	10.27	<.0001
x3	1	-0.831250	0.054780	-15.17	<.0001
_lambda	1	-0.964826	0.132514	-7.28	<.0001
_sigma2	1	0.147434	0.031318	4.71	<.0001

The following statements fit a spatial autoregressive confused (SAC) model and spatial autoregressive moving average (SARMA) model to the data:

```

/*-- SAC --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW;
  model y=x1-x3 / type=SAC;
  spatialid SID;
run;

/*-- SARMA --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW;
  model y=x1-x3 / type=SARMA;
  spatialid SID;
run;

```

The parameter estimates for the SAC and SARMA models are shown in [Output 11.2.5](#) and [Output 11.2.6](#), respectively.

Output 11.2.5 Parameter Estimates of an SAC Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.745140	0.108969	16.01	<.0001
x1	1	0.574630	0.047481	12.10	<.0001
x2	1	0.699848	0.057547	12.16	<.0001
x3	1	-0.909416	0.053365	-17.04	<.0001
_rho	1	-0.449744	0.075425	-5.96	<.0001
_lambda	1	-0.235971	0.277450	-0.85	0.3950
_sigma2	1	0.128845	0.025938	4.97	<.0001

Output 11.2.6 Parameter Estimates of a SARMA Model

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.721937	0.111720	15.41	<.0001
x1	1	0.576752	0.047432	12.16	<.0001
x2	1	0.694100	0.058118	11.94	<.0001
x3	1	-0.910829	0.053061	-17.17	<.0001
_rho	1	-0.434840	0.079844	-5.45	<.0001
_lambda	1	0.393432	0.342912	1.15	0.2512
_sigma2	1	0.133287	0.027984	4.76	<.0001

Example 11.3: Compact Representation of a Spatial Weights Matrix

When a spatial weights matrix is sparse, you might want to provide its compact representation rather than the full matrix to the CSPATIALREG procedure. This example shows you how to use the compact representation of a spatial weights matrix in PROC CSPATIALREG. For illustration, the simulated data sets SimData and SimW from “[Example 11.2: Simulated Data Example](#)” on page 533 are used here. The compact representation of the spatial weights matrix in the SimW data set is created and saved in the data set SimW_Compact.

The first 10 observations in the SimW_Compact data set are shown in [Figure 11.3.1](#).

Output 11.3.1 SimW_Compact Data Set

Obs	SID	cSID	Value
1	L50	L45	1
2	L32	L35	1
3	L50	L25	1
4	L45	L36	1
5	L10	L8	1
6	L44	L48	1
7	L7	L6	1
8	L36	L15	1
9	L5	L19	1
10	L37	L46	1

To fit a spatial autoregressive (SAR) model, you can use the following statements:

```
/*-- SAR --*/
proc cspatialreg data=mycas.SimData wmat=mycas.SimW_Compact;
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.3.2](#).

Output 11.3.2 Parameter Estimates of SAR Model with Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

To fit a spatial error model (SEM) instead of a SAR model to the data, you can use the following statements:

```
/*-- SEM --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact;
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.3.3](#).

Output 11.3.3 Parameter Estimates of SEM with Compact Representation

The CSPATIALREG Procedure

Parameter Estimates						
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t	
Intercept	1	1.166289	0.029514	39.52	<.0001	
x1	1	0.487975	0.049086	9.94	<.0001	
x2	1	0.634442	0.061776	10.27	<.0001	
x3	1	-0.831250	0.054780	-15.17	<.0001	
_lambda	1	-0.964826	0.132514	-7.28	<.0001	
_sigma2	1	0.147434	0.031318	4.71	<.0001	

If you want to fit another type of model, you need to change the value of the TYPE= option in the MODEL statement. For example, to fit a spatial autoregressive moving average (SARMA) model and a spatial autoregressive confused (SAC) model to the data, you can use the following statements:

```
/*-- SARMA --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact;
  model y=x1-x3 / type=SARMA;
  spatialid SID;
run;

/*-- SAC --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact;
  model y=x1-x3 / type=SAC;
  spatialid SID;
run;
```

The parameter estimates for the SARMA and SAC models are shown in [Output 11.3.4](#) and [Output 11.3.5](#), respectively.

Output 11.3.4 Parameter Estimates of a SARMA Model with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.721937	0.111720	15.41	<.0001
x1	1	0.576752	0.047432	12.16	<.0001
x2	1	0.694100	0.058118	11.94	<.0001
x3	1	-0.910829	0.053061	-17.17	<.0001
_rho	1	-0.434840	0.079844	-5.45	<.0001
_lambda	1	0.393432	0.342912	1.15	0.2512
_sigma2	1	0.133287	0.027984	4.76	<.0001

Output 11.3.5 Parameter Estimates of an SAC Model with Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.745140	0.108969	16.01	<.0001
x1	1	0.574630	0.047481	12.10	<.0001
x2	1	0.699848	0.057547	12.16	<.0001
x3	1	-0.909416	0.053365	-17.04	<.0001
_rho	1	-0.449744	0.075425	-5.96	<.0001
_lambda	1	-0.235971	0.277450	-0.85	0.3950
_sigma2	1	0.128845	0.025938	4.97	<.0001

Example 11.4: Taylor and Chebyshev Approximations

When you have a large data set (that is, the number of spatial units in your data is large), it becomes burdensome to fit some models. This is partly because all models except linear regression models require you to calculate the determinant of the matrix of a large size (such as $|\mathbf{I} - \rho\mathbf{W}|$ in a spatial autoregressive model). In these cases, using Taylor and Chebyshev approximations in the CSPATIALREG procedure can be helpful. PROC CSPATIALREG enables you to estimate spatial autoregressive, spatial Durbin, spatial error, and spatial Durbin error models that have a relatively large spatial weights matrix by using these two approximations. By using the two small data sets SimData and SimW from “[Example 11.2: Simulated Data Example](#)” on page 533, examples in this section show how you can invoke the two approximations in PROC CSPATIALREG.

The following statements fit a spatial autoregressive (SAR) model by using Chebyshev approximation:

```

/*-- SAR Model : full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(ORDER=10);
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;

```

You use the APPROXIMATION= option to specify which approximation technique to use. By default, Chebyshev approximation is used. The ORDER=10 option specifies a 10th-order Chebyshev polynomial to be used for approximation. The parameter estimates for this model are shown in [Output 11.4.1](#). Note that the spatial weights matrix in the SimW data set is a full matrix. Compared with [Output 11.2.2](#), Chebyshev approximation yields very similar parameter estimates.

Output 11.4.1 Parameter Estimates of SAR Model with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

Using the compact representation of the spatial weights matrix, you can submit the following statements to fit a SAR model by using Chebyshev approximation:

```
/*-- SAR Model : compact form of W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3 / type=SAR;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.4.2](#), which is identical to [Output 11.4.1](#).

Output 11.4.2 Parameter Estimates of SAR Model with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.780650	0.098703	18.04	<.0001
x1	1	0.573329	0.047395	12.10	<.0001
x2	1	0.707048	0.057181	12.37	<.0001
x3	1	-0.902843	0.053314	-16.93	<.0001
_rho	1	-0.473713	0.063008	-7.52	<.0001
_sigma2	1	0.131509	0.026350	4.99	<.0001

The following statements fit a spatial Durbin model (SDM) by using Taylor approximation. You specify the keyword TAYLOR in the APPROXIMATION= option to request Taylor approximation. In particular, you request a 50th-order Taylor polynomial to be used for approximation by specifying the ORDER=50 option.

```

/*-- SDM : Taylor approximation with full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 11.4.3](#). Compared with [Output 11.2.3](#), the SDM that is fit using Taylor approximation yields almost identical parameter estimates.

Output 11.4.3 Parameter Estimates of SDM with Taylor Approximation
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

With the compact representation, the following statements fit the SDM by using Taylor approximation:

```

/*-- SDM : Taylor approximation with compact W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact
  approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 11.4.4](#), which is identical to [Output 11.4.3](#).

Output 11.4.4 Parameter Estimates of SDM with Taylor Approximation and Compact Representation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932578	0.198882	9.72	<.0001
x1	1	0.548504	0.049806	11.01	<.0001
x2	1	0.686011	0.056266	12.19	<.0001
x3	1	-0.890161	0.053516	-16.63	<.0001
W_x1	1	0.172302	0.154018	1.12	0.2633
W_x2	1	0.023746	0.198557	0.12	0.9048
W_x3	1	-0.324808	0.228032	-1.42	0.1543
_rho	1	-0.639757	0.164651	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

To use Chebyshev approximation for the preceding SDM, submit the following statements:

```

/*-- SDM : Chebyshev approximation with full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(ORDER=10);
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 11.4.5](#), which is similar to [Output 11.4.3](#).

Output 11.4.5 Parameter Estimates of SDM with Chebyshev Approximation**The CSPATIALREG Procedure**

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932574	0.198876	9.72	<.0001
x1	1	0.548505	0.049806	11.01	<.0001
x2	1	0.686012	0.056266	12.19	<.0001
x3	1	-0.890162	0.053516	-16.63	<.0001
W_x1	1	0.172300	0.154015	1.12	0.2633
W_x2	1	0.023744	0.198554	0.12	0.9048
W_x3	1	-0.324805	0.228027	-1.42	0.1543
_rho	1	-0.639755	0.164646	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

To use Chebyshev approximation for this model with compact representation, submit the following statements:

```

/*-- SDM : Chebyshev approximation with compact W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3/ type=SAR;
  spatialeffects x1-x3;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 11.4.6](#).

Output 11.4.6 Parameter Estimates of SDM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.932574	0.198876	9.72	<.0001
x1	1	0.548505	0.049806	11.01	<.0001
x2	1	0.686012	0.056266	12.19	<.0001
x3	1	-0.890162	0.053516	-16.63	<.0001
W_x1	1	0.172300	0.154015	1.12	0.2633
W_x2	1	0.023744	0.198554	0.12	0.9048
W_x3	1	-0.324805	0.228027	-1.42	0.1543
_rho	1	-0.639755	0.164646	-3.89	0.0001
_sigma2	1	0.120527	0.024729	4.87	<.0001

The following statements fit a spatial error model (SEM) by using Chebyshev approximation:

```

/*-- SEM : full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(ORDER=10);
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;

```

The parameter estimates for this model are shown in [Output 11.4.7](#). Note that the spatial weights matrix in the mycas.SimW data table is a full matrix. Compared with [Output 11.2.4](#), Chebyshev approximation yields similar parameter estimates.

Output 11.4.7 Parameter Estimates of SEM with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166221	0.029666	39.31	<.0001
x1	1	0.488222	0.049184	9.93	<.0001
x2	1	0.635419	0.061432	10.34	<.0001
x3	1	-0.831401	0.054913	-15.14	<.0001
_lambda	1	-0.957363	0.117021	-8.18	<.0001
_sigma2	1	0.148034	0.031092	4.76	<.0001

Using the compact representation of the spatial weights matrix, you can submit the following statements to fit an SEM by using Chebyshev approximation:

```

/*-- SEM : compact form of W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3 / type=SEM;
  spatialid SID;
run;

```


The parameter estimates for this model are shown in [Output 11.4.8](#), which is almost identical to [Output 11.4.7](#).

Output 11.4.8 Parameter Estimates of SEM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.166221	0.029666	39.31	<.0001
x1	1	0.488222	0.049184	9.93	<.0001
x2	1	0.635419	0.061432	10.34	<.0001
x3	1	-0.831401	0.054913	-15.14	<.0001
_lambda	1	-0.957363	0.117021	-8.18	<.0001
_sigma2	1	0.148034	0.031092	4.76	<.0001

The following statements fit a spatial Durbin error model (SDEM) by using Taylor approximation:

```
/*-- SDEM : Taylor approximation with full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.4.9](#).

Output 11.4.9 Parameter Estimates of SDEM with Taylor Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.193178	0.047129	25.32	<.0001
x1	1	0.566497	0.056782	9.98	<.0001
x2	1	0.722060	0.059717	12.09	<.0001
x3	1	-0.909212	0.061286	-14.84	<.0001
W_x1	1	-0.135284	0.128339	-1.05	0.2918
W_x2	1	-0.418116	0.152899	-2.73	0.0062
W_x3	1	0.290041	0.192397	1.51	0.1317
_lambda	1	-0.723853	0.209844	-3.45	0.0006
_sigma2	1	0.126723	0.026820	4.73	<.0001

With the compact representation, the following statements fit the SDEM by using Taylor approximation:

```
/*-- SDEM : Taylor approximation with compact W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact
  approximation=(Taylor ORDER=50);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.4.10](#), which is identical to [Output 11.4.9](#).

Output 11.4.10 Parameter Estimates of SDEM with Taylor Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.193178	0.047129	25.32	<.0001
x1	1	0.566497	0.056782	9.98	<.0001
x2	1	0.722060	0.059717	12.09	<.0001
x3	1	-0.909212	0.061286	-14.84	<.0001
W_x1	1	-0.135284	0.128339	-1.05	0.2918
W_x2	1	-0.418116	0.152899	-2.73	0.0062
W_x3	1	0.290041	0.192397	1.51	0.1317
_lambda	1	-0.723853	0.209844	-3.45	0.0006
_sigma2	1	0.126723	0.026820	4.73	<.0001

To use Chebyshev approximation to fit the preceding SDEM, submit the following statements:

```
/*-- SDEM : Chebyshev approximation with full W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW approximation=(ORDER=10);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
  spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.4.11](#), which is similar to [Output 11.4.9](#).

Output 11.4.11 Parameter Estimates of SDEM with Chebyshev Approximation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.193178	0.047129	25.32	<.0001
x1	1	0.566500	0.056780	9.98	<.0001
x2	1	0.722062	0.059716	12.09	<.0001
x3	1	-0.909213	0.061285	-14.84	<.0001
W_x1	1	-0.135290	0.128335	-1.05	0.2918
W_x2	1	-0.418122	0.152895	-2.73	0.0062
W_x3	1	0.290046	0.192393	1.51	0.1317
_lambda	1	-0.723821	0.209774	-3.45	0.0006
_sigma2	1	0.126724	0.026819	4.73	<.0001

To use Chebyshev approximation for this model with compact representation, submit the following statements:

```
/*-- SDEM : Chebyshev approximation with compact W --*/
proc cspatialreg data=mycas.SimData Wmat=mycas.SimW_Compact approximation=(ORDER=10);
  model y=x1-x3/ type=SEM;
  spatialeffects x1-x3;
```

```
spatialid SID;
run;
```

The parameter estimates for this model are shown in [Output 11.4.12](#).

Output 11.4.12 Parameter Estimates of SDEM with Chebyshev Approximation and Compact Representation

The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	1.193178	0.047129	25.32	<.0001
x1	1	0.566500	0.056780	9.98	<.0001
x2	1	0.722062	0.059716	12.09	<.0001
x3	1	-0.909213	0.061285	-14.84	<.0001
W_x1	1	-0.135290	0.128335	-1.05	0.2918
W_x2	1	-0.418122	0.152895	-2.73	0.0062
W_x3	1	0.290046	0.192393	1.51	0.1317
_lambda	1	-0.723821	0.209774	-3.45	0.0006
_sigma2	1	0.126724	0.026819	4.73	<.0001

Example 11.5: Impact Estimation

Data Description and Objective

Examples in this section use the CrimeOH and CrimeWmat data sets that are described in “[Example 11.1: Columbus Crime Data](#)” on page 524 to demonstrate impact estimation in PROC CSPATIALREG.

Spatial Lag of X (SLX) Model

The following statements fit a spatial lag of X (SLX) model to the data by using Income and HValue as explanatory variables, and they estimate the average direct impacts, the average indirect impacts, and the average total impacts for these two explanatory variables.

```
/*-- SLX --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat impact(nmc=1000);
  model crime=income hvalue / type=LINEAR;
  spatialeffects income hvalue;
  spatialid sid;
run;
```

In this example, the TYPE=LINEAR option in the MODEL statement and SPATIALEFFECTS statement specify an SLX model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. The impact estimates for this model are shown in [Output 11.5.2](#). According to [Output 11.5.2](#), Income has a direct impact of -1.11 and an indirect impact of -1.37 on Crime, which equal the coefficient estimate of Income and the spatial lag of Income, respectively, in [Output 11.5.1](#). Similarly, HValue has a direct impact of -0.29 and an indirect impact of 0.19 on Crime, which equal the coefficient estimate of HValue and the spatial lag of HValue, respectively, in [Output 11.5.1](#).

Output 11.5.1 Parameter Estimates of the SLX Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	75.028374	6.278808	11.95	<.0001
income	1	-1.108925	0.354227	-3.13	0.0017
hvalue	1	-0.289729	0.096056	-3.02	0.0026
W_income	1	-1.370974	0.531869	-2.58	0.0099
W_hvalue	1	0.191771	0.189837	1.01	0.3124
_sigma2	1	107.287543	21.675353	4.95	<.0001

Output 11.5.2 Impact Estimates of Regressors in the SLX Model

Impact Estimates Summary									
Parameter	Direct			Indirect			Total		
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value
income	-1.108925	0.354227	-3.13	-1.370974	0.531869	-2.58	-2.479899	0.470529	-5.27
hvalue	-0.289729	0.096056	-3.02	0.191771	0.189837	1.01	-0.097958	0.192176	-0.51

Spatial Error Model (SEM)

The following statements fit an SEM model to the data by using Income and HValue as two explanatory variables, and they estimate impacts for these two variables:

```

/*-- SEM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat impact(nmc=10000);
    model crime=income hvalue / type=SEM;
    spatialid sid;
run;

```

In this example, the TYPE=SEM option in the MODEL statement specifies an SEM model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. The impact estimates for this model are shown in [Output 11.5.4](#). According to the results, both Income and HValue have direct impacts but not indirect impacts on the dependent variable Crime. Moreover, the direct impacts of Income and HValue equal their coefficient estimates in [Output 11.5.3](#).

Output 11.5.3 Parameter Estimates of the SEM Model
The CSPATIALREG Procedure

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	59.893276	5.883689	10.18	<.0001
income	1	-0.941312	0.370277	-2.54	0.0110
hvalue	1	-0.302252	0.090553	-3.34	0.0008
_lambda	1	0.561786	0.152423	3.69	0.0002
_sigma2	1	95.574411	20.038002	4.77	<.0001

Output 11.5.4 Impact Estimates of Regressors in the SEM Model

Impact Estimates Summary								
Parameter	Direct			Indirect			Total	
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation
income	-0.941312	0.370277	-2.54	0	0	.	-0.941312	0.370277
hvalue	-0.302252	0.090553	-3.34	0	0	.	-0.302252	0.090553

Spatial Autoregressive (SAR) Model

The following statements fit a SAR model to the data by using the explanatory variables Income and HValue, and they request that the average direct impacts, the average indirect impacts, and the average total impacts be computed for the two explanatory variables:

```

/*-- SAR --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat
  impact(nmc=50000 order=100);
  model crime=income hvalue / type=SAR;
  spatialid sid;
run;

```

The impact estimates for this model are shown in [Output 11.5.5](#). On one hand, you see that Income exerts a negative direct and indirect impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher incomes. Moreover, increased Income in nearby neighborhoods has a negative impact on the number of crimes. On the other hand, HValue exerts a negative direct impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher housing values.

Output 11.5.5 Impact Estimates of Regressors in the SAR Model
The CSPATIALREG Procedure

Impact Estimates Summary								
Parameter	Direct			Indirect			Total	
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation
income	-1.091669	0.332260	-3.29	-0.751319	0.372889	-2.01	-1.842988	0.565408
hvalue	-0.282143	0.094550	-2.98	-0.209330	0.141523	-1.48	-0.491473	0.211251

Spatial Durbin Model (SDM)

The following statements fit an SDM model to the data by using the explanatory variables Income and HValue, and they estimate impacts for these two variables:

```

/*-- SDM --*/
proc cspatialreg data=mycas.CrimeOH Wmat=mycas.CrimeWmat
  impact(nmc=50000 order=100);
  model crime=income hvalue / type=SAR;
  spatialeffects income hvalue;
  spatialid sid;
run;

```

In this example, the TYPE=SAR option in the MODEL statement specifies a SAR model. The SPATIALEFFECTS statement requests that the spatial lag of explanatory variables Income and HValue be included in the model. The IMPACT option requests that the average direct impacts, the average indirect impacts, and the average total impacts be computed for explanatory variables in the model. To estimate impacts, the NMC= option specifies 50,000 iterations of Monte Carlo simulation and the ORDER= option specifies 100 as the order of Neumann series. The impact estimates for this model are shown in [Output 11.5.6](#). On one hand, you see that the indirect impact of Income is not significant at the 5% significance level (because the *t* value of -1.58 is greater than -1.96). However, the direct impact of Income is negative and significant, suggesting that you would see a decrease in the number of crimes in neighborhoods that have higher incomes. Similarly, HValue exerts a negative direct impact on the number of crimes. This suggests that you would see a decrease in the number of crimes in neighborhoods that have higher housing values.

Output 11.5.6 Impact Estimates of Regressors in the SDM Model
The CSPATIALREG Procedure

Impact Estimates Summary									
Parameter	Direct			Indirect			Total		
	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value	Mean	Standard Deviation	t Value
income	-1.021846	0.329139	-3.10	-1.493714	0.945050	-1.58	-2.515560	1.008252	-2.49
hvalue	-0.278730	0.093730	-2.97	0.201998	0.368789	0.55	-0.076733	0.406750	-0.19

References

- Anselin, L. (1988). *Spatial Econometrics: Methods and Models*. Amsterdam: Springer.
- Anselin, L. (2001). "Spatial Econometrics." In *A Companion to Theoretical Econometrics*, edited by B. H. Baltagi, 310–330. Oxford: Wiley-Blackwell.
- Elhorst, J. P. (2013). *Spatial Econometrics: From Cross-Sectional Data to Spatial Panels*. Berlin: Springer.
- LaMotte, L. R. (1994). "A Note on the Role of Independence in *t* Statistics Constructed from Linear Statistics in Regression Models." *American Statistician* 48:238–240.
- LeSage, J., and Pace, R. K. (2009). *Introduction to Spatial Econometrics*. Boca Raton, FL: CRC Press.

Chapter 12

The ECM Procedure

Contents

Overview: ECM Procedure	550
Using CAS Sessions and CAS Engine Librefs	552
Getting Started: ECM Procedure	553
Syntax: ECM Procedure	557
Functional Summary	557
PROC ECM Statement	559
DISPLAY Statement	564
DISPLAYOUT Statement	565
MARGINAL Statement	566
OUTPUT Statement	567
OUTSUM Statement	568
VAR Statement	570
Details: ECM Procedure	571
Overview of the Algorithm	571
Percentile Computation Methods	573
Input Specification	574
Output Data Tables	575
Displayed Output	576
Examples: ECM Procedure	579
Example 12.1: Developing an Economic Capital Model	579
References	595

Overview: ECM Procedure

The ECM procedure develops an economic capital model. A financial enterprise that incurs losses that are inherent to the nature of its business needs to estimate the extent of those losses across multiple business lines. Developing a model for the enterprise-wide losses enables such an enterprise not only to estimate the minimum capital it must set aside to cover the worst-case losses and meet regulatory requirements but also to assess the economic viability and improve the risk management abilities of the enterprise.

Although there is no one precise definition of economic capital, it is generally agreed that economic capital depends on the worst-case losses you expect to cover across your enterprise's business lines or risk categories. The statistical approach that is described and illustrated in this chapter can be a component of an internal model that is allowed by a regulatory framework. Contrast the approach in this chapter to the standard model, which makes some fixed assumptions and is relatively easier to compute. However, the capital requirements that stem from using the standard model often overestimate the risks and result in higher capital requirements. So, it might be prudent to invest time and resources in developing an internal model to arrive at more accurate estimates of risk, which often result in lower capital requirements and enable you to free up some capital for developing your business and increasing shareholder value. The regulatory agency should be willing to accept your internal model as long as it is statistically sound and has reasonable assumptions.

For the purposes of the ECM procedure, a model for the total loss an enterprise can incur as a result of losses across its multiple lines of business is called an economic capital model (ECM). The process of estimating an ECM is called economic capital modeling. A particularly powerful method of developing an economic capital model consists of the following steps:

1. Collect the loss event data from all business lines that incur financial losses. The loss event data consist of frequency (count) of losses that a business line incurs in a particular time period and the severity (magnitude) of each loss.
2. Estimate separate loss distribution models for the frequency and severity for each business line. Usually, the frequency model is a parametric model that consists of a discrete probability distribution, and the severity model is a parametric model that consists of a continuous probability distribution. Each model can contain regression parameters that measure the impact of external factors on the location or shape of the probability distribution.
3. Create a compound distribution model (CDM) of the aggregate loss that a business line can incur in the particular time period. This step requires combining the frequency and severity models. Because of the possibly complex nature of the frequency and severity distributions, the CDM often cannot be encoded in a concise parametric form. Hence, it is usually estimated by simulating a large empirical sample of the aggregate loss.

At this step in the process, the worst-case losses for an individual business line can be estimated by computing the value-at-risk (VaR) or tail value-at-risk (TVaR) from the large empirical sample of the CDM.

4. Estimate a loss dependency structure of the losses across all business lines. This dependency structure essentially estimates how a loss in one business line is correlated with the losses in other business lines. A typical method of estimating the dependency is to fit a copula model to the aggregate losses in each business line, where losses are matched by the time period of interest.

5. Use the estimated dependency structure to simulate a large sample of probabilities of loss in all business lines. Each observation in the sample essentially records the probability of seeing a loss in each of the business lines in the time period of interest. In other words, each observation records one possible scenario where all business lines simultaneously incur losses, the extent of which varies according to the simulated probabilities. The simulated probabilities account for the dependency among the business lines. Simulating a large number of such scenarios provides a comprehensive picture of enterprise-wide losses. However, these simulation data are on the probability scale, which creates the need for the next step of the process.
6. For each observation in the copula simulation sample, invert the probability estimate of each business line by using that line's aggregate loss sample, which the CDM simulation creates in the third step. This produces an estimate of the loss for each business line. Aggregating the losses across all business lines produces an estimate the total loss for that observation. Repeating this process for all the observations in the copula simulation table results in a large sample of the total loss, which essentially encodes the probability distribution model of the enterprise-wide losses.
7. Compute VaR and TVaR estimates for the total loss by using the large sample that the preceding step generates. These VaR and TVaR estimates help you decide the economic capital needs of your enterprise to cover worst-case losses and meet regulatory requirements.

SAS Econometrics offers various procedures to help you implement each modeling and simulation step of this process. The CNTSELECT and SEVSELECT procedures help you estimate a wide range of frequency and severity models, respectively. The CCDM procedure helps you estimate the compound distribution model (CDM) by simulating a large distributed sample of the aggregate loss for each business line. The CCOPULA procedure helps you fit and simulate various types of copula models.

The ECM procedure helps you implement the last two critical steps of the economic capital modeling process. It uses the large, distributed CDM samples to estimate the empirical distribution function (EDF) of each business line's aggregate loss, and uses those EDF estimates to efficiently invert the probabilities in the large, distributed copula simulation sample.

The losses in each business line often depend on the economic and social environment in which the business operates. It is important to estimate the frequency and severity models that account for such external factors. The dependence of frequency and severity on external factors implies that the CDM and the economic capital model (ECM) also depend on those factors. Hence, it is not sufficient to estimate the ECM for just one set of values of the external factors, where each set of values is called an external scenario. You should estimate the ECM for multiple external scenarios, each representing a different possible state of the world. Because the CDM and ECM modeling steps need to be repeated for each external scenario, it is important that those steps run as efficiently as possible. Not only should each step use all available computational resources, but it should also be able to efficiently consume the data that the previous step generates. In particular, when the data are large and distributed across a cluster of computing and storage nodes, the modeling steps should consume the data in their distributed format instead of bringing them to one central node purely for the modeling purposes.

The SAS Econometrics procedures are designed to help you achieve computational efficiency, minimal data movement, and modeling convenience. The procedures other than PROC ECM are described in their respective chapters. This chapter describes the syntax and features of PROC ECM.

PROC ECM requires SAS Cloud Analytic Services (CAS) in order to run. Because PROC ECM runs on CAS, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: ECM Procedure

This section outlines the use of the ECM procedure to estimate the distribution of the total loss across two dependent lines of business. The example is intended as a gentle introduction to the key functionality of PROC ECM.

The losses that a particular line of business (LoB) incurs often have a compound probability distribution that cannot be expressed in a closed form, so you need to approximate the distribution by simulating a large, empirical sample from that distribution. A common way to do that is to fit probability distribution models for the frequency and severity of losses (which can usually be expressed in parametric forms) and use those models to simulate a large sample of the aggregate loss. You would typically use the CNTSELECT procedure to estimate the frequency model, the SEVSELECT procedure to estimate the severity model, and the CCDDM procedure to combine those frequency and severity models to simulate an aggregate loss sample from the compound distribution. However, for the purpose of this example, let the following two DATA steps perform the aggregate loss simulations by assuming that the losses in one LoB follow a negative binomial frequency distribution and a gamma severity distribution, whereas the losses in the second LoB follow a Poisson frequency distribution and a lognormal severity distribution. This example assumes that the CAS engine libref is named mycas, as defined in the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 552, but you can substitute any appropriately named CAS engine libref.

```
/* Simulate a sample from the compound distribution of
   negative binomial frequency and gamma severity. */
data mycas.loss1sample(keep=loss1) / sessref=mysess;
  call streaminit(12);
  theta = 3.0; p = 0.5; * negative binomial parameters;
  gTheta = 1000; gAlpha = 2.5; * gamma parameters;
  do n = 1 to 10000;
    count = rand('NEGB',p,theta);
    loss1 = 0;
    do c=1 to count;
      loss1 = loss1 + rand('Gamma', gAlpha, gTheta);
    end;
    output;
  end;
run;

/* Simulate a sample from the compound distribution of
   Poisson frequency and lognormal severity. */
data mycas.loss2sample(keep=loss2) / sessref=mysess;
  call streaminit(135);
  lambda = 2; * Poisson parameters;
  mu = log(2500); sigma = 0.85; * lognormal parameters;
  do n = 1 to 10000;
    count = rand('POISSON',lambda);
    loss2 = 0;
    do c=1 to count;
      loss2 = loss2 + exp(mu) * rand('LOGNORMAL')**sigma;
    end;
    output;
  end;
run;
```

The preceding two DATA steps produce two large sample tables, `mycas.Loss1Sample` and `mycas.Loss2Sample`, for the two LoBs. Each data table represents an empirical sample for the aggregate loss distribution of the respective LoB. The use of the `SESSREF=` option ensures that each DATA step runs on the CAS server in parallel by using multiple threads of execution on all the worker nodes in order to create large samples quickly. When each of these DATA steps is run on a CAS session that has three worker nodes, such that each worker node uses 32 threads of execution, the sample size of each LoB is 960,000, because the loop inside the DATA step runs 10,000 times independently in each thread on each worker node.

The losses in the two LoBs are usually correlated, and you can estimate their dependency structure by fitting a copula model. In the context of copula modeling, loss in each LoB constitutes a marginal random variable, referred to simply as a marginal. For this example, assume that the dependency is captured by a Gaussian copula with a correlation matrix that is recorded in the data table `mycas.CorrTab`:

```
data mycas.corrTab;
    loss1 = 1.0; loss2 = 0.35; output;
    loss1 = 0.35; loss2 = 1.0; output;
run;
```

To simulate the total loss across both LoBs that takes the dependency structure into account, you need to first simulate the probabilities of a loss value that each LoB incurs in the same time period. The following PROC CCOPULA step simulates 500,000 such loss probability combinations and stores them in the data table `mycas.LossProb`:

```
proc ccopula;
    var loss1 loss2;
    define cop normal (corr=mycas.corrtab);
    simulate cop / ndraws=500000 seed=234
        outuniform=mycas.lossprob;
run;
```

Now, you need to perform the following steps to estimate the distribution of the total loss:

1. For each observation in the copula simulation table, you need to convert the loss probability of each LoB to its loss estimate by computing the quantile function for that LoB's probability distribution. When the cumulative distribution function (CDF) or the quantile function of a marginal distribution is not available in a closed form, you need to estimate the quantile by the corresponding percentile from an empirical sample of that marginal's distribution. For this example, you need to use the empirical samples for the two LoBs that are available in the data tables `mycas.Loss1Sample` and `mycas.Loss2Sample`. After computing each LoB's loss estimate as a percentile, you need to add those loss estimates to estimate the total loss for the current observation of the copula simulation table.
2. You need to perform the preceding step for all observations of the copula simulation table. This results in a sample of the total loss, which represents an empirical estimate of the probability distribution of the total loss. You need to analyze this sample to compute the estimates of various quantities of interest, especially the risk measures such as the value-at-risk (VaR) and tail value-at-risk (TVaR).

The ECM procedure is created precisely to help you implement the preceding steps by submitting a few simple statements and executing them very efficiently by utilizing all the computational resources of a CAS server. The following PROC ECM step implements the preceding steps by using the data table `mycas.LossProb` (which contains the copula simulations) and the data tables `mycas.Loss1Sample` and `mycas.Loss2Sample` (which contain the empirical samples of the two LoBs):

```
proc ecm data=mycas.lossprob seed=57;
  marginal loss1: data=mycas.loss1sample;
  marginal loss2: data=mycas.loss2sample;
  outsum pctlpts=default tvarpts=90 95 97.5 to 99.5 by 1;
  output out=mycas.tlossSample var=tloss;
run;
```

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “Using CAS Sessions and CAS Engine Librefs” on page 12 and “Loading a SAS Data Set onto a CAS Server” on page 13 in Chapter 3, “Shared Concepts.”

The DATA= data table must contain the loss probabilities. Use of the OUTUNIFORM= option in the preceding PROC CCOPULA step ensures that the data table mycas.LossProb contains copula simulations on a uniform probability scale.

The SEED= option in the PROC ECM statement ensures that results are similar across multiple runs of the procedure for identical inputs and identical session configuration. This is required because, by default, the algorithm that PROC ECM uses samples the body region of the distribution to reduce the data movement among the worker nodes and to speed up the computations.

The two MARGINAL statements specify the two marginal variables, one for each LoB. The name of the marginal variable, which you must specify first in each MARGINAL statement followed by a colon (:), must match the name of the corresponding variable in the copula simulation table. The DATA= option in each MARGINAL statement specifies the data table that contains the empirical sample of that marginal.

The OUTSUM statement specifies the statistics that you want to estimate for the distribution of the total loss. The PCTLPTS=DEFAULT option requests that the percentiles be computed for the default percentile levels, which are {1, 5, 10, 25, 50, 75, 90, 95, 99, 99.5}. Each percentile is an estimate of the value-at-risk (VaR). The TVARPTS= option requests that the estimates of the tail value-at-risk (TVaR) be computed for 90th, 95th, 97.5th, 98.5th, and 99.5th percentile levels.

The OUTPUT statement requests that the total loss sample be written in the tloss variable in the output data table mycas.TlossSample. You can use this output table especially to compute a risk measure that is different from the two popular risk measures, VaR and TVaR, which PROC ECM computes for you.

Figure 12.1 shows some of the results that the preceding ECM step prepares. The “Summary Statistics” table displays various summary statistics. The “Percentiles” table displays the percentiles. The P th percentile is essentially an estimate of the VaR at level $\alpha = P/100$, because for a random loss variable L , the VaR at level α is defined as the smallest number l such that the probability that the loss exceeds l is no larger than $1 - \alpha$. Formally,

$$\begin{aligned}\text{VaR}_\alpha(L) &= \inf\{l \in \mathbb{R} : \Pr[L > l] \leq 1 - \alpha\} \\ &= \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\} \\ &= F_L^{-1}(\alpha)\end{aligned}$$

where F_L and F_L^{-1} denote the CDF and the quantile function of L , respectively. A percentile is essentially an empirical estimate of the quantile function, and hence an estimate of the VaR.

Figure 12.1 Summary Statistics and Percentile Estimates for the Total Loss

The ECM Procedure	
Summary Statistics	
Sample Size	500000
Minimum	0
Maximum	161698.7
Mean	14653.4
Standard Deviation	11330.6
Variance	128383524
Skewness	1.39642
Kurtosis	3.33577
Median	12307.6
Interquartile Range	14061.3
Percentiles	
Percentile	Value
1	0
5	972.38250
10	2549.5
25	6310.9
50	12307.6
75	20372.2
90	29598.5
95	36244.7
97.5	42721.0
98.5	47460.3
99	51283.2
99.5	57915.0
Percentile Method = 5	

The “Tail Value-at-Risk Estimates” table in [Figure 12.2](#) displays the VaR and TVaR estimates for the percentile levels that are specified in the TVARPTS= option. The $\Pr > \text{VaR}$ column indicates the probability that a loss exceeds the VaR value. For a level α , the TVaR_α is defined as the expected value of the loss given that the loss exceeds VaR_α . Formally, $\text{TVaR}_\alpha = E[L | L > \text{VaR}_\alpha]$. Because of this definition, TVaR is also sometimes called a conditional tail expectation (CTE). Computationally, it is the average of the loss values that exceed the P th percentile. As the “Tail Value-at-Risk Estimates” table shows, the TVaR value is always greater than the VaR value, which is expected because it is the mean of the losses that exceed the VaR value. In fact, TVaR is a better risk measure than the VaR, because TVaR accounts for the shape of the tail of the distribution. To capture the tail more accurately, you need to simulate not only large samples of the loss probabilities (copula simulation) but also large samples of each of the marginals. PROC ECM is specifically designed to handle very large copula simulation and marginal samples. It uses all the computational resources of a CAS server, and it uses a novel parallel and distributed algorithm for approximating the empirical distribution function (EDF) for computing the percentiles of the marginals. Together, these uses enable PROC ECM to perform the necessary computations significantly faster.

Figure 12.2 Tail Value-at-Risk (TVaR) Estimates for the Total Loss

Tail Value-at-Risk Estimates			
Percentile	Pr > VaR	Value at Risk (VaR)	Tail Value at Risk
90	0.1	29598.5	39108.7
95	0.05	36244.7	45682.5
97.5	0.025	42721.0	52247.9
98.5	0.015	47460.3	57151.1
99.5	0.005	57915.0	67992.2

Syntax: ECM Procedure

The following statements are available in the ECM procedure:

```

PROC ECM options ;
  MARGINAL marginal-variable: DATA=CAS-libref.data-table < marginal-options > ;
  VAR marginal-variables ;
  OUTPUT OUT=CAS-libref.data-table < output-options > ;
  OUTSUM < outsum-options > < statistic-keyword < =variable-name > ... statistic-keyword < =variable-name > > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;

```

Functional Summary

Table 12.1 summarizes the statements and options available in the ECM procedure.

Table 12.1 PROC ECM Functional Summary

Description	Statement	Option
Statements		
Specifies the information about a marginal variable	MARGINAL	
Specifies the subset of marginal variables to analyze	VAR	
Specifies where and how to write the full total loss sample	OUTPUT	
Specifies where and how to write the summary statistics of the total loss sample	OUTSUM	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	

Table 12.1 *continued*

Description	Statement	Option
Data Table Options		
Specifies the input data table that contains the copula simulation	PROC ECM	DATA=
Specifies the input data table that contains a marginal sample	MARGINAL	DATA=
Specifies the output data table for the total loss sample	OUTPUT	OUT=
Specifies the output data table for the summary statistics	OUTSUM	OUT=
Marginal Input Options		
Specifies the draw (sample) identifier for all marginals	PROC ECM	DRAWID=
Specifies the draw (sample) identifier for a specific marginal variable	MARGINAL	DRAWID=
Specifies the variable that contains the marginal variable sample	MARGINAL	SAMPLEVAR=
Empirical Distribution Function (EDF)		
Estimation Options		
Specifies the EDF accuracy for the body region of the distribution	PROC ECM	EDFACCURACY=
Computes exact counts for the final set of bins	PROC ECM	EXACTFINALCOUNT
Specifies the maximum number of iterations for the EDF estimation algorithm	PROC ECM	MAXITER=
Suppresses the shuffling of marginal data	PROC ECM	NOSHUFFLE
Specifies a sampling fraction for the body region of the distribution	PROC ECM	SAMPLEFRACTION=
Specifies a seed for the internal pseudorandom number generator	PROC ECM	SEED=
Specifies the EDF accuracy for the tail region of the distribution	PROC ECM	TAILEDFACTURACY=
Specifies the EDF value where the tail region of the distribution begins	PROC ECM	TAILSTARTEDF=
Specifies the tolerance for the EDF estimation algorithm	PROC ECM	TOLERANCE=
Output Preparation Options		
Writes the EDF and percentile estimates of marginals to the output data table	OUTPUT	MARGINALSOUT
Specifies the decimal precision to use in forming percentile variable names	OUTSUM	PCTLNDEC=

Table 12.1 *continued*

Description	Statement	Option
Specifies the prefix for the percentile variable names	OUTSUM	PCTLPRE=
Specifies the percentiles to compute and report	OUTSUM	PCTLPTS=
Specifies the method for computing the percentiles	PROC ECM	PCTLDEF=
Specifies the method for computing the percentiles of a specific marginal variable	MARGINAL	PCTLDEF=
Specifies the variable name for the total loss	OUTPUT	TOTALLOSSVAR=
Specifies the prefix for the TVaR variable names	OUTSUM	TVARPRE=
Specifies percentile values for which to compute and report the TVaR	OUTSUM	TVARPTS=
Specifies the denominator for computing second- and higher-order moments	PROC ECM	VARDEF=
Displayed Output Options		
Suppresses all displayed output	PROC ECM	NOPRINT
Specifies which displayed output to produce	PROC ECM	PRINT=

PROC ECM Statement

PROC ECM *options* ;

The PROC ECM statement invokes the procedure. You can specify the following *options*, which are listed in alphabetical order.

DATA=*CAS-libref.data-table*

COPULASIM=*CAS-libref.data-table*

names the input data table for PROC ECM to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 552.

data-table specifies the name of the input data table.

The DATA= data table specifies a copula sample on a uniform scale that is generated by using the OUTUNIFORM= option in the SIMULATE statement in the CCOPULA procedure.

Each observation in this data table is expected to contain the probability of the loss for each of the marginal variables. That probability is determined by accounting for the dependence structure among the marginal variables.

DRAWID=*sample-identifier*

SAMPLEID=*sample-identifier*

identifies the sample to use when the marginal data tables contain multiple samples of the marginal variable, where *sample-identifier* is an integer. A marginal data table can contain multiple samples when it is a result of the perturbation analysis that the CCDM procedure conducts. Each sample is identified by the value of the `_DRAWID_` variable. If you specify a *sample-identifier* value of *k*, then PROC ECM uses the observations in the marginal data tables whose `_DRAWID_ = k`.

The *sample-identifier* applies to all the marginal data tables that you specify in one or more MARGINAL statements. You can override the *sample-identifier* for a specific marginal table by specifying the DRAWID= option in the MARGINAL statement that you specify for the respective marginal variable.

If you omit this option or the DRAWID= option in the MARGINAL statement, then PROC ECM uses a default value of 0 for the marginal tables that contain the `_DRAWID_` variable.

EDFACCURACY=*number*

ACC=*number*

specifies the accuracy to be achieved when estimating the empirical distribution function (EDF) in the body region of the distribution, where the body region is defined as the region that contains values whose EDF estimate is less than the value that you specify in the TAILSTARTEDF= option. The *number* you specify represents the desired proportion of observations that each bin in the body region should contain. The *number* must be between 0 and 1.

PROC ECM uses *number* for estimating the EDF of each marginal. If you specify percentiles in the OUTSUM statement, PROC ECM also uses *number* when estimating the EDF of the total loss.

By default, EDFACCURACY=1.0E-5.

EXACTFINALCOUNT

computes the empirical distribution function (EDF) by making an additional pass over the data to count the final set of bins when the data are sampled and shuffled by the binning algorithm. This results in better EDF estimates, but at the cost of an additional data pass.

If you omit this option, then the final bin counts are approximated by inflating the counts by the sampling fraction. When the sampling fraction is less than 1, this count approximation happens only for the bins in the body region of the distribution.

If you specify a value of 1 for the **SAMPLEFRACTION=** option or if you specify the **NOSHUFFLE** option, then this option is not relevant and PROC ECM ignores it.

MAXITER=*number*

specifies the maximum number of iterations for the equal-proportions binning algorithm.

By default, MAXITER=50.

NOPRINT

suppresses all displayed output. If you specify this option, then PROC ECM ignores any value that you specify for the PRINT= option.

NOSHUFFLE

suppresses the use of shuffling in the EDF estimation process.

If you omit this option, then PROC ECM shuffles each marginal's data to different worker nodes so that it can perform in parallel the binning for different ranges of the marginal data. However, this additional parallelism comes at the cost of moving data across workers. When the majority of the data is in the body region of the distribution, you can reduce the amount of data being moved by using the [SAMPLEFRACTION=](#) option. However, if a marginal's data are very large or the [SAMPLEFRACTION=](#) value is closer to 1, then the amount of data movement can make the algorithm slow.

If you specify this option, the binning algorithm does not shuffle the marginal data. Instead, it estimates the boundaries for a global set of bins in each of the regions (body and tail) of the distribution. This estimation increases the number of parameters that the algorithm needs to optimize while reducing the parallelism in the binning algorithm. Also, because the algorithm needs to share the bin boundaries with all the worker nodes after every iteration, if the values of the [EDFACCURACY=](#) and [TAILEDACCURACY=](#) options are very small (resulting in a large number of bins), you might not save much in the data movement cost. The cost of shuffling the data, which the algorithm incurs only once, might be lower than the cost of sharing the large number of bin boundaries with all worker nodes after each iteration.

If you specify this option, the [SAMPLEFRACTION=](#) and [SEED=](#) options are irrelevant and PROC ECM ignores them.

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the method of computing the percentiles. This option applies to the computation of percentiles of the total loss sample. It also applies to each marginal for which you do not specify a percentile method in the corresponding MARGINAL statement.

You can specify the following values:

- 1** uses the weighted average.
- 2** uses the value closest to the sample size times the percentile.
- 3** uses the empirical distribution function.
- 4** uses the weighted average (identical to PCTLDEF=1).
- 5** uses the empirical distribution function with averaging.

For more information about each method, see the section “[Percentile Computation Methods](#)” on page 573. By default, PCTLDEF=5.

PRINT <(global-display-option)> =display-option

PRINT <(global-display-option)> =(display-option ... display-option)

specifies the desired displayed output. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses. For more information about the displayed output, see the section “[Displayed Output](#)” on page 576.

You can specify the following *global-display-option*:

ONLY

displays only the output that is requested by the *display-options*.

You can specify the following *display-options*:

ALL

displays all the output.

DATASUMMARY**DSUM**

displays a summary of the sample that is analyzed for each variable, marginal variable, or total loss variable for which PROC ECM prepares an empirical distribution function (EDF) estimate.

EDFINIT**EINIT**

displays the summary of the first initialization stage of the EDF estimation process.

EDFOPTDETAILS**EOPT**

displays the details for the data ranges for which the EDF estimation process runs an iterative binning algorithm.

EDFSUMMARY**ESUM**

displays the summary of the data ranges that the EDF estimation process analyzes.

NONE

displays no output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

PERCENTILES**PCTL**

displays the percentiles of the total loss sample. This includes all the predefined percentiles and percentiles that you request in the OUTSUM statement.

SUMMARYSTATISTICS**SUMSTAT**

displays the summary statistics of the total loss sample.

TVAR

displays the tail value-at-risk (TVaR) estimates of the total loss sample.

Not specifying the PRINT= option or the ONLY *global-display-option* is equivalent to specifying the PRINT=(SUMMARYSTATISTICS) option if you do not specify any percentiles in the OUTSUM statement, or is equivalent to specifying the PRINT=(PERCENTILES SUMMARYSTATISTICS) option if you specify percentiles in the OUTSUM statement.

SAMPLEFRACTION=*number*

BODYSAMPLEFRACTION=*number*

specifies the fraction of observations to sample from the body region of the distribution during the EDF estimation process, where *number* must be between 0 and 1.

Sampling helps reduce the amount of data that needs to be communicated across the worker nodes of the CAS server. It can reduce the accuracy of the EDF and percentile estimates, but if your marginal samples have large numbers of observations, then the loss in accuracy is not as significant as the reduction in communication cost that the sampling provides.

By default, SAMPLEFRACTION=0.5.

SEED=*number*

specifies an integer to use as the seed in generating the pseudorandom numbers that are used for sampling the body region of the distribution.

If you omit this option or if you specify a *number* that is negative or 0, then PROC ECM uses as the seed a number that depends on the time of day from the computer's clock.

TAILEDFACTURACY=*number*

TAILACC=*number*

specifies the accuracy to be achieved when estimating the empirical distribution function (EDF) in the tail region of the distribution, where *number* (which must be between 0 and 1) is the desired fraction of observations that each bin in the tail region should contain. The tail region is defined as the region that contains values whose EDF estimate is larger than or equal to the value that you specify in the TAILSTARTEDF= option.

PROC ECM uses *number* when it estimates the EDF of each marginal. If you specify percentiles in the OUTSUM statement, PROC ECM also uses *number* when it estimates the EDF of the total loss.

By default, TAILEDFACTURACY=1.0E-6.

TAILSTARTEDF=*number*

TAILSTART=*number*

specifies the empirical distribution function (EDF) value that marks the beginning of the tail region of the distribution, where *number* must be between 0 and 1. The first stage of the EDF estimation algorithm finds an approximate value y_t whose EDF is close to *number*. The part of the sample whose values are less than y_t defines the body region of the distribution, and the remaining values define the tail region of the distribution.

By default, TAILSTARTEDF=0.8.

TOLERANCE= ϵ

TOL= ϵ

specifies the tolerance value that determines when the equal-proportion binning algorithm has converged, where ϵ must be a number between 0 and 1. The algorithm converges when the difference in the fractions of observations in the largest and smallest bins is less than or equal to $\delta\epsilon$, where δ is the accuracy of the body or tail region that is being analyzed.

By default, TOLERANCE=0.5.

VARDEF=divisor

specifies the divisor to use in calculating the variance, standard deviation, kurtosis, and skewness of the total loss sample. You can specify one of the following values for the *divisor*, where N is the sample size:

DF

sets the divisor for variance to $N - 1$. This also changes the definitions of skewness and kurtosis.

N

sets the divisor to N .

By default, VARDEF=DF.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 576. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 576. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

MARGINAL Statement

MARGINAL *marginal-variable*: **DATA=***CAS-libref.data-table* < *marginal-options* > ;

MGL *marginal-variable*: **DATA=***CAS-libref.data-table* < *marginal-options* > ;

The MARGINAL statement enables you to specify information about one *marginal-variable*. Specify one MARGINAL statement for each marginal variable that you want to analyze.

You must specify the name of the *marginal-variable* followed by a colon (':') as the first item in the MARGINAL statement. This *marginal-variable* must appear in the copula simulation data table that you specify in the **DATA=** option in the PROC ECM statement.

Next, you must specify the data table by using the following option:

DATA=*CAS-libref.data-table*

specifies the data table that contains an empirical sample of the *marginal-variable*. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 552. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the **DATA=** option in the PROC ECM statement.

Typically, you specify a data table that was generated by the CCDM procedure in order to simulate a compound distribution sample from the underlying frequency and severity models that the CNTSELECT and SEVSELECT procedures create.

You can specify the following *marginal-options* to specify the variable and observations that contain the marginal sample:

SAMPLEVAR=*sample-variable*

SVAR=*sample-variable*

specifies the variable that contains the marginal sample in the marginal data table.

If the name of the *sample-variable* is the same as the *marginal-variable*, then you do not need to specify this option. If PROC ECM cannot find a variable named *marginal-variable* in the marginal table and if you do not specify this option, then the sample variable name is assumed to be `_AGGSEV_`, which must be present in the marginal data table.

DRAWID=*sample-identifier*

SAMPLEID=*sample-identifier*

identifies the sample to use when the marginal data table contains multiple samples of the marginal variable, where *sample-identifier* is an integer.

The data table can contain multiple samples when it is a result of the perturbation analysis that the CCDM procedure conducts. Each sample is identified by the value of the `_DRAWID_` variable. If you specify a *sample-identifier* value of *k*, then PROC ECM uses the observations from the marginal data table whose `_DRAWID_=k`.

If you do not specify either this option or the DRAWID= option in the PROC ECM statement, then PROC ECM uses a default value of 0 when the marginal table contains the `_DRAWID_` variable.

The value that you specify for this option overrides the value that you specify for the DRAWID= option in the PROC ECM statement.

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the method of computing the percentiles. The value that you specify for this option overrides the value that you specify for the PCTLDEF= option in the PROC ECM statement.

You can specify the following values:

- 1** uses the weighted average.
- 2** uses the value closest to the sample size times the percentile.
- 3** uses the empirical distribution function.
- 4** uses the weighted average (identical to PCTLDEF=1).
- 5** uses the empirical distribution function with averaging.

For more information about each method, see the section “[Percentile Computation Methods](#)” on page 573.

By default, PCTLDEF=5.

OUTPUT Statement

OUTPUT **OUT=***CAS-libref.data-table* < *output-options* > ;

The OUTPUT statement specifies the data table in which PROC ECM outputs the estimated total loss sample.

If you specify more than one OUTPUT statement, only the first one is used.

You must specify the output data table by using the following option:

OUT=*CAS-libref.data-table*

OUTSAMPLE=*CAS-libref.data-table*

specifies the name of the output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 552. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the DATA= option.

You can specify the following *output-options* to control the contents of the OUT= data table:

MARGINALSOUT

WRITEMARGINALS

writes the marginal probability and estimated percentile information to the OUT= data table. The marginal probability is copied from the copula simulation data table.

TOTALLOSSVAR=*variable-name*

VAR=*variable-name*

specifies the name of the variable to contain the total loss value in the OUT= data table. If you omit this option, then the variable is named `_TOTALLOSS_` by default.

For more information about the variables in the OUTSAMPLE= data table, see the section “[OUTSAMPLE= Data Table](#)” on page 575.

OUTSUM Statement

OUTSUM *< outsum-options >* *< statistic-keyword < =variable-name > ... statistic-keyword < =variable-name > >* ;

The OUTSUM statement specifies the data table in which PROC ECM writes the summary statistics of the total loss sample.

If you specify more than one OUTSUM statement, only the first one is used.

You can specify the following *outsum-options*:

OUT=*CAS-libref.data-table*

OUTSUM=*CAS-libref.data-table*

specifies the output data table that contains the summary statistics of the total loss sample. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 552. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

You can control the statistics that appear in this data table by specifying the [PCTLPTS=](#) option, the [TVARPTS=](#) option, and different *statistic-keyword* values.

PCTLNDEC=*integer-value*

specifies the maximum number of decimal places to use when creating the names of the variables for the percentile values in the [PCTLPTS=](#) or [TVARPTS=](#) options. For example, for a percentile value of 99.9995, if you specify [PCTLNDEC=4](#), then the suffix of the variable is 99_9995. Specifying a small number might result in duplicate variable names. If that happens, PROC ECM writes an error message and exits.

By default, PROC ECM computes the required number of decimal places by using the precision of the percentiles that you specify in the [PCTLPTS=](#) option or the [TVARPTS=](#) option.

PCTLPRE=*prefix*

specifies the prefix to use for creating the names of the variables that contain the percentile estimates. The percentile estimate for a value *p* is written to a variable that has the name of the form *prefixt*, where *t* is the text representation of *p* that is formed by replacing the decimal point with an underscore (`_`). For example, if [PCTLPRE=P](#), then the estimate of the 97.5th percentile is written to a variable whose name is `P97_5`. By default, [PCTLPRE=P](#).

PCTLPTS=*percentile-option*

specifies the percentiles of the total loss that you want PROC ECM to compute. The *percentile-option* can have one of the following two forms:

DEFAULT**DEF**

computes only the default list of percentile values, which is {1, 5, 10, 25, 50, 75, 90, 95, 99, 99.5}.

percentile-list

adds the percentiles that you specify in the *percentile-list* to the default list of percentiles, where *percentile-list* is a comma-separated list of percentile values, each of which must belong to the (0,100) open interval. You can also use a list notation of the following form: *< number1 > to < number2 > by < increment >*. For example, the following two options are equivalent:

```
pctlpts=10, 20, 99.6, 99.7, 99.8, 99.9
pctlpts=10, 20, 99.6 to 99.9 by 0.1
```

TVARPRE=*prefix*

specifies the prefix to use for creating the names of the variables that contain the tail value-at-risk (TVaR) estimates. The TVaR estimate for a percentile value *p* is written to a variable that has the name of the form *prefixt*, where *t* is the text representation of *p* that is formed by replacing the decimal point with an underscore (_). For example, if TVARPRE=TVaR, then the tail value-at-risk estimate for the 97.5th percentile is written to a variable whose name is TVaR97_5. By default, TVARPRE=TVaR.

TVARPTS=*tvar-percentile-list*

specifies the percentile values for which you want PROC ECM to compute the tail value-at-risk (TVaR) of the total loss, where *tvar-percentile-list* is a comma-separated list of percentile values, each of which must belong to the (0,100) open interval. You can also use a list notation of the form “*< number1 > to < number2 > by < increment >*”. For example, the following two options are equivalent:

```
tvarpts=90, 95, 96.25, 97.5, 98.75, 99.5
tvarpts=90, 95 to 99 by 1.25, 99.5
```

For each value in the *tvar-percentile-list*, PROC ECM also reports the percentile estimates, which are the estimates of the value-at-risk (VaR) for the corresponding TVaR estimate.

If you specify a data table in the OUTSUM= option, PROC ECM adds the TVaR estimates to that table. You can control the name of each TVaR estimate's variable by specifying the TVARPRE= option.

You can also specify one or more predefined statistics of the total loss sample to be written to the OUTSUM= table in the following form:

*statistic-keyword***< =variable-name >**

writes the statistic that corresponds to the *statistic-keyword* to a variable named *variable-name*. For the predefined statistics that correspond to percentiles, the *variable-name* takes precedence over the default name that PROC ECM forms by using the PCTLPRE= option. All variable names in the OUTSUM= data table have a limit of 32 characters.

If you do not specify the *variable-name*, then PROC ECM writes the statistic to a variable named *statistic-keyword*.

You can specify the following *statistic-keywords*, which result in the specified statistic being written to the OUTSUM= table:

KURTOSIS KURT	writes the kurtosis of the total loss sample.
MEAN	writes the mean of the total loss sample.
P01	writes the first percentile of the total loss sample.
P05	writes the fifth percentile of the total loss sample.
P10	writes the 10th percentile of the total loss sample.
P25 Q1	writes the first quartile (the 25th percentile) of the total loss sample.
P50 MEDIAN Q2	writes the median (the 50th percentile) of the total loss sample.
P75 Q3	writes the third quartile (the 75th percentile) of the total loss sample.
P90	writes the 90th percentile of the total loss sample.
P95	writes the 95th percentile of the total loss sample.
P99	writes the 99th percentile of the total loss sample.
P99_5 P995	writes the 99.5th percentile of the total loss sample.
QRANGE	writes the interquartile range (Q3–Q1) of the total loss sample.
SKEWNESS SKEW	writes the skewness of the total loss sample.
STDDEV STD	writes the standard deviation of the total loss sample.
VARIANCE	writes the variance of the total loss sample.

VAR Statement

VAR *marginal-variables* ;

The VAR statement specifies a subset of marginal variables to analyze.

You must specify a MARGINAL statement for each of the *marginal-variables* that you specify, and each variable in the list of *marginal-variables* must appear in the copula simulation data table.

Details: ECM Procedure

Overview of the Algorithm

The ECM procedure accepts as input two types of data tables. The first is a copula simulation table that contains the probability of a loss that is incurred in the time period of interest for each line of business (LoB) that you include in the copula model. These probabilities account for the dependency among the losses in different LoBs. You typically generate this table by using the CCOPULA procedure. The second type of table is an aggregate loss sample table for each LoB, which essentially encodes an empirical estimate of the probability distribution of the losses for that LoB. You typically generate this table by using the CCDM procedure. In the context of copula modeling, loss in each LoB constitutes a marginal random variable, which is simply called a marginal for brevity. You can specify multiple marginals in a PROC ECM step as long as you have used each specified marginal in the copula model and you provide a loss sample for each marginal.

The main mandate of PROC ECM is to convert the copula simulation table to a sample of the total loss that you expect to incur across all marginals. Let the copula simulation table contain N observations, and let each i th observation contain the probability p_{ik} for the k th marginal out of K marginals. Let $F_k(l)$ and $F_k^{-1}(p)$ denote the cumulative distribution function (CDF) and the quantile function, respectively, for the k th marginal. With this notation, conceptually, PROC ECM needs to execute the following steps:

1. For each i th observation in the copula simulation table:
 - a) Compute L_{ik} (the loss estimate for each k th marginal) as $L_{ik} = F_k^{-1}(p_{ik})$.
 - b) Compute L_i (the estimate of the total loss) by adding the loss estimates of all marginals as $L_i = \sum_{k=1}^K L_{ik}$.
2. Analyze the set of all $\{L_i\}$ values (which together constitute the final sample of the total loss) to compute summary statistics, value-at-risk (VaR) estimates, and tail value-at-risk (TVaR) estimates of the total loss distribution.

The key part in these steps is the computation of the quantile function for each marginal in step 1a. Because the quantile function is not available in a closed-form equation, PROC ECM has to compute it by using the empirical sample of the loss distribution of each marginal. An empirical estimate of a quantile is the percentile. So for each probability value p_{ik} , PROC ECM needs to compute the $100p_{ik}$ th percentile from the sample of the k th marginal. This seems simple at first, but it gets complicated when the marginal's sample is large and distributed across multiple nodes of the CAS server. Moreover, the copula simulation table is also large and distributed across multiple nodes, which means a large number of percentile computations need to be done on multiple nodes for each marginal. One straightforward solution is to assemble the entire sample of each marginal in the memory of one of the worker nodes, sort that sample, and compute the percentile. However, if a percentile of a marginal is computed and available only on one worker node, then the copula simulation table also must reside in its entirety on that same worker node. So essentially, tables of all the marginals and the copula simulation table need to be moved in their entirety to a single worker node. Not only is this time-consuming because of all the data movement among nodes, but it is also a very inefficient use of the available computational power and memory resources. Moreover, depending on the sizes of all tables, this simplistic solution might not even be feasible if the main memory of any one worker node is not large enough to hold all the tables.

To overcome these challenges, PROC ECM implements a novel algorithm to compute each marginal's percentile. The key component of the algorithm is the parallel and distributed estimation of the empirical distribution function (EDF). PROC ECM computes the EDF estimates in the form of a set of bin boundaries, such that the fraction of observations in each bin is approximately equal. By keeping approximately the same fraction in each bin, the algorithm essentially offers you control over the precision of the EDF estimates. The smaller the fraction of observations in each bin, the greater the precision and the higher the accuracy of the EDF estimates. The bin boundaries together represent a compressed version of the original empirical sample. As long as the desired accuracy is not too high to cause each bin to have very few observations, the algorithm can achieve a good degree of data compression and thereby save on the memory that it needs in order to store the bin boundaries. PROC ECM runs an iterative algorithm to adjust the bin boundaries to make the distribution of observation counts in the bins as close to a uniform distribution as possible. After an acceptable set of bin boundaries is found, PROC ECM computes the EDF estimate at the beginning of each bin. To compute a percentile, it uses these EDF estimates to do a quick search for the specified cumulative probability. The boundaries of the bins whose EDF estimates are closest to the specified cumulative probability are used to estimate the percentile. The section “[Percentile Computation Methods](#)” on page 573 describes the different percentile computation methods.

The EDF estimation algorithm is driven by the following main tuning parameters that you can control:

- **Start of the tail region:** This is expressed as the cumulative probability, p_t , at the start of the tail. The first stage of the algorithm performs an initial EDF-axis binning to estimate l_t , which is the $100p_t$ th percentile. PROC ECM uses that percentile to demarcate the body and tail regions of the distribution. All loss values that are larger than l_t are assigned to the tail region, and the rest of the values are assigned to the body region.

You can provide a guidance for the value of p_t by specifying the `TAILSTARTEDF=` option. PROC ECM uses your specified value to decide the number of bins that the first stage uses. In fact, it also uses the initial set of bins that the first stage finds to further demarcate multiple subregions within the body and tail regions, and it assigns those subregions to different worker nodes to process in parallel. The number of bins in the first stage depends not only on p_t but also on the number of worker nodes that are available to PROC ECM and the sampling fraction for the body region, because PROC ECM attempts to distribute the required binning computations equitably among all the available worker nodes. The actual EDF value at the start of the tail might be different than p_t .

- **Desired EDF accuracy:** This is expressed as the desired width of each bin along the EDF axis, which in turn is the fraction of observations that each bin should have. The second stage of the algorithm uses this value to estimate the number of bins it needs to create. The smaller the bin width, the larger the accuracy, the larger the number of bins, and possibly the slower the estimation process. If your specified bin width is so small—that is, your desired EDF accuracy is so high—that the target number of observations in each bin is smaller than five, then PROC ECM does not use the iterative binning algorithm in the second stage. Instead, it uses the sorted set of all the loss values in the region as bin boundaries.

You can specify different accuracy values for the body and tail regions of the distribution by using the `EDFACCURACY=` and `TAILEDFACTURACY=` options, respectively. If you are interested primarily in the tail region of the distribution, which is typically the case for financial losses, you can specify a relatively lower accuracy (larger bin width) for the body region to help speed up the algorithm.

- **Acceptable tolerance in the EDF accuracy:** This is expressed as a fractional multiplier, ϵ ($0 < \epsilon < 1$), of the accuracy. The algorithm attempts to make the difference in the fractions of observations in the

largest and smallest bins less than or equal to $\delta\epsilon$, where δ is the EDF accuracy of the body or tail region that is being analyzed. You can specify ϵ by using the **TOLERANCE=** option.

As the first test of convergence of the binning algorithm, PROC ECM uses Pearson's chi-squared test to test the null hypothesis that the distribution of counts in all the bins is same as the uniform distribution. However, in practice, even after the chi-squared test fails to reject the null hypothesis—that is, the p -value is larger than the usual thresholds of 0.05 or 0.1—the bin boundaries can often be adjusted further to reduce the difference in the fractions of the largest and smallest bins. PROC ECM enables you to control for this difference so that you can further improve the accuracy of the EDF and consequently the percentile estimates.

- **Sampling fraction for the body region:** The bin boundaries that are computed in the first stage of the algorithm demarcate multiple subregions of the body and tail regions. PROC ECM shuffles all values in the sample to these subregions so that they can be processed in parallel on all available worker nodes. When the sample is large and distributed, shuffling can be expensive. So sampling the body region (which typically contains a large fraction of the total observations) helps reduce the amount of data that needs to be processed and communicated among different nodes. Of course, sampling reduces the accuracy in the body region. But for large samples, the loss of accuracy is not as significant as the gains that are produced by reducing the amount of data to process. You can control the sampling fraction by specifying the **SAMPLEFRACTION=** option. Because the sampling process is driven by an internal pseudorandom number generator, you can specify the **SEED=** option to obtain similar results across multiple runs of PROC ECM for identical inputs and identical grid configuration. **NOTE:** The CAS server does not guarantee row order consistency for the same data table that is created or loaded in different CAS sessions, so the results might vary slightly despite specifying the same seed.

The algorithm never samples the tail region in order to acknowledge the fact that most interesting financial losses fall in the tail region.

In addition to the preceding set of main tuning parameters, you can also use the **EXACTFINALCOUNT** and **NOSHUFFLE** options to control the EDF estimation process.

Percentile Computation Methods

The ECM procedure computes a percentile by using the bins that it estimates for each random variable's sample. It uses the counts in each bin to estimate the value of the empirical distribution function (EDF) at the beginning of each bin. Let $\hat{F}_i$ denote the EDF at the beginning of the i th bin. The algorithm estimates B bins from a sample of size N . Assume $\hat{F}_i = 1$ for $i > B$. For the t th percentile, let $p = \frac{t}{100}$. PROC ECM first finds the i th bin that satisfies $\hat{F}_i < p \leq \hat{F}_{i+1}$. Then it computes a ratio g as

$$g = \frac{p - \hat{F}_i}{\hat{F}_{i+1} - \hat{F}_i}$$

Let b_i denote the value at which the i th bin begins. Assume $b_i = x_{\max}$ for $i > B$, where $x_{\max}$ is the maximum value in the sample. Let j denote the integer part of Np . Then the **PCTLDEF=** option defines the t th percentile, y , as described in Table 12.2.

Table 12.2 Percentile Definitions

PCTLDEF=	Description	Formula
1	Weighted average	$y = (1 - g)b_i + gb_{i+1}$
2	Value closest to Np	$y = b_i$ if $g < \frac{1}{2}$ $y = b_i$ if $g = \frac{1}{2}$ and j is even $y = b_{i+1}$ if $g = \frac{1}{2}$ and j is odd $y = b_{i+1}$ if $g > \frac{1}{2}$
3	Empirical distribution function	$y = b_i$ if $g = 0$ $y = b_{i+1}$ if $g > 0$
4	Weighted average (identical to PCTLDEF=1)	$y = (1 - g)b_i + gb_{i+1}$
5	Empirical distribution function with averaging	$y = \frac{1}{2}(b_i + b_{i+1})$ if $g = 0$ $y = b_{i+1}$ if $g > 0$

Note that each PCTLDEF= method is similar to the corresponding method that is used by the UNIVARIATE procedure in Base SAS. For more information, see *Base SAS Procedures Guide: Statistical Procedures*. The main differences are in the definition of g and in the use of the bin boundaries instead of the ordered sample values.

Input Specification

PROC ECM accepts as input the copula simulation data table (which you specify in the DATA= option in the PROC ECM statement) and marginal data tables (which you specify in the DATA= option in the MARGINAL statements). This section describes the information that these tables are expected to contain.

Copula Simulation Data Table

The copula simulation data table is expected to contain the output of the copula simulation at a uniform scale. It must contain all the marginal variables that you want to analyze. Each observation contains the probability of a loss value for each of the marginal variables; this probability is expected to be generated by accounting for the dependence structure between the marginal variables.

This is a required data table; you must specify it in the DATA= option in the PROC ECM statement.

The copula table is usually generated by specifying the OUTUNIFORM= option in the SIMULATE statement in the CCOPULA procedure for a dependency structure that is either defined by you (in the DEFINE statement in PROC CCOPULA) or estimated by PROC CCOPULA (according to the FIT statement in PROC CCOPULA).

Marginal Data Tables

The marginal data tables that you specify in the MARGINAL statements are expected to contain an empirical sample of the corresponding marginal variable. This is a required data table; you must specify it for each marginal that you want to analyze.

The empirical sample must appear in a variable that has one of the following names:

- the name of the marginal variable
- the name that you specify in the **SAMPLEVAR=** option in the MARGINAL statement
- **_AGGSEV_**

If a marginal table contains a variable named **_DRAWID_**, then the table is assumed to contain multiple empirical samples of the marginal variable. The value of the **DRAWID=** option in the MARGINAL statement or the value of the **DRAWID=** option in the PROC ECM statement determines which sample to use for the analysis.

Output Data Tables

PROC ECM writes the output data tables that you specify in the **OUT=** option of the **OUTPUT** and **OUTSUM** statements. The contents of these output data tables are described in the following sections.

OUTSAMPLE= Data Table

The OUTSAMPLE= data table records the sample of the total loss. It contains as many observations as the number of valid observations in the copula simulation data table. The OUTSAMPLE= data table contains the following variables:

Total-loss-variable

indicates the value of the total loss. The name of this variable is the value of the **TOTALLOSSVAR=** option in the OUTPUT statement. If you do not specify the **TOTALLOSSVAR=** option, then this variable is named **_TOTALLOSS_**.

The total loss for each observation is computed as the sum of the loss estimates of individual marginals. The loss estimate of each marginal is computed by inverting the estimated EDF at the probability of that marginal.

If you specify the **MARGINALSOUT** option in the OUTPUT statement, then the table also contains the following variables:

Probability-of-marginal-variable-1 ... Probability-of-marginal-variable-D

indicates the probability values of the *D* marginal variables, where *D* is the number of marginal variables that PROC ECM analyzes. The names of these variables are the *marginal-variable* values that you specify in the respective MARGINAL statements. The value of each variable is identical to the value of the respective variable in the copula simulation data table.

Loss-for-marginal-variable-1 ... Loss-for-marginal-variable-D

indicates the estimates of the losses of the D marginal variables, where D is the number of marginal variables that PROC ECM analyzes. The names of these variables are formed by adding a suffix of `_Loss` to the name of the corresponding marginal variable. The loss for a marginal variable is the percentile estimate for the probability value of that marginal.

OUTSUM= Data Table

The OUTSUM= data table records the estimates of summary statistics, percentiles, and tail values-at-risk (TVaR) of the estimated total loss. You can control the statistics that appear in this data table by specifying the PCTLPTS= option, the TVARPTS= option, and keywords for different statistics in the OUTSUM statement.

If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, then the OUTSUM= data table contains percentile estimates for all the default percentile levels (1, 5, 10, 25, 50, 75, 90, 95, 99, and 99.5) in addition to the percentile levels that you specifically request.

For more information about the method of naming the variables that correspond to the summary statistics, percentiles, and TVaR estimates, see the description of the OUTSUM statement.

Displayed Output

The ECM procedure optionally produces displayed output by using the Output Delivery System (ODS). You control all output by using the PRINT= option in the PROC ECM statement or by using the DISPLAY statement. Table 12.3 relates the PRINT= options to ODS tables.

Table 12.3 ODS Tables Produced by PROC ECM

ODS Table Name	Description	Option
DataSummary	Summary of the marginal or total loss sample	PRINT=DATASUMMARY
EDFInit	Initialization summary of the EDF estimation process	PRINT=EDFINIT
EDFInitOptDetails	Optimization details of the initialization of the EDF estimation process	PRINT=EDFINIT
EDFOptDetails	Optimization details of the EDF estimation process	PRINT=EDFOPTDETAILS
EDFSummary	Summary of the EDF estimation process	PRINT=EDFSUMMARY
Percentiles	Percentiles of the total loss sample	PRINT=PERCENTILES in the PROC ECM statement and PCTLPTS= in the OUTSUM statement

Table 12.3 *continued*

ODS Table Name	Description	Option
SummaryStatistics	Summary statistics of the total loss sample	PRINT=SUMMARYSTATISTICS
TVaR	Tail value-at-risk (TVaR) estimates of the total loss sample	PRINT=TVAR in the PROC ECM statement and TVARPTS= in the OUTSUM statement
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

If you do not specify the PRINT= option and if you do not specify the NOPRINT, PRINT=NONE, or PRINT=ONLY options, then by default PROC ECM produces the SummaryStatistics ODS table. If you specify the PCTLPTS= option in the OUTSUM statement, then the default displayed output also includes the Percentiles ODS table. If you specify the TVARPTS= option in the OUTSUM statement, then the default displayed output also includes the Percentiles and TVaR ODS tables.

ODS Table Details

Table 12.4 provides detailed descriptions of the tables that different PRINT= options display.

Table 12.4 Description of ODS Tables

ODS Table Name	Detailed Description
DataSummary	The “Data Summary” table contains the summary of each sample for which PROC ECM estimates the EDF.
EDFInit	The “EDF Binning Initialization” table contains, for each marginal variable, the summary of the first (initialization) stage of the EDF estimation process. This summary includes the number of bins that the first stage uses and the convergence status of the optimization process of the first stage. It also reports the estimated empirical distribution function (EDF) at the start of the tail region and the tail start value. Finally, it reports whether the second stage of the binning algorithm processes different regions of the data range in parallel by distributing the bins among worker nodes. If you specify the PCTLPTS= or TVARPTS= options in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.

Table 12.4 *continued*

ODS Table Name	Detailed Description
EDFInitOptDetails	The “EDF Binning Initialization Details” table contains, for each marginal variable, the details of the optimization process of the first page. The table reports the desired proportion of counts in each initial bin, the number of iterations, and the final value of the optimization criterion (which is the difference in proportions of the bins that have the largest and the smallest counts at the end of the first stage). The table also reports the chi-square test statistic (along with its degrees of freedom) and the p-value (which indicates how close the final distribution of counts in the bins is to the desired uniform distribution). Typically, a p-value that is larger than 0.05 indicates that the final distribution of bin counts is not statistically significantly different from a uniform distribution. If you specify the PCTLPTS= or TVARPTS= options in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.
EDFOptDetails	The “EDF Binning Optimization Details” table contains, for each marginal variable, the details of the optimization process that PROC ECM runs in each region that does not use data points to form bins. For each region, the table reports the desired proportion, the number of iterations, and the final value of the optimization criterion, which is the difference in proportions of the bins that have the largest and the smallest counts at the end of the optimization. The table also reports the chi-square test statistic along with its degrees of freedom and the p-value. Typically, a p-value that is larger than 0.05 indicates that the final distribution of bin counts in that region is not statistically significantly different from the desired uniform distribution. If you specify the PCTLPTS= or TVARPTS= options in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.
EDFSummary	The “EDF Binning Summary” table contains, for each marginal variable, the summary of the regions for which PROC ECM processes the bins independently. For each region, it reports the range and location (body, tail, or body-tail), the number of observations in that region, and the number of bins that are formed in that region. If the status for a region indicates that PROC ECM has used the data points to form bin boundaries, then PROC ECM does not run the optimization process for that region and that region’s information does not appear in the “EDF Binning Optimization Details” table. For all other regions, PROC ECM runs the optimization process, reports the convergence status of the optimization process in this table, and reports the details of the optimization process in the “EDF Binning Optimization Details” table. If you specify the PCTLPTS= or TVARPTS= options in the OUTSUM statement, PROC ECM displays this table also for the total loss variable’s sample.

Table 12.4 *continued*

ODS Table Name	Detailed Description
Percentiles	The “Percentiles” table contains estimates of the percentiles of the total loss sample. This table includes the default set of percentiles and the percentiles that you request in the OUTSUM statement.
SummaryStatistics	The “Summary Statistics” table contains estimates of the following summary statistics of the total loss sample: number of observations, maximum value, minimum value, mean, standard deviation, variance, skewness, and kurtosis. If you specify the PCTLPTS= or TVARPTS= option in the OUTSUM statement, the table also contains the estimates of median and interquartile range.
TVaR	The “Tail Value-at-Risk Estimates” table contains the tail value-at-risk (TVaR) estimates for the percentile levels that you specify in the TVARPTS= option. The table also reports the value-at-risk (VaR) estimate for each percentile level. It also includes the probability that a loss value exceeds the VaR estimate, which is equal to $1 - p/100$ for a percentile level p .

Examples: ECM Procedure

Example 12.1: Developing an Economic Capital Model

This example illustrates all the steps of developing an economic capital model by using SAS Econometrics procedures, including the ECM procedure. The section “[Overview: ECM Procedure](#)” on page 550 describes the process in detail. Following is a quick summary of all the steps in the process:

1. Collect and prepare data.
2. For each line of business (LoB), do the following:
 - a) Fit and select the best frequency model for the number of loss events in a particular time period.
 - b) Fit and select the best severity model for the magnitude of each loss.
 - c) Estimate a compound distribution model (CDM) by combining the best frequency and severity models to create a large sample of the aggregate loss.
3. Estimate a dependency structure among the losses of the LoBs by fitting and selecting the best copula model.
4. Use the best copula model to simulate a large sample of probabilities of loss values that are observed in all LoBs in the same time period.

5. Estimate an economic capital model by combining the copula simulation sample with the aggregate loss samples of all LoBs.

The following subsections illustrate in detail each of these steps by developing a simple ECM for a bank's operational risk.

Data Preparation

Consider that you are a bank and, as part of creating an economic capital model for your operational risk, you want to analyze losses in two LoBs: retail banking and commercial banking. In addition to collecting the loss frequency and loss severity data for each LoB, you should also collect data about the key risk indicators (KRIs), which are the external factors that you believe might affect the frequency and severity of losses in at least one LOB. Let CorpKRI1, CorpKRI2, CbKRI1, CbKRI2, CbKRI3, RbKRI1, RbKRI2, and RbKRI3 be the available KRIs, where the prefix Corp indicates corporate-level KRIs, the prefix Cb indicates KRIs specific to commercial banking, and the prefix Rb indicates KRIs specific to retail banking. Some examples of corporate-level KRIs are the ratio of temporary employees to permanent employees and the number of security breaches that are reported during a year. Some examples of commercial banking KRIs are the number of credit defaults, the proportion of financed assets that are movable, and the penalty claims against your bank that can result from processing delays. Some examples of retail banking KRIs are the number of credit cards that are reported stolen, the fraction of employees who have not undergone fraud detection training, and the number of forged drafts and checks that are presented in a year. To mimic the real-world applications, let different yet overlapping sets of KRIs be used for the frequency and severity models. Let the analysis period be one month, and let the data preparation process result in the following data tables:

- The Work.OpRiskLossCounts data table contains the frequency regressors and the variable NumLoss, which records the number of losses that an LoB incurs in a month. The LoBs are identified by the BY variable Line.
- The Work.OpRiskLosses data table contains the severity regressors and the variable LossValue, which records the severity of each loss. The LoBs are identified by the BY variable Line.
- The Work.OpRiskMatchedLosses data table contains the aggregate losses of all LoBs, where the losses in each LoB are aggregated by month. These matched losses are needed for copula modeling.

The following DATA steps prepare the preceding three data tables:

```

/*** Simulate loss counts data ***/
data OpRiskLossCounts(keep=line corpKRI1 corpKRI2
                      cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 numloss);
  call streaminit(12345);
  array cx{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 cb23;
  array cbetaR{9} _TEMPORARY_ (0.1 1 -0.5 0 0 0 0.5 0.75 0);
  array cbetaC{9} _TEMPORARY_ (0.5 0.75 0.3 -0.25 0.25 0.5 0 0 0.3);

  alpha = 0.35; theta = 1/alpha;
  call streaminit(12345);
  do obs=1 to 1000;
    do i=1 to dim(cx)-1;
      cx(i) = rand('NORMAL');
    end;

```

```

cb23 = cx(4) * cx(5);

line = 'CommercialBanking';
xbeta = cbetaC(1);
do i=1 to dim(cx);
    xbeta = xbeta + cx(i) * cbetaC(i+1);
end;
Mu = exp(xbeta); p = theta/(Mu+theta);
numloss = rand('NEGB',p,theta);
output;

line = 'RetailBanking';
xbeta = cbetaR(1);
do i=1 to dim(cx);
    xbeta = xbeta + cx(i) * cbetaR(i+1);
end;
lambda = exp(xbeta);
numloss = rand('POISSON', lambda);
output;
end;
run;

/** Simulate severity of losses and matched loss data */
data OpRiskLosses(keep=line corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 lossValue)
    OpRiskMatchedLosses(keep=CB RB);
    array sx{6} corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 corplrb3;
    array sbetaC{7} _TEMPORARY_ (5 1 0 0.3 0 0 0);
    array sbetaR{7} _TEMPORARY_ (3.5 0 0.5 0 -0.8 0.6 0.2);
    retain CB RB sigma alpha;
    if (_n_=1) then do;
        call streaminit(67890);
        sigma = 1.0; /* for lognormal */
        alpha = 3.0; /* for gamma */
    end;
    set OpRiskLossCounts(keep=line numloss corpKRI1 corpKRI2 cbKRI2 rbKRI1);
    sx(5) = rand('NORMAL'); /* simulate rbKRI3 value */
    corplrb3 = sx(1) * sx(5);

    if (line='CommercialBanking') then do;
        CB = 0;
        do il=1 to numloss;
            /* lognormal */
            Mu = sbetaC(1);
            do i=1 to dim(sx);
                Mu = Mu + sx(i) * sbetaC(i+1);
            end;
            lossValue = exp(Mu) * rand('LOGNORMAL')**Sigma;
            CB = CB + lossValue;
            output OpRiskLosses;
        end;
    end;
    else do; /* line='RetailBanking' */
        RB = 0;
        do il=1 to numloss;

```

```

        /* gamma */
        Mu = sbetaR(1);
        do i=1 to dim(sx);
            Mu = Mu + sx(i) * sbetaR(i+1);
        end;
        lossValue = rand('Gamma', Alpha, exp(Mu));
        RB = RB + lossValue;
        output OpRiskLosses;
    end;
    output OpRiskMatchedLosses;
end;
run;

```

The following DATA steps load the data tables into your CAS session. This example assumes that the CAS engine libref is named mycas, as defined in the section “Using CAS Sessions and CAS Engine Librefs” on page 552, but you can substitute any appropriately named CAS engine libref.

```

/** Load data tables into the CAS session */
data mycas.OpRiskLossCounts;
    set OpRiskLossCounts;
run;
data mycas.OpRiskLosses;
    set OpRiskLosses;
run;
data mycas.OpRiskMatchedLosses;
    set OpRiskMatchedLosses (where=(CB > 0 and RB > 0));
run;

```

When your frequency or severity models contain regressors, you must also create an external scenario data table to contain values that you expect to observe for each regressor in some future state. The final CDM and ECM estimates depend on this scenario data table. For this example, the scenario data table is named Work.MultiConditionScenario, and it contains observations for 12 months of a future year. The following DATA steps simulate it and load it into your CAS session:

```

/** Generate a scenario data table for multiple operating conditions */
data multiConditionScenario (keep=line opEnvId corpKRI1 corpKRI2
    cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3);
    array x{8} corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 rbKRI1 rbKRI2 rbKRI3;
    call streaminit(5151);
    do opEnvId=1 to 12;
        do i=1 to dim(x);
            x(i) = rand('NORMAL');
        end;
        line = 'CommercialBanking';
        output;
        line = 'RetailBanking';
        output;
    end;
run;
proc print; run;

/** Load the scenario data table into the CAS session */
data mycas.multiConditionScenario;
    set multiConditionScenario;
run;

```


After this critical step of collecting and preparing the data, you are ready for the subsequent modeling and simulation steps.

Frequency (Count) Modeling

The CNTSELECT procedure in SAS Econometrics estimates frequency models. It fits a count regression model, where the dependent variable is a count variable that, conditional on the regressors, has one of the following discrete probability distributions: Poisson, negative binomial with linear or quadratic variance, or Conway-Maxwell-Poisson (CMP). It can also fit a zero-inflated version of each distribution, enabling you to account for the large number of zeros that are often present in real-life loss count data. For detailed information about all the features of PROC CNTSELECT, see Chapter 8, “[The CNTSELECT Procedure](#).”

The following statements use the CNTSELECT procedure to fit Poisson and negative binomial distribution models to the example data:

```

/** Fit Poisson count models */
proc cntselect data=mycas.OpRiskLossCounts store=mycas.opriskStorePoisson;
  by line;
  model numloss=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 cbKRI2*cbKRI3
    rbKRI1 rbKRI2/dist=poisson;
  selection;
  ods output SelectedModel.FitModelSummary=fitPoisson;
run;

/** Fit negative binomial count models */
proc cntselect data=mycas.OpRiskLossCounts store=mycas.opriskStoreNegbin;
  by line;
  model numloss=corpKRI1 corpKRI2 cbKRI1 cbKRI2 cbKRI3 cbKRI2*cbKRI3
    rbKRI1 rbKRI2/dist=negbin(p=2);
  selection;
  ods output SelectedModel.FitModelSummary=fitNegbin
    SelectedModel.ParameterEstimates=estNegbin;
run;

```

In the PROC CNTSELECT step, the BY statement ensures that the specified model is analyzed for both LoBs. The MODEL statements specify all the possible regression effects that you believe can affect the number of losses in all LoBs. The SELECTION statements perform model (variable) selection independently for each BY group. The default settings of the SELECTION statement choose the best subset of regression effects to maximize the Schwarz Bayesian criterion (SBC). The ODS OUTPUT statements store the fit summary of the selected model in the Work.FitPoisson and Work.FitNegbin data sets. The following DATA step reads those two data sets to prepare a comparative fit summary:

```

/** Summarize count modeling results */
data countFitSummary;
  set fitPoisson (keep=line Property PropertyValue where=(Property='SBC'));
  Model='Poisson';
  output;
  set fitNegbin (keep=line Property PropertyValue where=(Property='SBC'));
  Model='NegBin(p=2)';
  output;
run;

```

Output 12.1.1 shows the comparison of the fit summary of the final count models of both business lines. It indicates that the negative binomial model is the best model for the commercial banking LoB, because it has a smaller SBC value than the Poisson model. For the retail banking LoB, the Poisson model is the best model. Occasionally, both the Poisson and negative binomial models might have identical SBC values. This usually implies that the negative binomial model is equivalent to the Poisson model, because the overdispersion parameter α is close to 0. Formally, the conditional variance of the negative binomial distribution is $V(y|x) = \mu + \alpha\mu^2$, where μ denotes the mean. As α becomes closer 0, the variance becomes closer to the mean, making it equivalent to a Poisson distribution. The estimate of the `_Alpha` parameter in Output 12.1.2 confirms this.

Output 12.1.1 Comparing Fit Statistics of Count Models

line	Model	Property	PropertyValue
CommercialBanking	Poisson	SBC	4010.42
CommercialBanking	NegBin(p=2)	SBC	3608.513
RetailBanking	Poisson	SBC	2940.835
RetailBanking	NegBin(p=2)	SBC	2947.743

Output 12.1.2 Parameter Estimates of Negative Binomial Count Model (Retail Banking)

The CNTSELECT Procedure

Selected Model

line=RetailBanking

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Intercept	1	0.124771	0.031714	3.93	<.0001
corpKRI1	1	0.989739	0.019536	50.66	<.0001
corpKRI2	1	-0.504251	0.019392	-26.00	<.0001
rbKRI1	1	0.522582	0.019863	26.31	<.0001
rbKRI2	1	0.712791	0.020270	35.16	<.0001
_Alpha	0	5.5335069E-8	.	.	.

The `STORE=` options in each PROC CNTSELECT step create item-store data tables that store all the model fit information, including the selected model's parameter estimates, which are used to estimate the compound distribution model in a later step.

Severity Modeling

The SEVSELECT procedure in SAS Econometrics estimates severity models. In the simplest form, it fits multiple continuous probability distributions to the input loss severity data (which are possibly censored or truncated), and prepares a comparative summary of the fit statistics of all candidate distributions to help you choose the best candidate. PROC SEVSELECT includes a default set of 10 distributions: Burr, exponential, gamma, generalized Pareto (GPD), inverse Gaussian (Wald), lognormal, Pareto, scaled Tweedie, Tweedie, and Weibull. However, you can define and fit your own distributions, such as mixture distributions. PROC SEVSELECT models the effect of regressors on the scale parameter or the log-scale parameter of

the distribution, and it supports automatic model (variable) selection. For detailed information about all the features of PROC SEVSELECT, see Chapter 15, “[The SEVSELECT Procedure](#).”

The following PROC SEVSELECT step fits severity models for three candidate distributions—gamma, lognormal, and Weibull—to the example data:

```

/*** Fit severity models ***
proc sevselect data=mycas.OpRiskLosses store=mycas.opriskSevstore(promote=yes)
    covout print=(allfitstats);
    by line;
    loss lossValue;
    dist gamma logn weibull;
    scalemodel corpKRI1 corpKRI2 cbKRI2 rbKRI1 rbKRI3 corpKRI1*rbKRI3;
    selection method=forward(select=aicc);
run;

```

The LOSS statement specifies the severity loss variable. The DIST statement specifies the list of candidate distributions. The STORE= option names an item store to hold the final model specifications and their parameter estimates. The COVOUT option requests that the parameter covariance estimates be kept in the item store for use in a later step. The SCALEMODEL statement lists all candidate regression effects. The SELECTION statement requests that the best subset of regression effects be chosen according to the forward selection method, where candidate subsets are compared using the corrected Akaike’s information criterion (AICC). Within each BY group, PROC SEVSELECT runs the regression effect selection process independently for each candidate distribution.

After the selection process completes for all distributions, PROC SEVSELECT prepares a comparative summary of the fit statistics of the final selected scale regression models of all candidate distributions, as shown in [Output 12.1.3](#). There are two classes of fit statistics: likelihood-based (AIC, AICC, and SBC) and EDF-based (KS, AD, CvM). Akaike’s information criterion (AIC) is an asymptotic version of AICC. The Kolmogorov-Smirnov (KS), Anderson-Darling (AD), and Cramér–von Mises (CvM) statistics are the EDF-based statistics, which measure the distance between the CDF and the EDF. For scale regression models, the CDF depends on the values of the regressors, which can vary from one observation to another, making it difficult to interpret the EDF-based fit statistics. So you should use the likelihood-based statistics to compare the scale regression models. For the example data, the lognormal distribution is the best for the commercial banking line, and the gamma distribution is the best for the retail banking line, according to all the likelihood-based fit statistics.

Output 12.1.3 Comparison of Statistics of Fit for the Final Selected Severity Models

The SEVSELECT Procedure

Weibull Distribution Selected Model

line=CommercialBanking

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Gamma	46964	46974	46974	47004	3.17537	64.32374	3.95517
Logn	46471*	46481*	46481*	46512*	3.18322	52.02956	4.16386
Weibull	46986	46996	46996	47027	2.94192*	47.65430*	2.36234*

Asterisk (*) denotes the best model in the column.

Output 12.1.3 continued

line=RetailBanking							
All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Gamma	30851*	30865*	30865*	30908*	7.27617	453.85603	21.00515
Logn	30985	30999	30999	31041	6.66834*	346.69875*	20.11890
Weibull	30979	30993	30993	31036	7.38807	564.92046	18.82058*
Asterisk (*) denotes the best model in the column.							

Compound Distribution Modeling

The CCDM procedure in SAS Econometrics estimates a compound distribution model (CDM) for the aggregate loss by combining the best frequency and severity models. Before illustrating the use of PROC CCDM, it is useful to define the CDM. If N and X represent the frequency and severity random variables, respectively, then the aggregate loss S is defined as $S = \sum_{j=1}^N X_j$. If $F_X(x)$ denotes the cumulative distribution function (CDF) of X and if $\Pr(N = n)$ denotes the probability of seeing n losses as per the frequency distribution, then the CDF of S is theoretically computable as $F_S(s) = \sum_{n=0}^{\infty} \Pr(N = n) \cdot F_X^{*n}(x)$, where $F_X^{*n}(x)$ denotes the n -fold convolution of the CDF of X . The probability distribution model for S , characterized by $F_S(s)$, is called a CDM. Direct computation of F_S is usually a difficult task because the n -fold convolution needs to be computed. Some methods exist for certain combinations of severity and frequency distributions (for example, Klugman, Panjer, and Willmot 1998, Ch. 4), but even those methods do not offer closed-form or easily computable expressions for F_S . Further, when the distributions of X and N are conditional on external factors (regressors), as is usually the case, each set of regressor values results in a different distribution for N and X . For all these reasons, PROC CCDM estimates the CDM by using a Monte Carlo simulation method. For detailed information about all the features of PROC CCDM, see Chapter 5, “The CCDM Procedure.” An important feature that this example uses is PROC CCDM’s ability to generate multiple perturbed aggregate loss samples, each of which corresponds to a random perturbation of the frequency and severity parameters from their mean values according to the covariance structure that the CNTSELECT and SEVSELECT procedures estimate. This feature helps assess the uncertainty (variability) in the estimates of the CDM.

The following PROC CCDM steps estimate the CDM for each LoB. Because the best count model for each LoB is different for this example, you need two separate steps, each of which uses an appropriate count model item-store table in the COUNTSTORE= option.

```

/** Simulate CDM for commercial banking LoB */
proc ccdm data=mycas.multiConditionScenario(where=(line='CommercialBanking'))
    countstore=mycas.opriskStoreNegbin severitystore=mycas.opriskSevstore
    seed=13579 nreplicates=10000 nperturb=50 print=all;
    by line;
    severitymodel logn;
    output out=mycas.aggregateLossCB samplevar=agglossCB / perturbOut;
run;

/** Simulate CDM for retail banking LoB */
proc ccdm data=mycas.multiConditionScenario(where=(line='RetailBanking'))
    countstore=mycas.opriskStorePoisson severitystore=mycas.opriskSevstore
    seed=13579 nreplicates=10000 nperturb=50 print=all;

```

```

by line;
severitymodel gamma;
output out=mycas.aggregateLossRB samplevar=agglossRB / perturbOut;
run;

```

When the frequency and severity models contain regression effects, you need to use the DATA= option to specify a data table that contains an external scenario that you want to analyze. This example uses a simulated scenario in the mycas.MultiConditionScenario data table. The NPERTURB= option requests that parameter perturbation analysis be conducted to generate 50 perturbed samples. The NREPLICATES= option specifies the size of each unperturbed and perturbed aggregate loss sample. The OUTPUT statement and its PERTURBOUT option request that all 51 samples be stored in a data table. PROC CCDM creates the output data table by using parallel computations and stores it in a distributed manner across all worker nodes. These samples are useful for estimating the uncertainty in the final ECM estimates in a later stage of the ECM process.

Output 12.1.4 shows one of the tables that the perturbation analysis produces for the commercial banking LoB. You can use this table to estimate the VaR for an individual LoB. If you use the 99.5th percentile to estimate the VaR, then the “Sample Percentile Perturbation Analysis” table indicates that the VaR for the commercial banking business is approximately $51,506 \pm 2,469$ units. The perturbation analysis gives you an estimate of the variability in each statistic of interest, which is more informative than just the estimate of the mean.

Output 12.1.4 Perturbation Analysis of Aggregate Losses (Commercial Banking LoB)

The CCDM Procedure

Severity Model: Logn
Count Model: NegBin(p=2)

line=CommercialBanking

Sample Percentile Perturbation Analysis		
Percentile	Estimate	Standard Error
1	5331.5	389.61983
5	7981.9	502.82119
25	13327.9	724.53569
50	18397.9	960.34400
75	24732.9	1302.9
95	36434.6	1904.8
99	47025.5	2288.8
99.5	51506.0	2469.0
Number of Perturbed Samples = 50		
Size of Each Sample = 10000		

Copula Modeling

The CCOPULA procedure in SAS Econometrics helps you analyze the dependency among losses of multiple LoBs. It supports the dependency structures that are expressed in terms of an elliptical copula (normal and t copulas) or an Archimedean copula (Clayton, Frank, and Gumbel copulas). A copula is especially useful for modeling the dependency structure in the ECM process, because it decouples the marginal distributions—that is, the probability distributions of individual random variables—from their joint probability distribution. This decoupling is based on an important theorem by Sklar (1959), which shows that a copula can be derived from any joint distribution function and that any copula can be combined with any set of marginal distributions to result in a multivariate distribution function. PROC CCOPULA can both fit a copula model and simulate samples from a fitted copula model. For detailed information about all the features of PROC CCOPULA, see Chapter 6, “The CCOPULA Procedure.”

The following PROC CCOPULA steps fit four different types of copulas to the matched loss data in the data table mycas.OpRiskMatchedLosses:

```

/*** Fit different copula models ***/
proc ccopula data=mycas.OpRiskMatchedLosses;
    var CB RB;
    fit Gumbel / store=mycas.copStoreG;
    ods output fitmodelsummary=fitsummG;
run;
proc ccopula data=mycas.OpRiskMatchedLosses;
    var CB RB;
    fit T / store=mycas.copStoreT;
    ods output fitmodelsummary=fitsummT;
run;
proc ccopula data=mycas.OpRiskMatchedLosses;
    var CB RB;
    fit Frank / store=mycas.copStoreF;
    ods output fitmodelsummary=fitsummF;
run;
proc ccopula data=mycas.OpRiskMatchedLosses;
    var CB RB;
    fit Clayton / store=mycas.copStoreC;
    ods output fitmodelsummary=fitsummC;
run;

```

In each PROC CCOPULA step, the VAR statement specifies the marginal variables, which are CB and RB for losses in the commercial and retail banking LoBs, respectively. The FIT statement specifies the type of the copula and names the item-store data table in which to store the specification and estimates of the copula parameters. The ODS OUTPUT statement stores the fit summary of the copula, which you can analyze to produce a comparative table of the SBC fit statistics of all copulas as shown in [Output 12.1.5](#). The table shows that the Gumbel copula has the best fit for the example data by virtue of having the smallest SBC value.

Output 12.1.5 Fit Statistics for Copula Models

Copula	SBC
Gumbel	-69.6587
T	-61.4826
Frank	-48.3310
Clayton	-25.3843

Copula Simulation

The next step in the ECM process is to simulate multiple future scenarios of loss values that might be observed in all LoBs in the same time period. Each scenario is represented by the probabilities of loss values in all LoBs while taking into account the dependency structure among their losses. The use of the OUTUNIFORM= option in the OUTPUT statement of the CCOPULA procedure helps you create a large number of such simulated scenarios. The following PROC CCOPULA step illustrates that for the best-fitting Gumbel copula for this example's data:

```

/** Simulate marginal probabilities on the uniform scale from the Gumbel copula */
proc ccopula;
    simulate cop / restore=mycas.copStoreG
                ndraws=10000 seed=234
                outuniform=mycas.CbRbProbs;
run;

proc print data=mycas.CbRbProbs(obs=5) noobs; run;

```

The SIMULATE statement specifies the mycas.copStoreG item store to read the estimates of the Gumbel copula's parameter. The NDRAWS= option simulates 10,000 observations from this copula. The OUTUNIFORM= option requests that the observations be generated in the uniform scale and stored in the mycas.CbRbProbs data table. PROC CCOPULA uses the multiple nodes of a CAS session to simulate and store the data in a distributed manner. The first few simulated observations in the data table are shown in [Output 12.1.6](#). Each observation contains the probability of a loss in each LoB.

Output 12.1.6 Example of Simulated Copula Observations

CB	RB
0.0585445576	0.317802304
0.9579833083	0.9688330154
0.2114479224	0.3521376759
0.3857146159	0.9856645119
0.7071269566	0.7585000789

Economic Capital Modeling: Combining Copula Simulation with CDM Samples

The final step of the ECM process is to estimate the distribution of the total loss across all LoBs. This requires transforming each observation in the copula simulation table (which contains the probability of a loss value in each LoB) to an observation that contains an estimate of the actual loss for each LoB. For a conceptual overview of this process, see the section “[Overview of the Algorithm](#)” on page 571.

The ECM procedure is designed specifically to help you implement this step. The following PROC ECM step converts the loss probabilities in the data table mycas.CbRbProbs to a sample of the total loss across two marginals, CB and RB, for the commercial and retail banking LoBs, respectively:

```

/* Estimate an empirical model for the total loss */
proc ecm data=mycas.CbRbProbs tailstart=0.85 edfaccuracy=1.0e-2
    samplefraction=0.7 seed=789 print=all;
    marginal CB: data=mycas.aggregateLossCB samplevar=agglossCB;
    marginal RB: data=mycas.aggregateLossRB samplevar=agglossRB;
    outsum out=mycas.tlSummary mean stddev skew kurtosis qrange=iqr
    pctlpts=50, 90, 95, 97.5, 99, 99.5 pctlpre=VaR

```



```

    tvarpts=90, 95, 97.5, 99, 99.5 pctlpre=TVaR;
run;

```

The two MARGINAL statements specify the data tables that contain the aggregate loss samples of each LoB as created by the CCDM steps. The identifying name of the marginal that you specify in the MARGINAL statement must match the corresponding name that you use in the VAR statement of the CCOPULA procedure. Because the name of the variable that contains the sample is different from the name of the marginal, you need to specify the SAMPLEVAR= option in each MARGINAL statement.

PROC ECM uses the loss sample in each marginal's data table to estimate the empirical distribution function (EDF) of that marginal. The EDF estimation process works in two stages. The first stage divides the sample values into body and tail regions. For this example, the TAILSTART= option requests that the tail region start at a value whose EDF is approximately 0.85. The actual EDF value at the start of the tail region also depends on the number of worker nodes and the sampling fraction for the body region. The second stage estimates the EDF in each region independently by using the desired accuracy of EDF estimates in each region. By default, PROC ECM attempts to achieve higher accuracy in the tail region. So for this example, you should expect more accurate loss estimates for probability values that are larger than approximately 0.85. The body region can afford to have a relatively lower EDF accuracy. For this example, the EDFACCURACCY= option specifies it to be 0.01, which means that the width of the bins along the EDF axis is approximately 0.01 in the body region. In particular, when PROC ECM processes an interval $[p_1, p_2]$ of the EDF axis, it attempts to bin the data into approximately B bins, where $B \approx (p_2 - p_1)/0.01$. The SAMPLEFRACTION= option requests that only about 70% of the observations in the body region be used for estimating the EDF in that region. This helps reduce the amount of data movement among worker nodes. The SEED= option ensures that the results are similar across multiple runs of the procedure with identical inputs.

For each observation in the DATA= data table, PROC ECM computes the percentile for the probability value of each marginal by using the estimated EDF of that marginal. Adding these percentiles across all marginals results in the estimate of the total loss for that observation. Repeating this process for all observations results in the sample of the total loss, which encodes the probability distribution of the total loss, which in turn is an empirical estimate of the economic capital model (ECM). The OUTSUM statement specifies the summary statistics and various percentile levels of risk measures, VaR and TVaR, that you want to compute for the estimated ECM. The OUT= option requests that the computed statistics and risk measures be stored in the data table mycas.tlSummary.

The PRINT=ALL option in the PROC ECM statement displays all the result tables that PROC ECM produces. For each marginal, up to five result tables are displayed, as illustrated in [Output 12.1.7](#), [Output 12.1.8](#), and [Output 12.1.9](#) for the commercial banking marginal CB.

[Output 12.1.7](#) shows the “Data Summary” table, which summarizes the data sample that PROC ECM reads for CB. It displays the data table name; the number of total, valid, and zero-valued observations; and the range of the data. When the input data table is a result of the parameter perturbation analysis that PROC CCDM conducts (as in this example), it contains multiple samples, each identified by a unique value of the _DRAWID_ variable. The “Data Summary” table displays the draw identifier, which is the value of the _DRAWID_ variable it uses to read the marginal sample. You can control this value by specifying a global value in the DRAWID= option in the PROC ECM statement or a marginal-specific value in the DRAWID= option in a MARGINAL statement. Similarly, the input data table can contain samples for multiple severity distributions when you specify multiple distributions in the SEVERITYMODEL statement of PROC CCDM. PROC ECM chooses the first severity distribution that it finds and displays its name in the “Data Summary” table.

Output 12.1.7 Data Summary (Commercial Banking)**The ECM Procedure****Marginal Variable CB**

Data Summary	
Data Table	AGGREGATELOSSCB
Severity Model	Logn
Draw Identifier	0
Observations	10000
Valid Observations	10000
Zero-valued Observations	0
Minimum	1314.980723
Maximum	83648

Output 12.1.8 shows the tables that provide an insight into the first stage of the EDF estimation algorithm for the commercial banking marginal CB.

Output 12.1.8 Summary of the First Stage of EDF Estimation (Commercial Banking)

EDF Binning Initialization					
Number of Bins	7				
Convergence Status	Converged				
Tail Start EDF	0.858100				
Tail Start	28690				
Bins Distributed Among Workers	Yes				

EDF Binning Initialization Details					
Target	Proportion	Iterations	Criterion	DF	Chi-Square
					Pr > ChiSq
0.142857	4	0.0016000	6	0.1034	1.0000

Following is a description of the contents of each table in Output 12.1.8:

- “EDF Binning Initialization”: This table summarizes the result of the first-stage binning by displaying the number of bins and the status of the iterative bin boundary optimization algorithm. It also shows the EDF value and the actual loss value at the start of the tail region. Finally, it shows whether the bins are distributed among worker nodes at the end of the first stage.
- “EDF Binning Initialization Details”: This table displays the details of the iterative bin boundary optimization algorithm of the first stage. For B number of bins (which is 7 for this marginal), the algorithm attempts to achieve a target proportion of $1/B$ observations in each bin. After three iterations, the maximum difference in fractions of observations in the largest and smallest bins is 0.0016. The chi-square test for uniformity of counts in the bins uses $B - 1$ degrees of freedom. The value of the test statistic and its corresponding p -value show that the final distribution of counts in bins is close to the uniform distribution.

Output 12.1.9 shows the tables that provide an insight into the second stage of the EDF estimation process for the commercial banking marginal CB.

Output 12.1.9 Summary of the Second Stage of EDF Estimation (Commercial Banking)

EDF Binning Summary						
Region	Begin	End	Location	N	Bins	Status
1	1315.0	16836.7	Body	2980	43	Converged
2	16836.7	28689.5	Body	2955	43	Converged
3	28689.5	83647.8	Tail	1419	1420	Data points used as bins

EDF Binning Optimization Details						
		Target				
Region	Proportion	Iterations	Criterion	DF	Chi-Square	Pr > ChiSq
1	0.023256	3	0.010067	42	11.5591	1.0000
2	0.023256	1	0.0071066	42	16.1327	0.9999

Following is a description of the contents of each table in [Output 12.1.9](#):

- “EDF Binning Summary”: This table displays the different subregions of values that the second stage processes in parallel. Typically, the number of rows in this table is proportional to the number of worker nodes. For each subregion, the table displays whether the subregion is located in the body or tail region of the distribution. You might also occasionally see the ‘Body-Tail’ location value, which indicates that a subregion straddles the body and tail regions. Next, the table displays the number of observations that are processed and the number of bins that are used. The number of observations in the body subregions reflect the sampling fraction. The number of bins reflects the specified EDF accuracy. Finally, the last column displays the status of the binning process for that subregion. If the number of bins is small enough to hold at least five observations in each bin, an iterative binning algorithm is used and its convergence status is displayed. If the desired accuracy of the region is too high, the number of bins is large and the algorithm uses data points as bin boundaries, as the table shows for the tail region.
- “EDF Binning Optimization Details”: This table displays the details of the iterative bin boundary optimization algorithm for each subregion that does not use data as bin boundaries. The Region column identifies the region, and the remaining columns are similar to the columns of the “EDF Binning Initialization Details” table. For this example, the binning algorithm needs very few iterations in both subregions to achieve a nearly uniform distribution of counts.

If the algorithm decides to use data points as bin boundaries in all the subregions, then this table is not displayed.

PROC ECM also executes the same EDF estimation algorithm for the retail banking marginal RB, and the PRINT=ALL option displays up to five similar result tables for that marginal. Further, because the OUTSUM statement requests ECM statistics that require percentile-based estimates, PROC ECM also executes the same algorithm to estimate the EDF of the generated total loss sample and then uses that EDF estimate to compute the required percentiles and percentile-based risk measures. The PRINT=ALL option displays up to five similar result tables, which contain the summary of the total loss sample and information about the first and second stages of the EDF estimation algorithm for that sample.

After displaying the results of the EDF estimation algorithms for all marginals and the total loss sample, PROC ECM displays the summary statistics and requested risk measures for the total loss sample. [Output 12.1.10](#) shows the “Summary Statistics” and “Tail Value-at-Risk Estimates” tables for this example.

Output 12.1.10 Summary Statistics and Estimates of Risk Measures (VaR and TVaR) of the ECM

Summary Statistics			
Sample Size		10000	
Minimum		5985.2	
Maximum		82785.3	
Mean		23330.9	
Standard Deviation		9697.7	
Variance		94045774.9	
Skewness		1.03636	
Kurtosis		1.72531	
Median		22042.0	
Interquartile Range		12260.3	

Tail Value-at-Risk Estimates			
Percentile	Pr > VaR	Value at Risk (VaR)	Tail Value at Risk
90	0.1	35801.5	43599.0
95	0.05	41379.5	48895.1
97.5	0.025	46884.5	53857.9
99	0.01	53845.3	59774.4
99.5	0.005	57692.1	63980.3

The “Summary Statistics” table displays the size of the sample, the range of the total loss values in the sample, and various moments of the sample. It also displays median and interquartile range, the robust statistics of location and scale, respectively. The “Tail Value-at-Risk Estimates” table displays the VaR and TVaR estimates for the percentile levels in the TVARPTS= option in the OUTSUM statement. These result tables together provide you a comprehensive statistical picture of the distribution of the total loss. You can especially use the VaR and TVaR estimates to decide the capital requirements for your enterprise. But, you can take this analysis one step further.

The preceding estimates are for a specific set of marginal samples. However, the perturbation analysis that PROC CCDM conducts to capture the uncertainty in the parameter estimates of the severity and frequency models results in multiple samples for each marginal. Similar to how PROC CCDM uses those multiple perturbed samples to estimate the location and dispersion (variability) of various summary statistics and percentiles of a marginal’s CDM, you can use different combinations of the perturbed samples of all the marginals to generate multiple samples of the total loss, compute the risk measures for each such total loss sample, and summarize those estimates to estimate the location and dispersion of all risk measures.

Each perturbed sample is identified by a draw identifier that PROC CCDM writes in the variable `_DRAWID_` when you use the NPerturb= option in the PROC CCDM statement and the PERTURBOUT option in the OUTPUT statement. PROC ECM enables you to specify a value of the draw identifier to use for each marginal by using the DRAWID= option in the MARGINAL statement. The following programming statements use this feature to define a SAS macro that generates multiple combinations of the perturbed samples of both the marginals of this example and invokes PROC ECM for each combination:

```

/*** Macro to generate multiple ECM samples by using multiple perturbed CDM samples,
    and to estimate mean, standard deviation, median, and interquartile range
    of various risk measures of the total loss. ***
%macro EstimateECMvariability(maxDrawId1=10, maxDrawId2=10);

```

```

%do draw1=0 %to &maxDrawId1;
  %do draw2=0 %to &maxDrawId2;
    %put draw1=&draw1 draw2=&draw2;
    proc ecm data=mycas.CbRbProbs noprint;
      marginal CB: data=mycas.aggregateLossCB samplevar=agglossCB drawId=&draw1;
      marginal RB: data=mycas.aggregateLossRB samplevar=agglossRB drawId=&draw2;
      outsum out=mycas.tlSumm mean stddev skew kurtosis qrange=iqr
        pctlpts=50, 95, 97.5, 99.5 pctlpre=VaR
        tvarpts=95, 97.5, 99.5 pctlpre=TVaR;
    run;
    data tlSumm;
      set mycas.tlSumm;
    run;
    %if &draw1=0 and &draw2=0 %then %do;
      data allecmstats;
        set tlSumm;
      run;
    %end;
    %else %do;
      proc datasets library=work nolist;
        append base=allecmstats data=tlSumm;
      run;
    %end;
  %end;
%end;

proc means data=allecmstats(rename=(VaR50=Median)) n mean std median qrange;
  var Mean Median VaR95 TVaR95 VaR97_5 TVaR97_5 VaR99_5 TVaR99_5;
quit;
%mend;

/*** Compute ECM variability estimates ***/
%EstimateECMvariability(maxDrawId1=9, maxDrawId2=9);

```

The preceding code accumulates the summary statistics and risk measures (VaR and TVaR) that each PROC ECM step computes into the data table `Work.AllEcmStats`. The PROC MEANS step then summarizes that data table to estimate the location and dispersion of each summary statistic and risk measure. The final results that the `%EstimateECMvariability` macro prepares are shown in [Output 12.1.11](#). If you decide to use 97.5th percentile as the VaR of the economic capital model, then for this example, the VaR estimate is about $46,577 \pm 1,651$ units if you use the mean and standard deviation as measures of location and dispersion. The corresponding TVaR estimate is about $53,601 \pm 1,927$ units. If you use the median and interquartile range as measures of location and dispersion, then the VaR estimate is about $46,990 \pm 2,893$ units and the TVaR estimate is about $53,986 \pm 3,430$ units.

Output 12.1.11 Location and Dispersion Estimates of Various ECM Summary Statistics and Risk Measures**The MEANS Procedure**

Variable	Label	N	Mean	Std Dev	Median	Quartile Range
MEAN	Mean	100	23055.90	919.7058973	23210.32	1281.05
Median	50th Percentile	100	21640.49	893.0818598	21886.03	1204.07
VaR95	95th Percentile	100	41240.44	1466.52	41558.78	2210.47
TVaR95	Tail Value-at-Risk (TVaR): 95th Percentile	100	48649.55	1720.10	49014.94	3094.74
VaR97_5	97.5th Percentile	100	46576.53	1650.62	46990.43	2893.05
TVaR97_5	Tail Value-at-Risk (TVaR): 97.5th Percentile	100	53600.80	1926.75	53985.67	3429.59
VaR99_5	99.5th Percentile	100	57458.20	2117.15	57889.35	3420.17
TVaR99_5	Tail Value-at-Risk (TVaR): 99.5th Percentile	100	64239.98	2369.58	64864.45	3310.49

You have now completed the process of developing an economic capital model. You should be able to use this template to analyze multiple dependent lines of business or risk categories of your enterprise and get a statistically sound estimate of the economic capital model to decide the minimum capital that you need to set aside for covering the worst-case losses across the parts of the enterprise that you include in the ECM.

References

- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.
- Sklar, A. (1959). “Fonctions de répartition à n dimensions et leurs marges [Distribution functions with n dimensions and their margins].” *Publications de l’Institut de Statistique de L’Université de Paris* 8:229–231.

Chapter 13

The HMM Procedure

Contents

Overview: HMM Procedure	598
Using CAS Sessions and CAS Engine Librefs	599
Getting Started: HMM Procedure	600
Gaussian Hidden Markov Model	600
Gaussian Hidden Markov Model for Cross-Sectional Time Series Data	612
Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data	616
Regime-Switching Regression Model	636
Regime-Switching Autoregression Model	642
Syntax: HMM Procedure	647
Functional Summary	647
PROC HMM Statement	650
DECODE Statement	652
DISPLAY Statement	653
DISPLAYOUT Statement	654
ESTIMATE Statement	655
EVALUATE Statement	655
FILTER Statement	656
FORECAST Statement	656
ID Statement	657
INITIAL Statement	657
MODEL Statement	658
OPTIMIZE Statement	661
PRIOR Statement	664
SCORE Statement	666
SMOOTH Statement	667
STORE Statement	667
Details: HMM Procedure	668
Hidden Markov Model	668
Gaussian Hidden Markov Model	672
Gaussian Mixture Hidden Markov Model	672
Regime-Switching Regression Model	673
Regime-Switching Autoregression Model	674
Regime-Switching Mean-Adjusted Autoregression Model	676
Parameter Estimation Methods	678
Optimization Algorithms	683

Matrix Expression	683
Label Switching Problem	694
Analytic Store Technology	706
Information Criteria	710
Missing Values	710
Data Table Output	711
ODS Table Names	715
Several Types of Models as Special Cases of Hidden Markov Models	717
Examples: HMM Procedure	723
Example 13.1: Discovering the Hidden Market States by Using the Regime-Switching Autoregression Model	723
Example 13.2: Clustering Time Series by Using the Gaussian Hidden Markov Model	757
Example 13.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model	768
References	779

Overview: HMM Procedure

The HMM procedure supports hidden Markov models (HMMs), which have been widely applied in economics, finance, science, and engineering. The hidden Markov models have many well-known aliases, such as the general state space model (GSSM), regime-switching model (RSM), Markov-switching model (MSM), and Markov regime-switching model (MRSM). PROC HMM supports the Gaussian hidden Markov model (Gaussian HMM), the Gaussian mixture hidden Markov model (GM HMM), the regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM), and the regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) in both the standard and mean-adjusted forms.

The HMM procedure supports the following:

- cross-sectional time series data
- the maximum likelihood (ML) method and the maximum a posteriori (MAP) method for parameter estimation, which is a nonlinear optimization problem
- the prior distribution for parameters when you use the MAP method
- three types of optimization algorithms: the active set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm
- multistart mode for the active set and interior point optimization algorithms; in this mode, the local solver approaches the problem from multiple starting points, possibly finding a better local optimum as a result. However, the computing cost in multistart mode can be huge.
- specification of initial values for optimization
- state-independent constraints on parameters

- output of parameter estimates to a data table, which can be read back later in order to use those parameter estimates as the initial values for a new estimation
- output of the covariance matrix of parameter estimates
- specification of the order in which to label the states (and components in the case of GM HMMs) in order to solve the label switching problem, and output of the changes of state labels (and component labels in the case of GM HMMs)
- estimation of several models that have different numbers of states or different orders of autoregressive process (or both) in order to help you with the important and difficult task of model selection
- output of the filtering, smoothing, and decoding results
- output of the mean, standard error, covariance matrix, median, and confidence interval of forecasts of dependent variables
- output of the probabilities of states in the forecasts
- multistep forecasts after each observation
- output of the log likelihood up to any observation
- output of the following information criteria: Akaike's information criterion (AIC), the corrected AIC (AICC), the Bayesian information criterion (BIC), and the Hannan-Quinn criterion (HQC)
- saving the model information and using it later to score other new data
- the analytic store technology. You can even apply HMMs to new data on a platform that does not support the HMM procedure.

PROC HMM requires SAS Cloud Analytic Services (CAS) in order to run, and it does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: HMM Procedure

This section provides several examples of the types of models that the HMM procedure supports.

Gaussian Hidden Markov Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a normal distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the variable $\mathbf{S}_t$. The variable $\mathbf{S}_t$, the so-called state, follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian hidden Markov model (Gaussian HMM). The Gaussian HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \dots, K\}$, consisting of parameters that define the state-dependent distribution of observable variables.

Consider a bivariate Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.001 & 0.999 \\ 0.999 & 0.001 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```
%let pi1 = 0.5;
%let a11 = 0.001;
%let a22 = 0.001;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;
%let T = 10000;
%let seed = 1234;

data DGPone;
  retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &pi1.;
      /* Cholesky decomposition of sigma1 */
      cd1_11 = sqrt(&sigma1_11.);
      cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
      cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
      /* Cholesky decomposition of sigma2 */
      cd2_11 = sqrt(&sigma2_11.);
```

```

        cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
        cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
    end;
    else do;
        /* transition probability matrix */
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
        /* (x,y) ~ N(mu1, Sigma1) at state 1 */
        x = &mu1_1. + cd1_11*e1;
        y = &mu1_2. + cd1_21*e1+cd1_22*e2;
    end;
    else do;
        /* (x,y) ~ N(mu2, Sigma2) at state 2 */
        x = &mu2_1. + cd2_11*e1;
        y = &mu2_2. + cd2_21*e1+cd2_22*e2;
    end;
    output;
    lags = s;
end;
run;

data One;
    set DGPone;
    keep t x y;
run;

```

In general, the data generating process is unknown. What can be seen is the data table One, but not the data table DGPone.

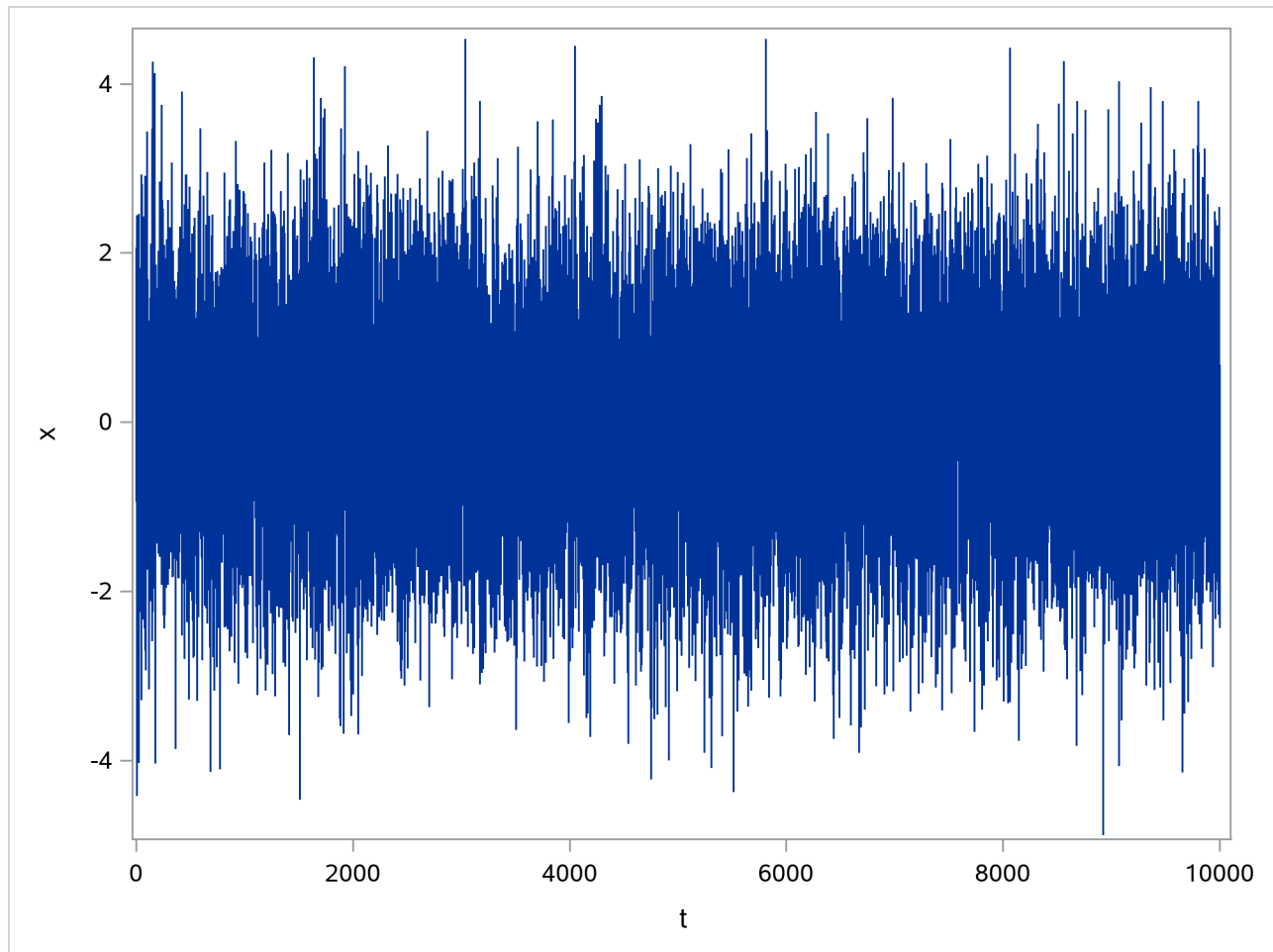
The following statements plot the simulated series x :

```

proc sgplot data=One;
    series x=t y=x;
run;

```

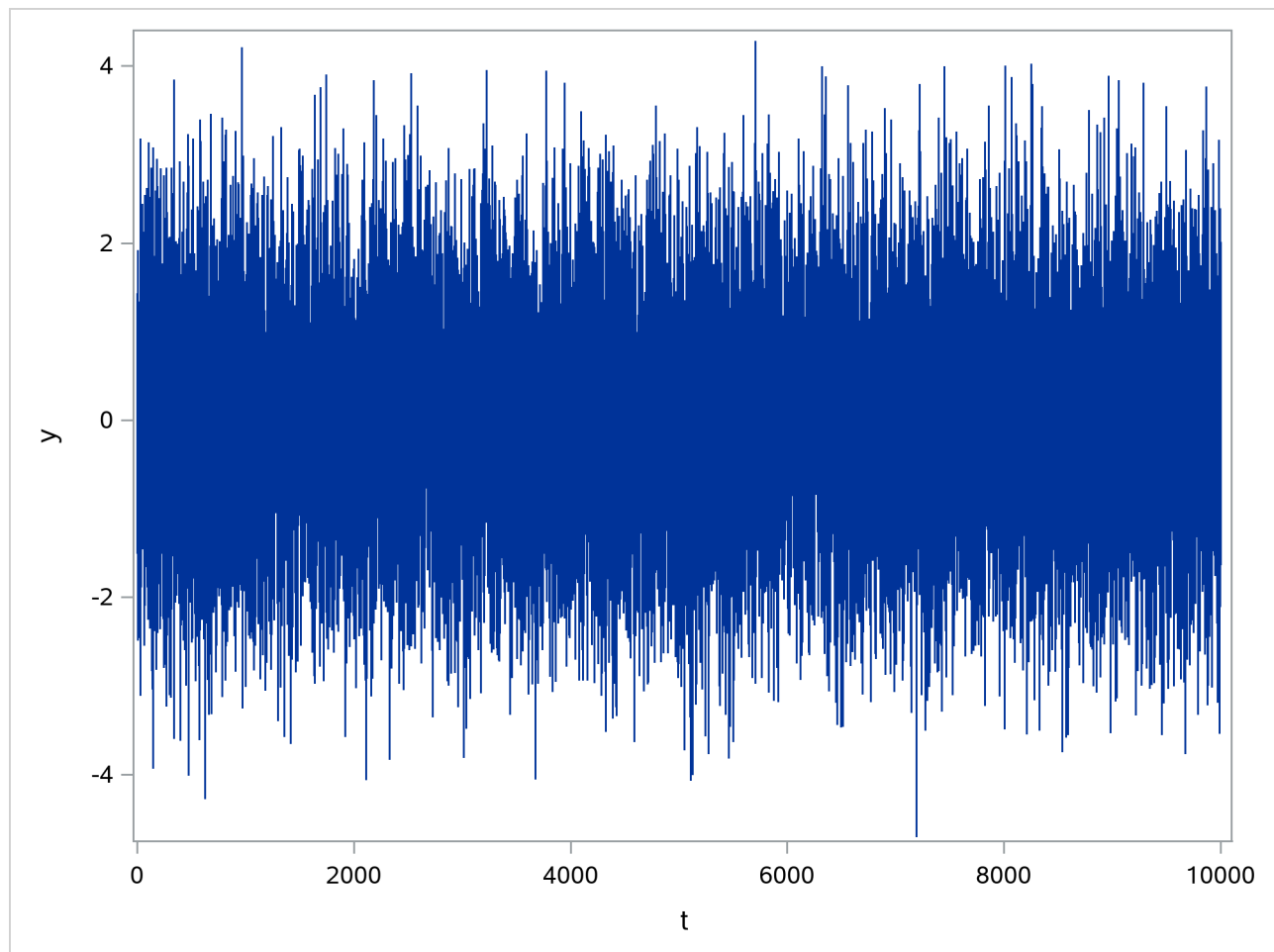
Figure 13.1 shows the plot of x , whose points seem to fall randomly around 0.

Figure 13.1 Plot of Time Series x 

The following statements plot the simulated series y :

```
proc sgplot data=One;  
  series x=t y=y;  
run;
```

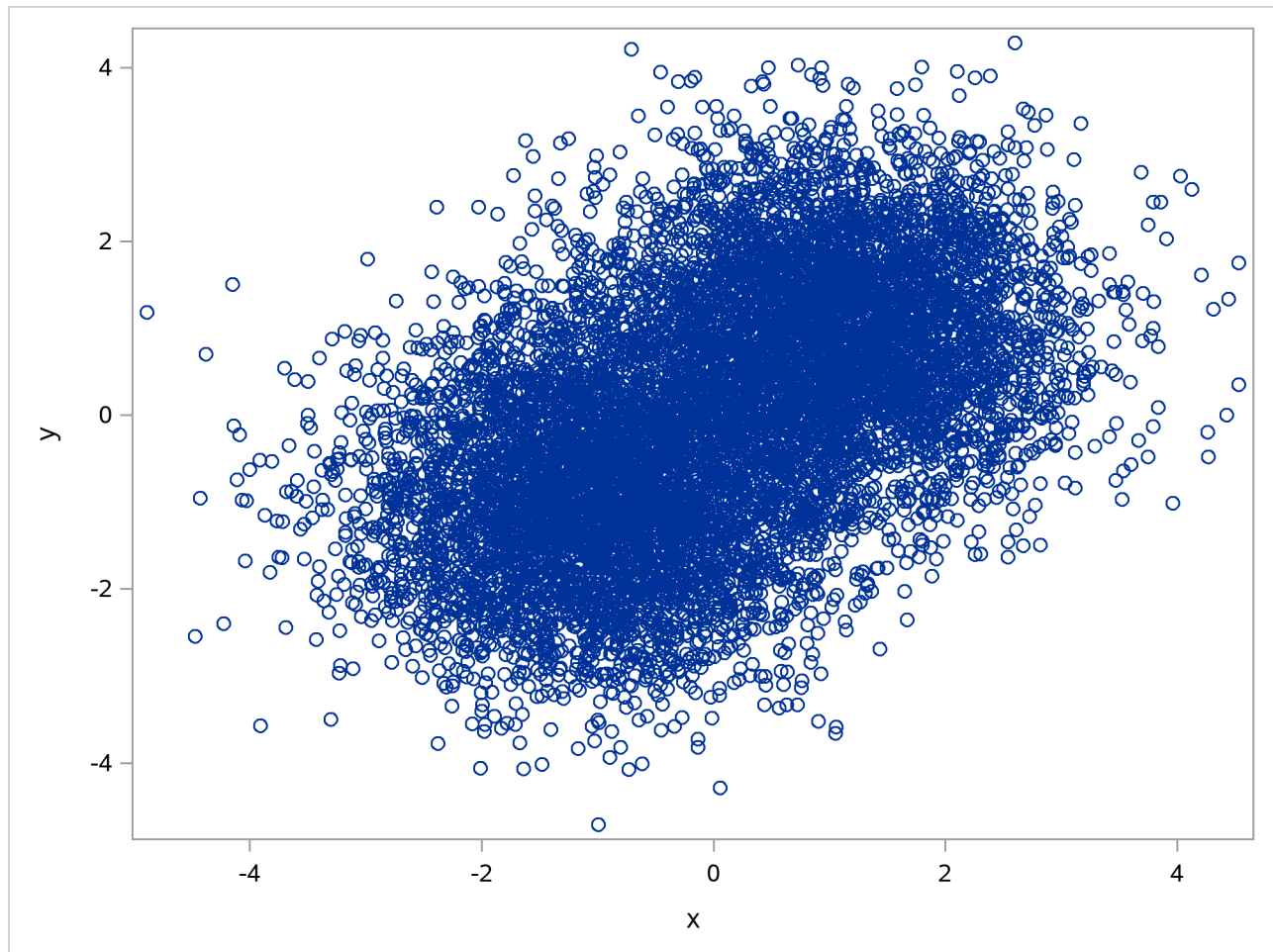
Figure 13.2 shows the plot of y , whose points also seem to fall randomly around 0.

Figure 13.2 Plot of Time Series y 

Ignoring the sequential information, the following statements plot the observed points (x, y) :

```
proc sgplot data=One;
  scatter y=y x=x;
run;
```

As shown in [Figure 13.3](#), it seems that the points come from two different cluster centers, and many of them lie between two cluster centers.

Figure 13.3 Unclassified Observations

It is very difficult for any cluster analysis methods that cannot use the sequential information to find the correct states for the points that lie between two cluster centers. The simplest cluster analysis method to classify the points, because of the symmetry shown in Figure 13.3, is to draw a line $y = -x$, classify all points above the line to one state, and classify all points below the line to another state.¹ The following statements implement this simplest cluster analysis method and calculate the accuracy, based on the data table DGPone:

```
/* classify the data points by the line  $y = -x$  */
data Cluster;
  set One;
  if (y > -x) then state = 1;
  else state = 2;
  keep t state;
run;

data clusterCheck;
  merge DGPone(in=a) Cluster(in=b);
```

¹In theory, no cluster analysis method, without using the sequential information, could do a better job on this data table than this simplest cluster analysis method.

```

    by t;
    if(s=state) then correct = 1;
    else correct = 0;
    if(a=b);
    keep correct;
run;

data clusterAccuracy;
    set clusterCheck;
    retain count 0 correctCount 0;
    count = count + 1;
    correctCount = correctCount + correct;
    if(count>=T.-0.5) then do;
        accuracy = correctCount / count;
        if (accuracy<0.5) then accuracy = 1 - accuracy;
        output;
    end;
    keep accuracy;
run;

proc print data = clusterAccuracy noobs; run;

```

The accuracy of the cluster analysis, which is shown in [Figure 13.4](#), is 92.02%. This implies that the cluster analysis cannot choose the correct states for about 800 data points!

Figure 13.4 Accuracy of Cluster Analysis

<u>accuracy</u>
0.9202

What is the big difference that you get by using the sequential information through HMMs? Before estimating any model, make a copy of the client-side data on the server:

```
data mycas.One; set One; run;
```

The following statements estimate the two-state Gaussian HMM and then classify the data points through decoding; here decoding means to find the best possible state path for the observed data. The SORT=DESC(MU) option requests that the states be labeled in descending order of the mean parameter values; for more information about the label switching problem, see the section “[Label Switching Problem](#)” on page 694.

```

proc hmm data=mycas.One labelSwitch=(sort=desc(mu));
    id time=t;
    model x y / type=gaussian nstate=2;
    optimize printLevel=3 printIterFreq=1;
    decode out=mycas.oneDecode;
run;

```

The number of observations and model information are shown in [Figure 13.5](#).

Figure 13.5 Number of Observations and Model Information

Observation Information	
Number of Observations	10000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian HMM
Stationary	Yes
Number of States	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in Figure 13.6.

Figure 13.6 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	Estimate
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MU1_1	0.642059
MU1_2	0.513308
MU2_1	-1.609545
MU2_2	-1.329369
SIGMA1_1_1	1.152038
SIGMA1_2_1	0.320856
SIGMA1_2_2	1.478814
SIGMA2_1_1	0.525065
SIGMA2_2_1	-0.243910
SIGMA2_2_2	0.892018

Initial Value of Objective Function	-34510.54106
-------------------------------------	--------------

The details of the optimization algorithm are shown in Figure 13.7.

Figure 13.7 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in [Figure 13.8](#).

Figure 13.8 Iterations of the Optimization Process

Iteration History						
Iteration	Objective Function	Optimality Error	Infeasibility	Function Calls	Gradient Calls	Hessian Calls
1	-34510.54106	610.24392	0.00000	2	1	0
2	-29343.35393	350.54614	0.00000	11	10	1
3	-28606.45004	78.05203	0.00000	19	18	2
4	-28467.87680	1.42097	0.00000	28	27	3
5	-28460.41282	0.34998	0.00000	37	36	4
6	-28459.08728	0.10030	0.00000	46	45	5
7	-28458.80039	0.02968	0.00000	55	54	6
8	-28458.73576	0.00812	0.00000	64	63	7
9	-28458.72552	0.00167	0.00000	73	72	8
10	-28458.72484	0.00018	0.00000	82	81	9
11	-28458.72483	0.00000	0.00000	91	90	10

The final parameter values and the objective function value at the end of the optimization are shown in [Figure 13.9](#).

Figure 13.9 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	Estimate
TPM1_1	0.002071
TPM1_2	0.997929
TPM2_1	0.999863
TPM2_2	0.000137
MU1_1	1.006466
MU1_2	0.994184
MU2_1	-0.993036
MU2_2	-1.007869
SIGMA1_1_1	1.011152
SIGMA1_2_1	0.006387
SIGMA1_2_2	0.988329
SIGMA2_1_1	0.992333
SIGMA2_2_1	-0.005378
SIGMA2_2_2	1.008722

Final Value of Objective Function	-28458.72483
-----------------------------------	--------------

The estimates of the initial state probability vector (ISPV) are shown in [Figure 13.10](#).

Figure 13.10 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.50048
2	0.49952

The estimates of the transition probability matrix (TPM) are shown in [Figure 13.11](#).

Figure 13.11 Estimates of Transition Probability Matrix

Transition Probabilities		
State	1	2
1	0.00207	0.99793
2	0.99986	0.00014

The estimates of the mean vector of the Gaussian distribution for each state are shown in [Figure 13.12](#).

Figure 13.12 Estimate of Mean Vector for Each State

Mu		
State	x	y
1	1.00647	0.99418
2	-0.99304	-1.00787

The estimates of the covariance matrix of the Gaussian distribution for each state are shown in [Figure 13.13](#).

Figure 13.13 Estimate of Covariance Matrix for Each State

Sigma			
State	Variable	x	y
1	x	1.01115	0.00639
	y	0.00639	0.98833
2	x	0.99233	-0.00538
	y	-0.00538	1.00872

The estimates of all parameters are shown in [Figure 13.14](#), which displays columns for the parameter name, estimate value, standard error, *t*-value, and *p*-value.

Figure 13.14 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.002071	0.000711	2.91	0.0036
TPM1_2	0.997929	0.000711	1403.15	<.0001
TPM2_1	0.999863	0.000342	2926.28	<.0001
TPM2_2	0.000137	0.000342	0.40	0.6888
MU1_1	1.006466	0.014228	70.74	<.0001
MU1_2	0.994184	0.014054	70.74	<.0001
MU2_1	-0.993036	0.014096	-70.45	<.0001
MU2_2	-1.007869	0.014217	-70.89	<.0001
SIGMA1_1_1	1.011152	0.020252	49.93	<.0001
SIGMA1_2_1	0.006387	0.014139	0.45	0.6514
SIGMA1_2_2	0.988329	0.019757	50.02	<.0001
SIGMA2_1_1	0.992333	0.019859	49.97	<.0001
SIGMA2_2_1	-0.005378	0.014159	-0.38	0.7041
SIGMA2_2_2	1.008722	0.020194	49.95	<.0001

The log likelihood and information criteria are shown in [Figure 13.15](#).

Figure 13.15 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-28458.72483
AIC	56941.449663
AICC	56941.480904
BIC	57027.973748
HQC	56970.737507

The accuracy of classification through the Gaussian HMM is calculated by the following statements:

```

data gHmmCheck;
  merge DGPone(in=a) mycas.oneDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t x y s correct ds;
run;

%let qFlipState = -1;
data gHmmAccuracy;
  set gHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T.-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gHmmCheck;
  set gHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gHmmAccuracy noobs; run;

```

The accuracy of the Gaussian HMM, which is shown in [Figure 13.16](#), is 99.99%. This means that the Gaussian HMM successfully takes advantage of the sequential information: only 1 data point cannot be correctly classified, compared to about 800 points that cannot be correctly classified by the cluster analysis method without the sequential information.

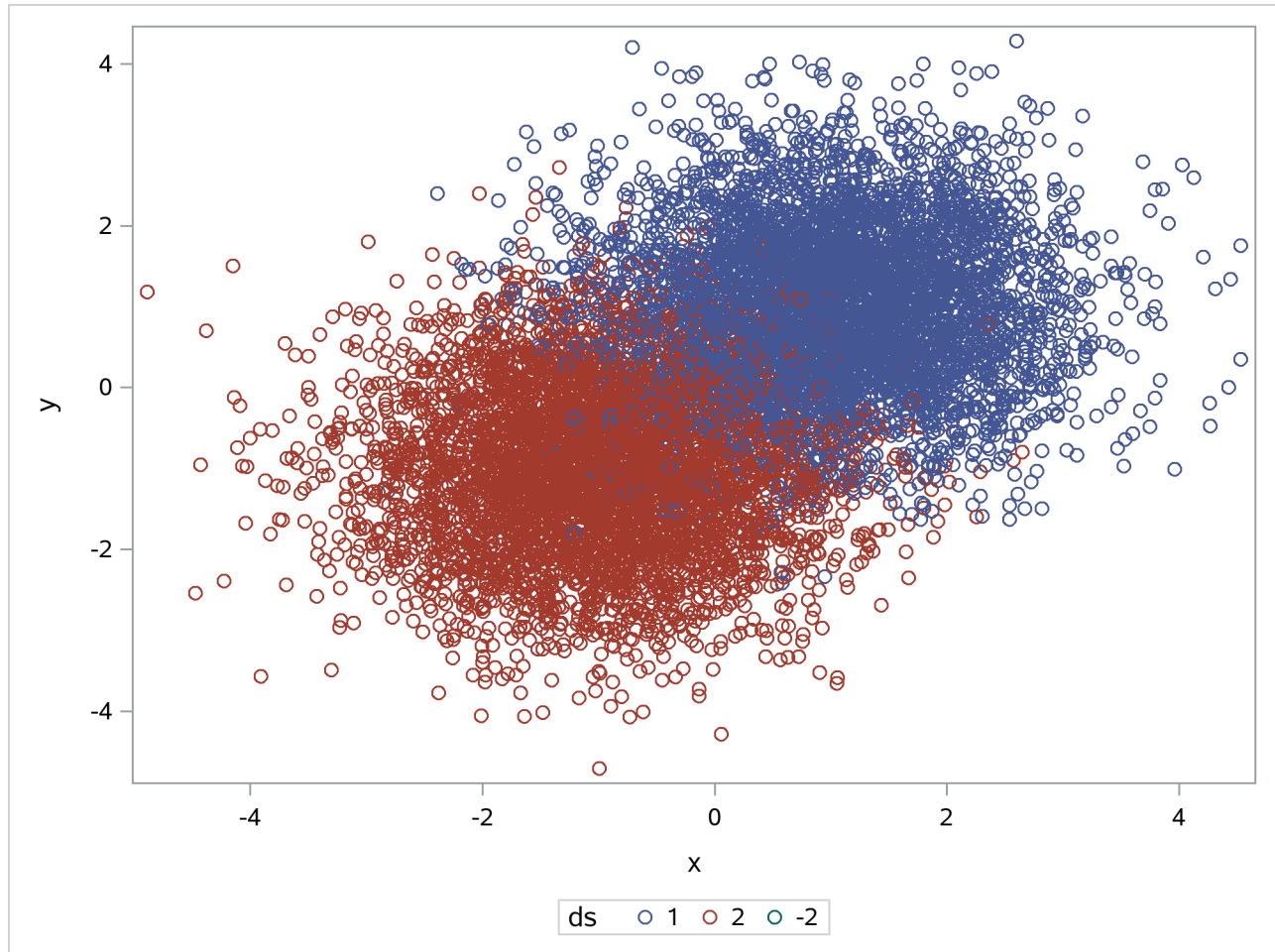
Figure 13.16 Accuracy of Decoding by the Gaussian HMM

accuracy
0.9999

To show how well the data are classified, the previous statements label any incorrectly classified data points with negative state values. In this example, only 1 point is not correctly classified. You plot the classified data points by using the following statements. The plot is shown in [Figure 13.17](#).

```
proc sgplot data=gHmmCheck;
  scatter y=y x=x / group=ds;
run;
```

Figure 13.17 Classified Data Points by Gaussian HMM



Gaussian Hidden Markov Model for Cross-Sectional Time Series Data

Cross-sectional time series data consist of observations over both time and many subjects (such as individuals, objects, firms, or geographical areas). Let $\mathbf{Y}_{n,t} = (y_{n,1t}, \dots, y_{n,pt})'$, where $n = 1, \dots, N$ and $t = 1, \dots, T_n$, denote a p -dimensional vector of random variables at the t th position in the n th section, where T_n is the sample size of the n th section. The $\mathbf{Y}_{n,t}$ follows the Gaussian HMM; that is,

$$\mathbf{Y}_{n,t} | \mathbf{S}_{n,t} \sim N(\mu_{\mathbf{S}_{n,t}}, \Sigma_{\mathbf{S}_{n,t}})$$

where $\mu_{\mathbf{S}_{n,t}}$ and $\Sigma_{\mathbf{S}_{n,t}}$ are mean and covariance parameters, respectively, whose values depend on the state variable $\mathbf{S}_{n,t}$. $\mathbf{S}_{n,t}$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_{n,t} | \mathbf{S}_{n,t-1}, \mathbf{S}_{n,t-2}, \dots, \mathbf{S}_{n,1}) = p(\mathbf{S}_{n,t} | \mathbf{S}_{n,t-1})$$

where $p(.|.)$ denotes the conditional probability. $\mathbf{S}_{n,t}$ is independent of any $\mathbf{S}_{m,s}, m = 1, \dots, N, s = 1, \dots, T_m$, and $m \neq n$. The range of $\mathbf{S}_{n,t}$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_{n,t} = j | \mathbf{S}_{n,t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The initial state probability vector (ISPV), π , of the first state $\mathbf{S}_{n,1}$ is

$$\pi = \{\pi_i = p(\mathbf{S}_{n,1} = i), i = 1, \dots, K\}$$

The Gaussian HMM for cross-sectional time series data, which is the same as the Gaussian HMM for time series data, can be described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{\mu_i, \Sigma_i, i = 1, \dots, K\}$, which consists of parameters that define the state-dependent distribution of observable variables.

Consider a univariate six-state Gaussian HMM that has the following parameter values:

$$\pi = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.95 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.95 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.95 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.95 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}$$

$$\mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \\ \mu_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \\ \Sigma_3 \\ \Sigma_4 \\ \Sigma_5 \\ \Sigma_6 \end{pmatrix} = \begin{pmatrix} 0.0625 \\ 0.25 \\ 1 \\ 4 \\ 16 \\ 64 \end{pmatrix} \right\}$$

The following statements simulate the 10,000-section time series from the previous model to provide test data for the HMM procedure:

```
%let nSections = 10000;
%let T = 100;
%let seed = 1234;
%let nStates = 6;
%let mu1 = 0;
%let sigma1 = 0.0625;
%let mu2 = 0;
%let sigma2 = 0.25;
%let mu3 = 0;
%let sigma3 = 1;
%let mu4 = 0;
%let sigma4 = 4;
```

```

%let mu5 = 0;
%let sigma5 = 16;
%let mu6 = 0;
%let sigma6 = 64;
%let selfTransProb = 0.95;

%macro createCstsTable(tableName);
  data &tableName.;
    do sec = 1 to &nSections.;
      state = ceil(uniform(&seed.)*&nStates.);
      do t = 1 to &T.;
        %do i = 1 %to &nStates.;
          if(state=&i.) then do;
            y = &mu&i.. + sqrt(&sigma&i..)*normal(&seed.);
            output;
          end;
        %end;
        u = uniform(&seed.);
        if(u>&selfTransProb.) then do;
          u = (u-&selfTransProb.)/(1-&selfTransProb.)*(&nStates.-1);
          state=state + ceil(u);
          if(state>&nStates.) then state = state - &nStates.;
        end;
      end;
    end;
  run;
%mend;

%createCstsTable(cstsDGP);

data mycas.cstsDGP;
  set cstsDGP;
run;

```

The following statements estimate the Gaussian HMM:

```

proc hmm data=mycas.cstsDGP
  outstat=mycas.cstsStat;
  id section=sec time=t;
  model y / method=ml type=gaussian nstate=6;
  optimize ALGORITHM=activeset printlevel=3 printIterFreq=1;
  estimate out=mycas.cstsEst;
  evaluate out=mycas.cstsEval;
  decode out=mycas.cstsDecode;
  filter out=mycas.cstsFilter;
  smooth out=mycas.cstsSmooth;
  forecast out=mycas.cstsForecast;
  score outmodel=mycas.cstsModel;
run;

```

The estimates of the ISPV, TPM, mean parameters, and covariance parameters are shown in [Figure 13.18](#), [Figure 13.19](#), [Figure 13.20](#), and [Figure 13.21](#), respectively. They are all very close to the true parameters that are used in the data generating process.

Figure 13.18 Estimates of the Initial State Probability Vector (ISPV)

Initial State Probabilities	
State	Estimation
1	0.16814
2	0.16340
3	0.16812
4	0.16424
5	0.16594
6	0.17016

Figure 13.19 Estimates of the Transition Probability Matrix (TPM)

Transition Probabilities						
State	1	2	3	4	5	6
1	0.94993	0.00963	0.01013	0.01034	0.00970	0.01028
2	0.00992	0.95071	0.01050	0.00906	0.00942	0.01037
3	0.01040	0.00973	0.94935	0.00950	0.01067	0.01035
4	0.01010	0.00976	0.00973	0.94994	0.01052	0.00995
5	0.01015	0.00939	0.01032	0.01017	0.95012	0.00985
6	0.01002	0.00963	0.01049	0.01009	0.00932	0.95045

Figure 13.20 Estimates of the Mean Parameters

Mu	
State	y
1	0.00025
2	-0.00200
3	-0.00254
4	0.00695
5	-0.02130
6	0.01899

Figure 13.21 Estimates of the Covariance Parameters

Sigma		
State	Variable	y
1	y	0.06257
2	y	0.24731
3	y	0.99708
4	y	3.97252
5	y	15.95629
6	y	64.01087

Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. Conditional on a latent variable $\mathbf{S}_t$, $\mathbf{Y}_t$ follows a Gaussian mixture distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim \text{GM}(\{w_{\mathbf{S}_t,1}, \dots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \dots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \dots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\text{GM}(\dots)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are the mean and covariance parameters for the j th Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. The state variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the variable $\mathbf{S}_t$ is unobservable and follows a Markov chain and because $\mathbf{Y}_t$ follows a Gaussian mixture distribution conditional on $\mathbf{S}_t$, the model is called the Gaussian mixture hidden Markov model (GM HMM). The GM HMM is sometimes described as $\{\pi, \mathbf{A}, \mathbf{B}\}$, where $\mathbf{B} \equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \dots, K, j = 1, \dots, M\}$, and it consists of parameters that define the state-dependent distribution of observable variables.

The GM HMM for cross-sectional time series data is similar to the extension from Gaussian HMM to Gaussian HMM for cross-sectional time series data; for more information, see the section “[Gaussian Hidden Markov Model for Cross-Sectional Time Series Data](#)” on page 612.

Consider a bivariate bi-component GM HMM that has the following parameter values. The ISPV and TPM are

$$\pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{pmatrix}$$

At state 1, the two components have weights

$$c_{11} = 0.6, c_{12} = 0.4$$

and the mean and covariance parameters for two components are

$$\mu_{11} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \Sigma_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{12} = \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \Sigma_{12} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

At state 2, the two components have weights

$$c_{21} = 0.7, c_{22} = 0.3$$

and the mean and covariance parameters for two components are

$$\mu_{21} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_{22} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \Sigma_{22} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}$$

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;

data gmDGP;
    retain cd1_1_11 cd1_1_21 cd1_1_22
           cd1_2_11 cd1_2_21 cd1_2_22
           cd2_1_11 cd2_1_21 cd2_1_22
           cd2_2_11 cd2_2_21 cd2_2_22;
```

```

do t = 1 to &T.;
  if(t=1) then do;
    * initial probability distribution;
    p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
    * Cholesky decomposition;
    cd1_1_11 = sqrt(&sigma1_1_11.);
    cd1_1_21 = &sigma1_1_21./cd1_1_11;
    cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
    cd1_2_11 = sqrt(&sigma1_2_11.);
    cd1_2_21 = &sigma1_2_21./cd1_2_11;
    cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
    cd2_1_11 = sqrt(&sigma2_1_11.);
    cd2_1_21 = &sigma2_1_21./cd2_1_11;
    cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
    cd2_2_11 = sqrt(&sigma2_2_11.);
    cd2_2_21 = &sigma2_2_21./cd2_2_11;
    cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
  end;
  else do;
    * transition probability matrix;
    if(lags=1) then p = &a11.;
    else p = 1-&a22.;
  end;
  u = uniform(&seed.);
  if(u<=p) then s=1;
  else s = 2;
  e1 = normal(&seed.); e2 = normal(&seed.);
  if(s=1) then do;
    * choose component;
    u = uniform(&seed.);
    if(u<=&c1_1.) then do;
      * (x,y) ~ N(mu, Sigma) at state 1, component 1;
      x = &mu1_1_1. + cd1_1_11*e1;
      y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
    end;
    else do;
      * (x,y) ~ N(mu, Sigma) at state 1, component 2;
      x = &mu1_2_1. + cd1_2_11*e1;
      y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
    end;
  end;
  else do;
    * choose component;
    u = uniform(&seed.);
    if(u<=&c2_1.) then do;
      * (x,y) ~ N(mu, Sigma) at state 2, component 1;
      x = &mu2_1_1. + cd2_1_11*e1;
      y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
    end;
    else do;
      * (x,y) ~ N(mu, Sigma) at state 2, component 2;
      x = &mu2_2_1. + cd2_2_11*e1;
      y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
    end;
  end;
end;

```

```

        end;
        output;
        lags = s;
    end;
run;

data gmTrain;
    set gmDGP(where=(t<=&T./2));
    keep t x y;
run;

data gmTest;
    set gmDGP(where=(t>&T./2));
    keep t x y;
run;

```

In general, the data generating process is unknown. The data tables gmTrain and gmTest can be seen, but the data table gmDGP cannot be seen.

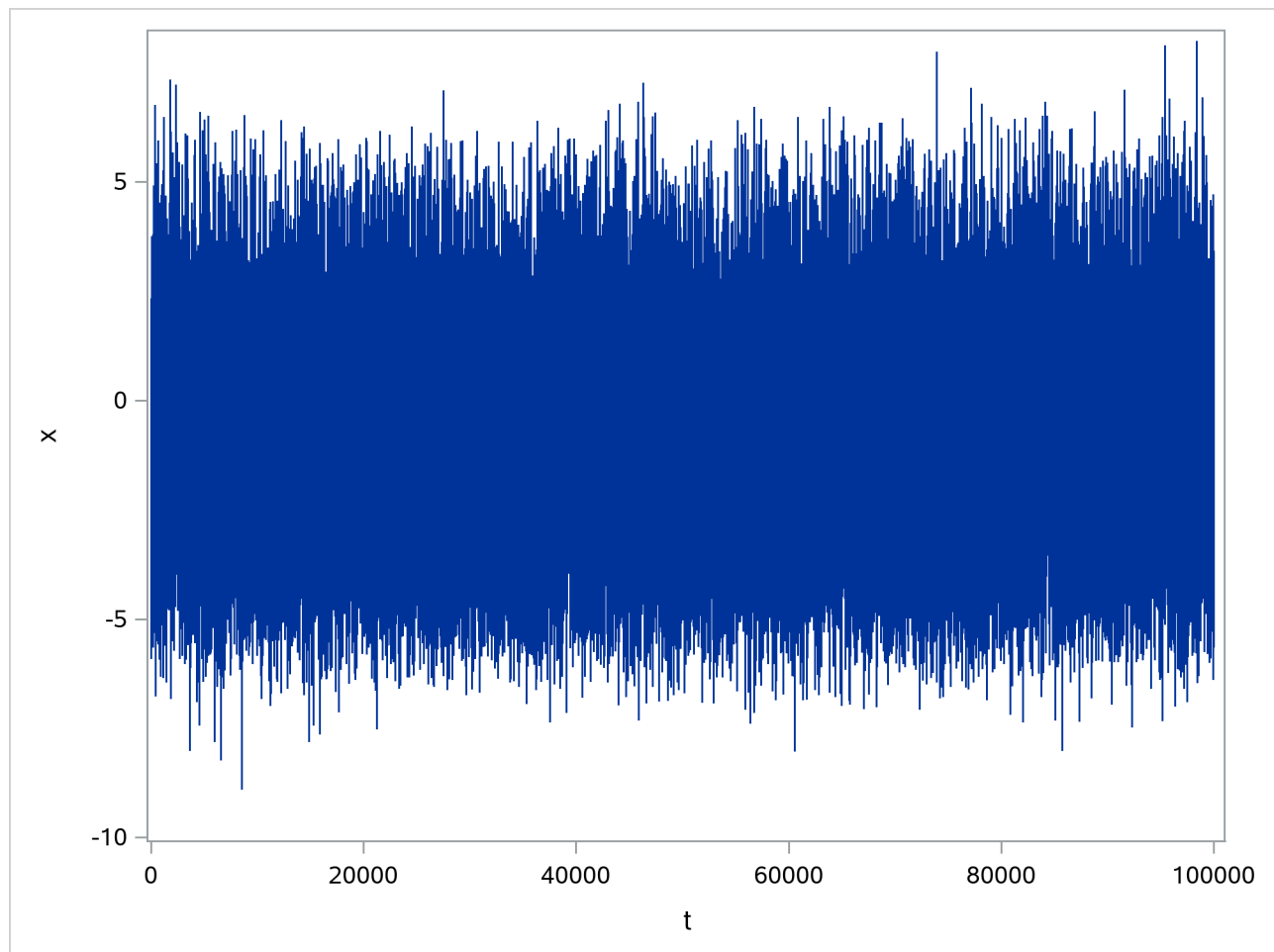
First, assume that the following statements plot the simulated series x that is contained in the data table gmTrain:

```

proc sgplot data=gmTrain;
    series x=t y=x;
run;

```

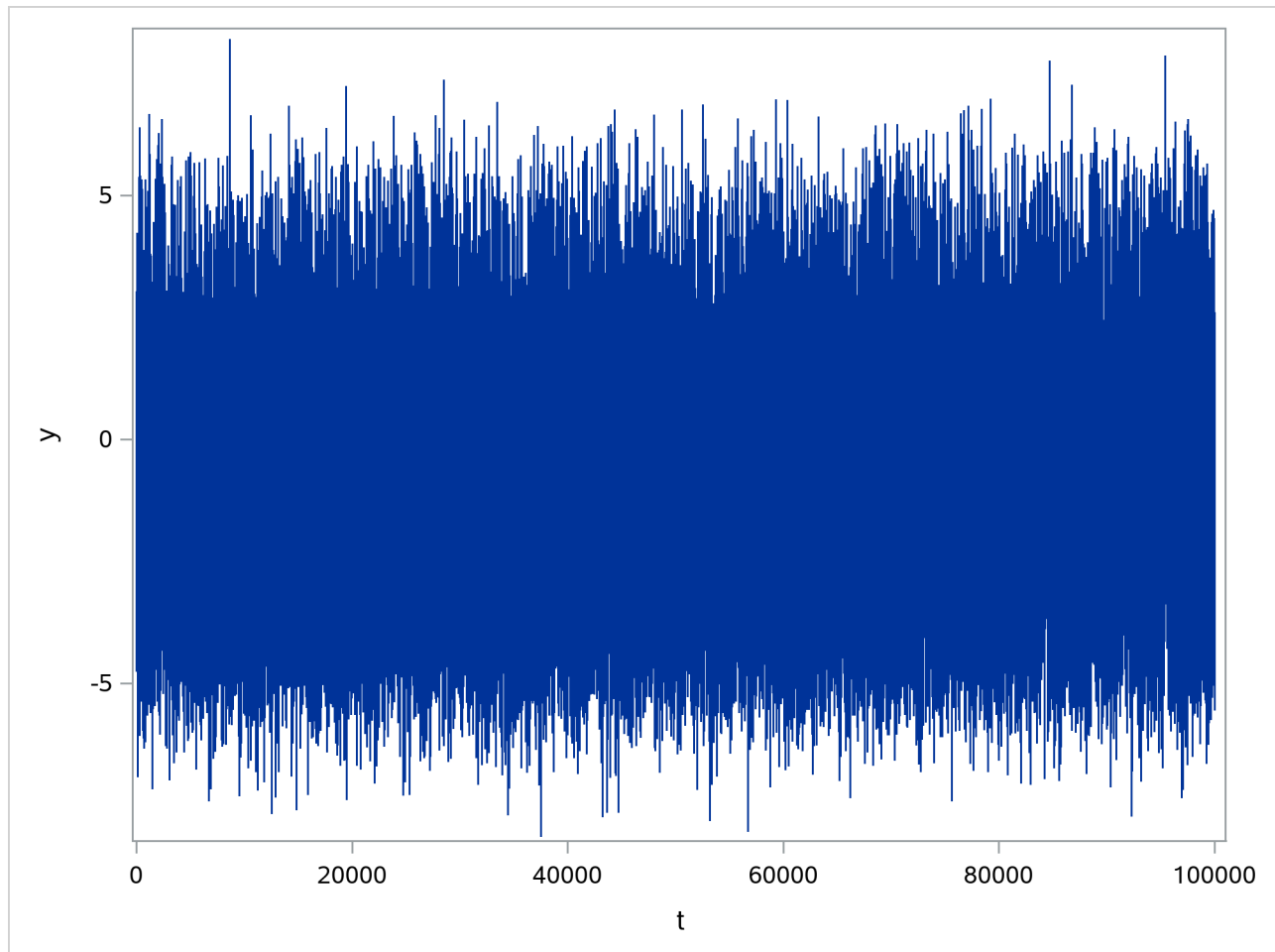
Figure 13.22 shows the plot of x , whose points seem to fall randomly around 0.

Figure 13.22 Plot of Time Series x 

The following statements plot the simulated series y that is contained in the data set `gmTrain`:

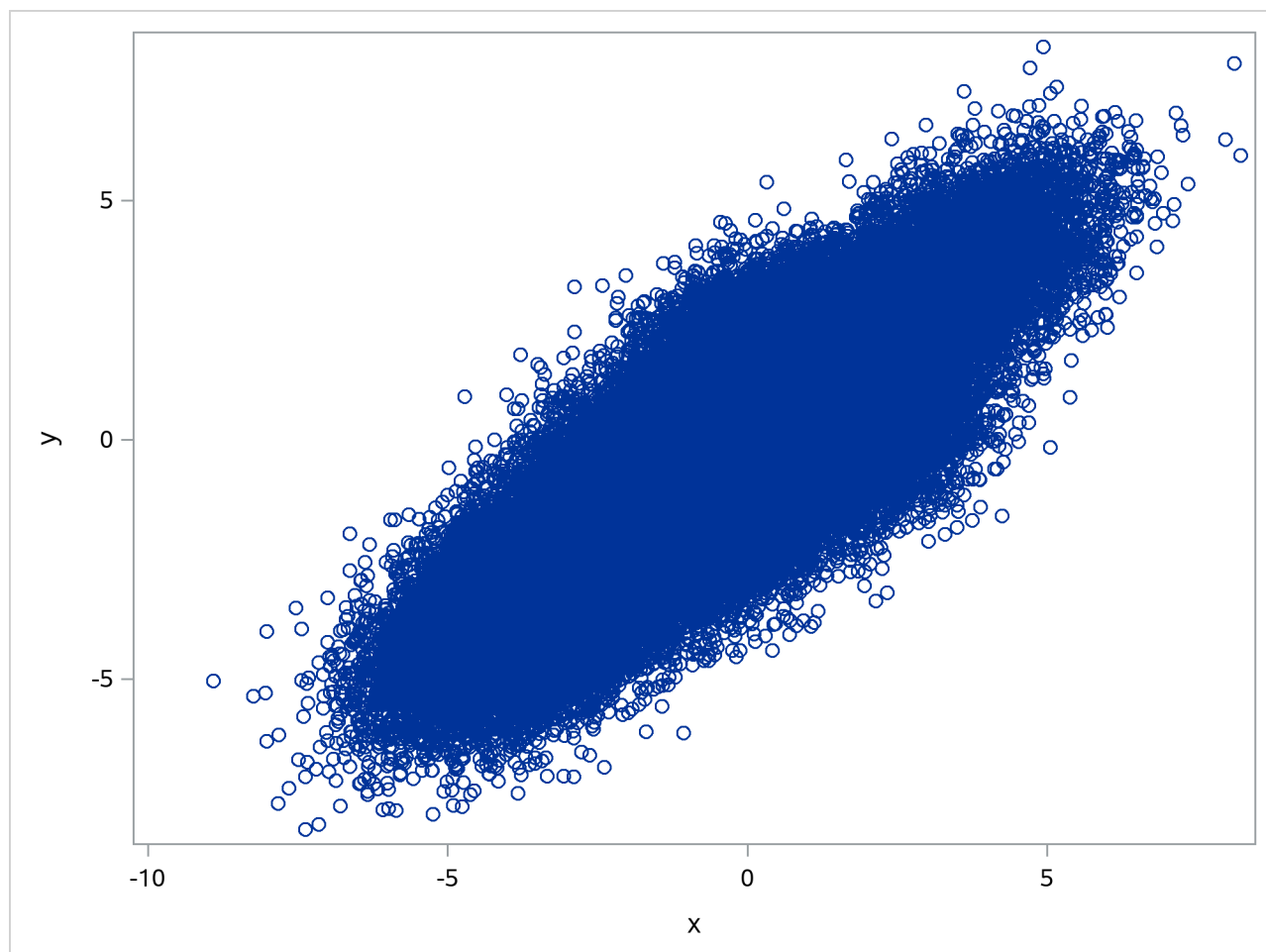
```
proc sgplot data=gmTrain;  
  series x=t y=y;  
run;
```

Figure 13.23 shows the plot of y , whose points also seem to fall randomly around 0.

Figure 13.23 Plot of Time Series y 

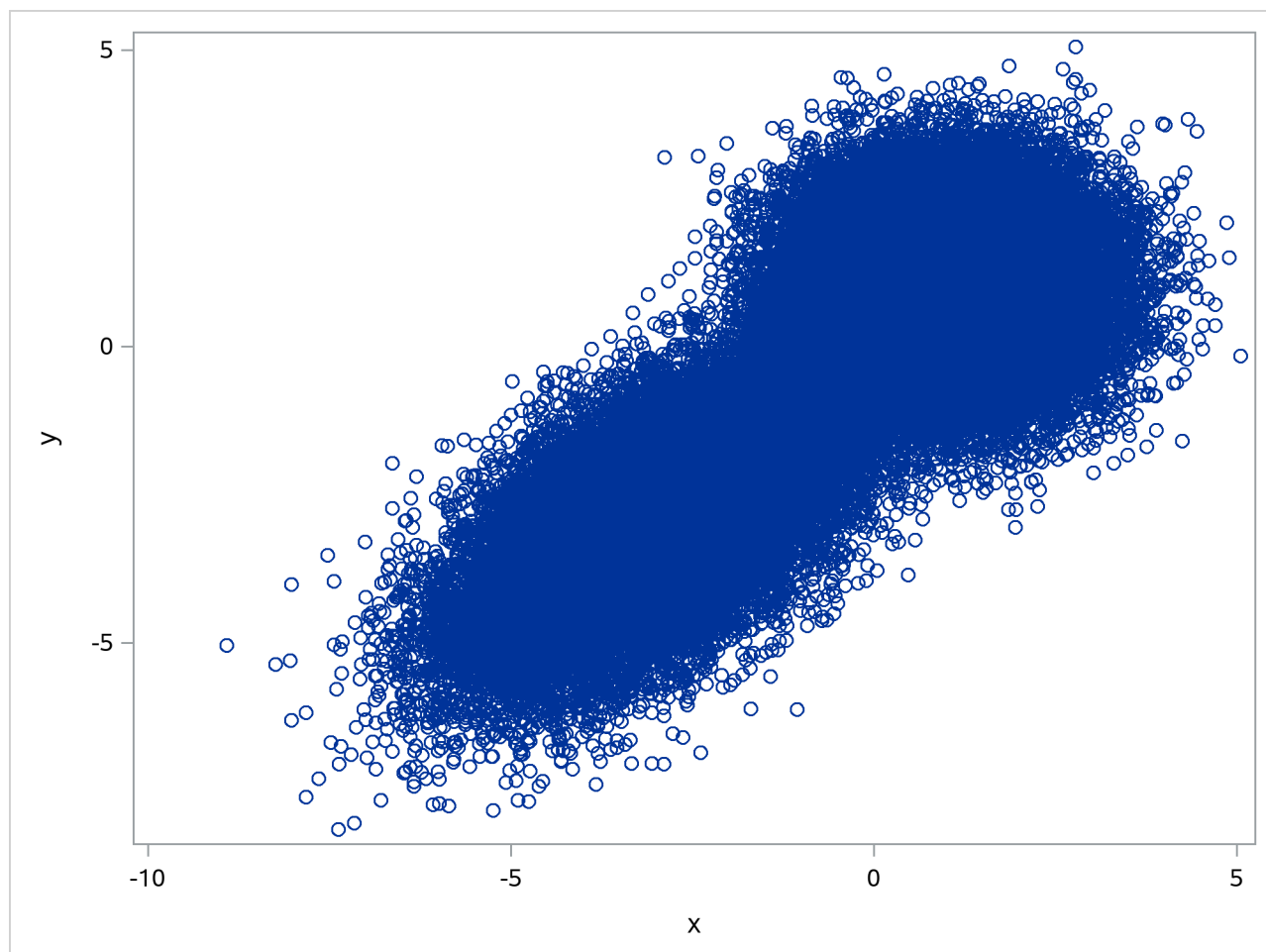
Ignoring the sequential information, the following statements plot the observed points (x, y) that are contained in the data table gmTrain:

```
proc sgplot data=gmTrain;  
    scatter y=y x=x;  
run;
```

Figure 13.24 Unclassified Observations

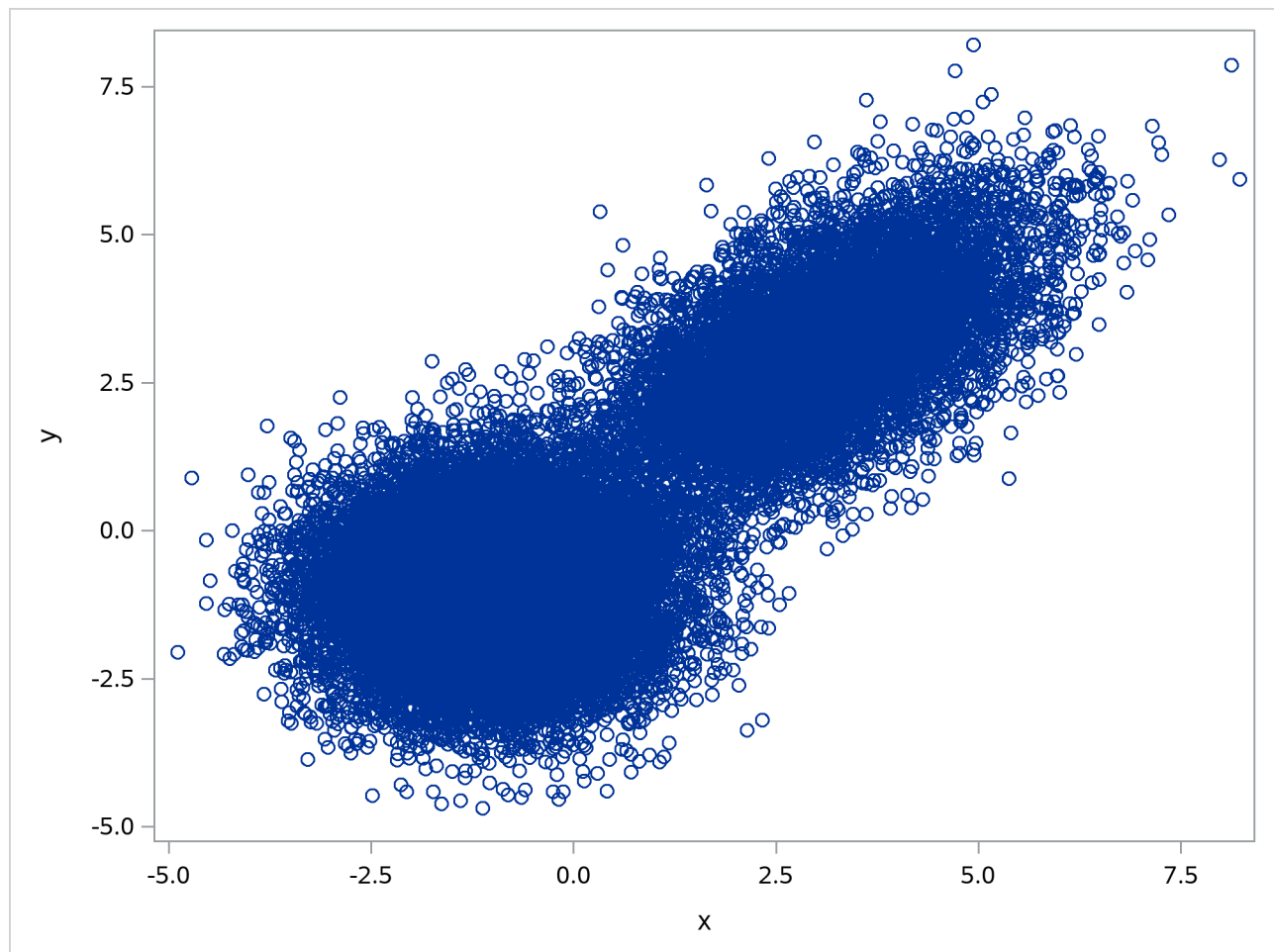
As shown in [Figure 13.24](#), it seems that all points come from one bivariate Gaussian distribution that has a high positive correlation. However, according to the true DGP, some points are from state 1, in which there are two clusters, as shown in [Figure 13.25](#):

```
proc sgplot data=gmdgp (where=(t<=&t./2 and s=1));
  scatter y=y x=x;
run;
```


Figure 13.25 Observations from State 1

Other points are from state 2 in which there are two clusters, as shown in [Figure 13.26](#):

```
proc sgplot data=gmdgp (where=(t<=&T./2 and s=2));
  scatter y=y x=x;
run;
```

Figure 13.26 Observations from State 2

Without considering the sequential order, it is impossible to distinguish the two states.

Before you estimate any model, you should make a copy of the client-side data on the server, as in the following DATA steps:

```
data mycas.gmTrain; set gmTrain; run;  
data mycas.gmTest; set gmTest; run;
```

The following statements estimate the bistate bi-component GM HMM. The type of model is specified in the TYPE= option in the MODEL statement. The number of states is specified in the NSTATE= option in the MODEL statement. The number of components for each state is specified in the NCOMPONENT= option in the MODEL statement. The OUTMODEL= option in the SCORE statement outputs the model information for future scoring on new data table. The LABELSWITCH=(SORT=NONE) option in the PROC HMM statement suppresses the relabeling of states and components; for more information about the label switching problem, see the section “[Label Switching Problem](#)” on page 694.

```
proc hmm data=mycas.gmTrain labelSwitch=(sort=none);
  id time=t;
  model x y / type=gaussianMixture nstate=2 ncomponent=2;
  optimize printLevel=3 printIterFreq=1;
  score outmodel=mycas.gmModel;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The number of observations and model information are shown in [Figure 13.27](#).

Figure 13.27 Number of Observations and Model Information

Observation Information	
Number of Observations	100000
Number of Missing Observations	0

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2

The default method, which is the maximum likelihood method, is applied; this is a nonlinear optimization problem. The initial parameter values and the objective function value at the start of the optimization are shown in [Figure 13.28](#).

Figure 13.28 Initial Parameter Values and Objective Function Value

Optimization Start	
Parameter	Estimate
TPM1_1	0.500000
TPM1_2	0.500000
TPM2_1	0.500000
TPM2_2	0.500000
MCP1_1	0.500000
MCP1_2	0.500000
MCP2_1	0.500000
MCP2_2	0.500000
MU1_1_1	0.416654
MU1_1_2	0.961871
MU1_2_1	-3.422684
MU1_2_2	-3.125473
MU2_1_1	2.963828
MU2_1_2	1.590644
MU2_2_1	-0.815362
MU2_2_2	-2.039325
SIGMA1_1_1_1	1.354409
SIGMA1_1_2_1	0.762768
SIGMA1_1_2_2	1.393576
SIGMA1_2_1_1	0.961355
SIGMA1_2_2_1	0.661717
SIGMA1_2_2_2	1.581451
SIGMA2_1_1_1	1.025889
SIGMA2_1_2_1	1.399630
SIGMA2_1_2_2	2.884377
SIGMA2_2_1_1	0.749535
SIGMA2_2_2_1	0.265184
SIGMA2_2_2_2	0.633097

Initial Value of Objective Function		-401529.3793
-------------------------------------	--	--------------

The details of the optimization algorithm are shown in [Figure 13.29](#).

Figure 13.29 Optimization Algorithm Details

Algorithm Settings	
Maximum Number of Iterations	128
Tolerance for Optimality Error	1E-6
Tolerance for Infeasibility	1E-6
Maximum Allowed Time (seconds)	1.797693E308
Maximum Magnitude of Objective Function Value	1E20
Random Seed	1
Solution Type	1
Multi-Start Globalization Scheme	0
Multi-Start Globalization Bound Range	2

The iterations of the optimization process are shown in Figure 13.30.

Figure 13.30 Iterations of the Optimization Process

Iteration	Objective Function	Iteration History		Function Calls	Gradient Calls	Hessian Calls
		Optimality Error	Infeasibility			
1	-401529.3793	2887.16466	0.00000	2	1	0
2	-389554.1158	2133.36152	0.00000	10	9	1
3	-386179.6546	1199.52355	0.00000	19	18	2
4	-371985.2011	848.19186	0.00000	29	28	3
5	-367516.0600	249.88933	0.00000	38	37	4
6	-366322.8416	161.33937	0.00000	49	48	5
7	-365982.2352	46.74362	0.00000	59	58	6
8	-365945.5123	5.29758	0.00000	69	68	7
9	-365945.0948	0.06672	0.00000	79	78	8
10	-365945.0946	0.00000	0.00000	89	88	9

The final parameter values and the objective function value at the end of the optimization are shown in Figure 13.31.

Figure 13.31 Final Parameter Values and Objective Function Value

Optimization Results	
Parameter	Estimate
TPM1_1	0.979636
TPM1_2	0.020364
TPM2_1	0.049729
TPM2_2	0.950271
MCP1_1	0.600332
MCP1_2	0.399668
MCP2_1	0.297811
MCP2_2	0.702189
MU1_1_1	0.993524
MU1_1_2	0.996707
MU1_2_1	-3.003906
MU1_2_2	-3.005379
MU2_1_1	3.005306
MU2_1_2	3.009691
MU2_2_1	-0.997604
MU2_2_2	-1.005076
SIGMA1_1_1_1	1.000305
SIGMA1_1_2_1	0.007588
SIGMA1_1_2_2	1.003672
SIGMA1_2_1_1	1.704270
SIGMA1_2_2_1	1.003413
SIGMA1_2_2_2	1.688212
SIGMA2_1_1_1	1.712932
SIGMA2_1_2_1	1.014865
SIGMA2_1_2_2	1.704181
SIGMA2_2_1_1	1.015895
SIGMA2_2_2_1	0.012899
SIGMA2_2_2_2	1.006142
Final Value of Objective Function	
-365945.0946	

The estimates of the initial state probability vector (ISPV) are shown in [Figure 13.32](#).

Figure 13.32 Estimates of Initial State Probability Vector

Initial State Probabilities	
State	Estimation
1	0.70947
2	0.29053

The estimates of the transition probability matrix (TPM) are shown in [Figure 13.33](#).

Figure 13.33 Estimates of Transition Probability Matrix

Transition Probabilities		
State	1	2
1	0.97964	0.02036
2	0.04973	0.95027

The estimates of the mixture component probabilities (MCP) are shown in [Figure 13.34](#).

Figure 13.34 Estimates of Mixture Component Probabilities

Mixture Component Probabilities		
Component		
State	1	2
1	0.60033	0.39967
2	0.29781	0.70219

The estimates of the mean vector of the Gaussian distribution for each component at each state are shown in [Figure 13.35](#).

Figure 13.35 Estimate of Mean Vector for Each Component at Each State

Mu				
State	Component		x	y
1	1		0.99352	0.99671
	2		-3.00391	-3.00538
2	1		3.00531	3.00969
	2		-0.99760	-1.00508

The estimates of the covariance matrix of the Gaussian distribution for each component at each state are shown in [Figure 13.36](#).

Figure 13.36 Estimate of Covariance Matrix for Each Component at Each State

Sigma				
State	Component	Variable	x	y
1	1	x	1.00031	0.00759
		y	0.00759	1.00367
	2	x	1.70427	1.00341
		y	1.00341	1.68821
2	1	x	1.71293	1.01486
		y	1.01486	1.70418
	2	x	1.01589	0.01290
		y	0.01290	1.00614

The estimates of all parameters are shown in [Figure 13.37](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The state labels (“1” and “2” in this example) and the component labels (“1” and “2” in this example) might exchange randomly, compared to the true DGP. In the

preceding estimation, the state labels are the same as the labels in the DGP. The component labels for state 1 are also the same as the ones in the DGP, but the component labels for state 2 do exchange! Compared to the parameter values in the DGP, the estimates are very accurate.

Figure 13.37 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.979636	0.000600	1632.32	<.0001
TPM1_2	0.020364	0.000600	33.93	<.0001
TPM2_1	0.049729	0.001422	34.98	<.0001
TPM2_2	0.950271	0.001422	668.37	<.0001
MCP1_1	0.600332	0.001987	302.17	<.0001
MCP1_2	0.399668	0.001987	201.17	<.0001
MCP2_1	0.297811	0.002966	100.42	<.0001
MCP2_2	0.702189	0.002966	236.78	<.0001
MU1_1_1	0.993524	0.005202	191.00	<.0001
MU1_1_2	0.996707	0.005225	190.77	<.0001
MU1_2_1	-3.003906	0.009259	-324.43	<.0001
MU1_2_2	-3.005379	0.009199	-326.71	<.0001
MU2_1_1	3.005306	0.017586	170.89	<.0001
MU2_1_2	3.009691	0.017492	172.06	<.0001
MU2_2_1	-0.997604	0.007672	-130.04	<.0001
MU2_2_2	-1.005076	0.007645	-131.47	<.0001
SIGMA1_1_1_1	1.000305	0.007559	132.33	<.0001
SIGMA1_1_2_1	0.007588	0.005615	1.35	0.1765
SIGMA1_1_2_2	1.003672	0.007625	131.63	<.0001
SIGMA1_2_1_1	1.704270	0.018040	94.47	<.0001
SIGMA1_2_2_1	1.003413	0.015372	65.27	<.0001
SIGMA1_2_2_2	1.688212	0.017808	94.80	<.0001
SIGMA2_1_1_1	1.712932	0.033960	50.44	<.0001
SIGMA2_1_2_1	1.014865	0.028939	35.07	<.0001
SIGMA2_1_2_2	1.704181	0.033648	50.65	<.0001
SIGMA2_2_1_1	1.015895	0.011295	89.94	<.0001
SIGMA2_2_2_1	0.012899	0.008289	1.56	0.1197
SIGMA2_2_2_2	1.006142	0.011248	89.45	<.0001

The log likelihood and information criteria are shown in [Figure 13.38](#).

Figure 13.38 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-365945.0946
AIC	731942.18929
AICC	731942.20333
BIC	732189.52535
HQC	732017.24975

The estimates from the data table `gmTrain` are used to score the testing data table `gmTest` to check the predictability in the following statements:

```
proc hmm data=mycas.gmTest;
  score inmodel=mycas.gmModel;
  decode out=mycas.gmDecode;
run;
```

The accuracy of classification through the GM HMM is calculated by the following statements:

```
data gmHmmCheck;
  merge gmDGP(in=a where=(t>&T./2)) mycas.gmDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t x y s correct ds;
run;

%let qFlipState = -1;
data gmHmmAccuracy;
  set gmHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T./2-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gmHmmCheck;
  set gmHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gmHmmAccuracy noobs; run;
```

The accuracy of the GM HMM, which is shown in [Figure 13.39](#), is about 98%. This means that the GM HMM successfully takes advantage of the sequential information.

Figure 13.39 Accuracy of Decoding by the GM HMM

accuracy
0.97835

Now, consider the GM HMM for cross-sectional time series data. Each section's data follow the same GM HMM as described previously. The following statements generate the data table for analysis. There are 200 sections in the data table, and each section of time series consist of 1,000 observations.

```
%let a11 = 0.98;
%let a22 = 0.95;
%let c1_1 = 0.6;
%let c1_2 = 0.4;
%let c2_1 = 0.7;
%let c2_2 = 0.3;
%let mu1_1_1 = 1;
%let mu1_1_2 = 1;
%let mu1_2_1 = -3;
%let mu1_2_2 = -3;
%let sigma1_1_11 = 1;
%let sigma1_1_21 = 0;
%let sigma1_1_22 = 1;
%let sigma1_2_11 = 1.69;
%let sigma1_2_21 = 1;
%let sigma1_2_22 = 1.69;
%let mu2_1_1 = -1;
%let mu2_1_2 = -1;
%let mu2_2_1 = 3;
%let mu2_2_2 = 3;
%let sigma2_1_11 = 1;
%let sigma2_1_21 = 0;
%let sigma2_1_22 = 1;
%let sigma2_2_11 = 1.69;
%let sigma2_2_21 = 1;
%let sigma2_2_22 = 1.69;
%let seed = 1234;
%let T = 200000;
%let nSections = 200;

data gmDGP;
  retain cd1_1_11 cd1_1_21 cd1_1_22
         cd1_2_11 cd1_2_21 cd1_2_22
         cd2_1_11 cd2_1_21 cd2_1_22
         cd2_2_11 cd2_2_21 cd2_2_22;
  do t = 1 to &T.;
    sec = ceil(t*&nSections./&T.);
    if(t=1) then do;
      * initial probability distribution;
      p = (1-&a22.)/((1-&a11.)+(1-&a22.));;
      * Cholesky decomposition;
      cd1_1_11 = sqrt(&sigma1_1_11.);
      cd1_1_21 = &sigma1_1_21./cd1_1_11;
      cd1_1_22 = sqrt(&sigma1_1_22.-cd1_1_21*cd1_1_21);
      cd1_2_11 = sqrt(&sigma1_2_11.);
      cd1_2_21 = &sigma1_2_21./cd1_2_11;
      cd1_2_22 = sqrt(&sigma1_2_22.-cd1_2_21*cd1_2_21);
      cd2_1_11 = sqrt(&sigma2_1_11.);
      cd2_1_21 = &sigma2_1_21./cd2_1_11;
      cd2_1_22 = sqrt(&sigma2_1_22.-cd2_1_21*cd2_1_21);
      cd2_2_11 = sqrt(&sigma2_2_11.);
      cd2_2_21 = &sigma2_2_21./cd2_2_11;
```

```

        cd2_2_22 = sqrt(&sigma2_2_22.-cd2_2_21*cd2_2_21);
    end;
    else do;
        * transition probability matrix;
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
        * choose component;
        u = uniform(&seed.);
        if(u<=&c1_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 1;
            x = &mu1_1_1. + cd1_1_11*e1;
            y = &mu1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
        end;
        else do;
            * (x,y) ~ N(mu, Sigma) at state 1, component 2;
            x = &mu1_2_1. + cd1_2_11*e1;
            y = &mu1_2_2. + cd1_2_21*e1+cd1_2_22*e2;
        end;
    end;
    else do;
        * choose component;
        u = uniform(&seed.);
        if(u<=&c2_1.) then do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 1;
            x = &mu2_1_1. + cd2_1_11*e1;
            y = &mu2_1_2. + cd2_1_21*e1+cd2_1_22*e2;
        end;
        else do;
            * (x,y) ~ N(mu, Sigma) at state 2, component 2;
            x = &mu2_2_1. + cd2_2_11*e1;
            y = &mu2_2_2. + cd2_2_21*e1+cd2_2_22*e2;
        end;
    end;
    output;
    lags = s;
end;
run;

data gmTrain;
    set gmDGP(where=(sec<=&nSections./2));
    keep sec t x y;
run;

data gmTest;
    set gmDGP(where=(sec>&nSections./2));
    keep sec t x y;
run;

```

The following statements estimate the GM HMM for the cross-sectional time series data after making the copies of the data tables on the server. The SECTION= option in the ID statement requests that the cross-sectional time series data be analyzed.

```
data mycas.gmTrain; set gmTrain; run;
data mycas.gmTest; set gmTest; run;

proc hmm data=mycas.gmTrain labelSwitch=(sort=none);
  id time=t section=sec;
  model x y / type=gaussianMixture nstate=2 ncomponent=2;
  optimize printLevel=3 printIterFreq=1;
  score outmodel=mycas.gmModel;
  filter out=mycas.gmFilter;
  evaluate out=mycas.gmEval;
run;
```

The model information is shown in [Figure 13.40](#). There are 100 sections in the data.

Figure 13.40 Model Information

Model Information	
Type of Model	Gaussian Mixture HMM
Stationary	Yes
Number of States	2
Number of Components for a State	2
Number of Dependent Variables	2
Number of Sections	100

The estimates of all parameters are shown in [Figure 13.41](#); the estimates are very close to the parameter values in the DGP.

Figure 13.41 Parameter Estimates

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.979697	0.000598	1639.23	<.0001
TPM1_2	0.020303	0.000598	33.97	<.0001
TPM2_1	0.049991	0.001419	35.23	<.0001
TPM2_2	0.950009	0.001419	669.51	<.0001
MCP1_1	0.600330	0.001987	302.13	<.0001
MCP1_2	0.399670	0.001987	201.15	<.0001
MCP2_1	0.297864	0.002966	100.41	<.0001
MCP2_2	0.702136	0.002966	236.70	<.0001
MU1_1_1	0.993469	0.005202	190.97	<.0001
MU1_1_2	0.996750	0.005224	190.80	<.0001
MU1_2_1	-3.004280	0.009259	-324.45	<.0001
MU1_2_2	-3.005705	0.009202	-326.64	<.0001
MU2_1_1	3.004517	0.017596	170.75	<.0001
MU2_1_2	3.008912	0.017510	171.84	<.0001
MU2_2_1	-0.997409	0.007674	-129.97	<.0001
MU2_2_2	-1.005153	0.007644	-131.50	<.0001
SIGMA1_1_1_1	1.000332	0.007560	132.32	<.0001
SIGMA1_1_2_1	0.007321	0.005614	1.30	0.1922
SIGMA1_1_2_2	1.003283	0.007624	131.60	<.0001
SIGMA1_2_1_1	1.703717	0.018034	94.47	<.0001
SIGMA1_2_2_1	1.002882	0.015368	65.26	<.0001
SIGMA1_2_2_2	1.687759	0.017806	94.78	<.0001
SIGMA2_1_1_1	1.713186	0.033969	50.43	<.0001
SIGMA2_1_2_1	1.015588	0.028964	35.06	<.0001
SIGMA2_1_2_2	1.705412	0.033689	50.62	<.0001
SIGMA2_2_1_1	1.015835	0.011301	89.89	<.0001
SIGMA2_2_2_1	0.012931	0.008291	1.56	0.1188
SIGMA2_2_2_2	1.005816	0.011243	89.46	<.0001

You can apply the estimates from the data table gmTrain to the testing data table gmTest to check the predictability in the following statements:

```
proc hmm data=mycas.gmTest;
  score inmodel=mycas.gmModel;
  decode out=mycas.gmDecode;
  filter out=mycas.gmFilter2;
  evaluate out=mycas.gmEval2;
run;

data gmHmmCheck;
  merge gmDGP(in=a where=(sec>&nSections./2)) mycas.gmDecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep sec t x y s correct ds;
```

```

run;

%let qFlipState = -1;
data gmHmmAccuracy;
  set gmHmmCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T./2-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data gmHmmCheck;
  set gmHmmCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = gmHmmAccuracy noobs; run;

```

The accuracy of the GM HMM for cross-sectional time series data, which is shown in Figure 13.42, is also around 98%.

Figure 13.42 Accuracy of Decoding by the GM HMM for Cross-Sectional Time Series Data

accuracy
0.97827

Regime-Switching Regression Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(.|.)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the regression parameters depend on the state, the $\mathbf{Y}_t$ follows different regressions in different regimes; hence, this type of model is called the regime-switching regression model.

Consider a univariate regime-switching regression model that has two regimes and two regressors (an intercept and an exogenous variable):

$$y_t = \begin{cases} x_t + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ 4 + 1.5x_t + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

```
%let x_lb = -8;
%let x_ub = 0;
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let const1_1 = 0;
%let x11_0_1_1 = 1;
%let cov1_11 = 2.56;
%let const2_1 = 4;
%let x12_0_1_1 = 1.5;
%let cov2_11 = 4;
%let T = 4000;
%let seed = 1234;

data rsregdgp;
  retain cd1_11 cd2_11;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &pi1.;
      /* Cholesky decomposition of COV1 */
```

```

        cd1_11 = sqrt(&cov1_11.);
        /* Cholesky decomposition of COV2 */
        cd2_11 = sqrt(&cov2_11.);
    end;
    else do;
        /* transition probability matrix */
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    x = &x_lb. + (&x_ub. - &x_lb.)*uniform(&seed.);
    e = normal(&seed.);
    if(s=1) then do;
        /* y ~ N(beta1_0+beta1_1*x, Sigma1) at state 1 */
        y = &const1_1. + &x11_0_1_1 * x + cd1_11*e;
    end;
    else do;
        /* y ~ N(beta2_0+beta2_1*x, Sigma2) at state 2 */
        y = &const2_1. + &x12_0_1_1 * x + cd2_11*e;
    end;
    output;
    lags = s;
end;
run;

data rsreg;
    set rsregdgp;
    keep t y x;
run;

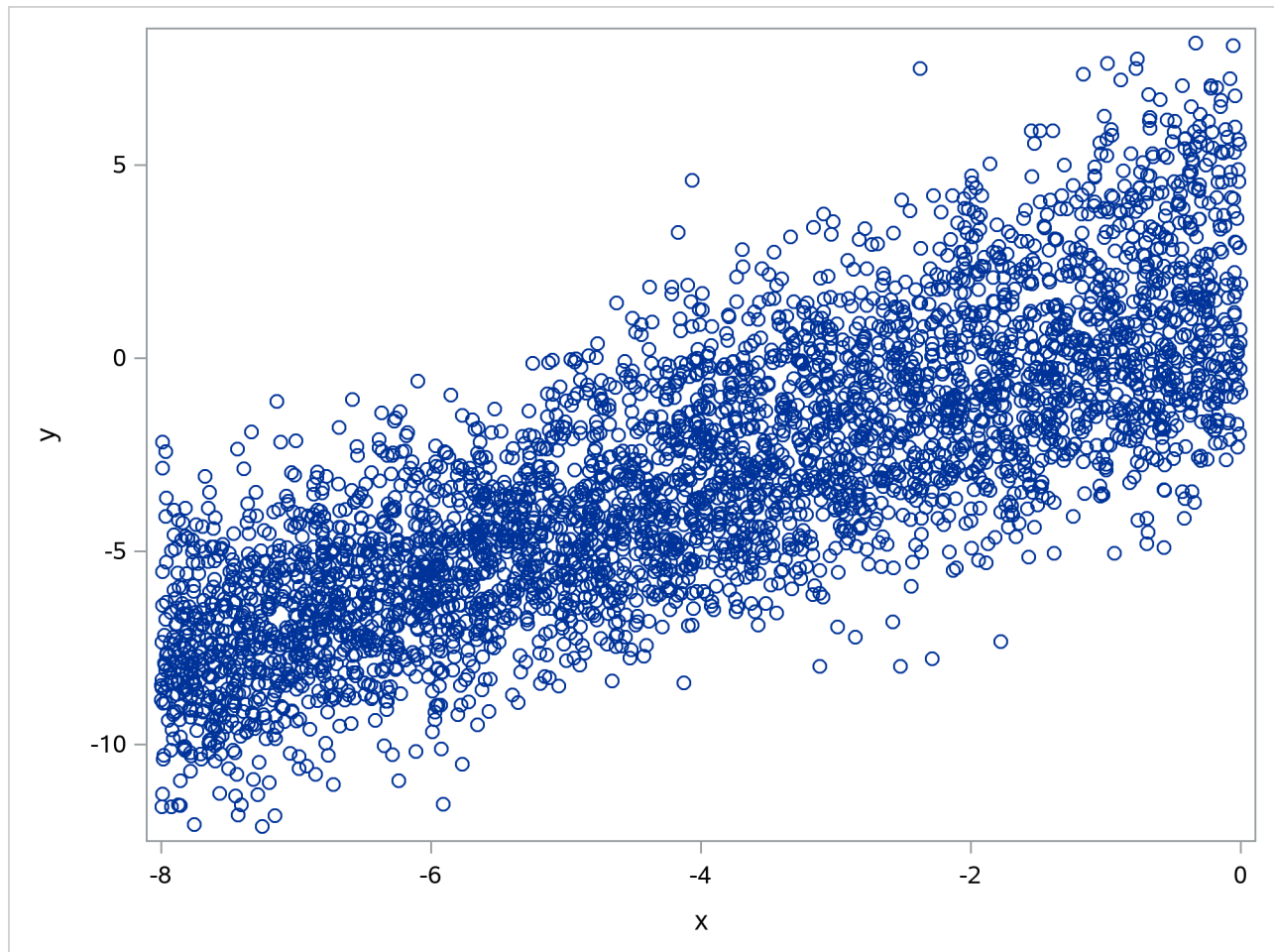
```

The following statements create the scatter plot in [Figure 13.43](#):

```

proc sgplot data=rsreg;
    scatter y=y x=x;
run;

```


Figure 13.43 Scatter Plot of Data Points

As shown in Figure 13.43, when the time dependence is ignored, it is difficult—if not impossible—to distinguish the two regimes.

The following code uploads the data table and estimates the regime-switching regression model:

```
data mycas.rsreg; set rsreg; run;
proc hmm data=mycas.rsreg;
  id time=t;
  model y = x / type=reg nstate=2 method=ml;
  optimize algorithm=activeset maxiter=256 printLevel=3 printIterFreq=1;
  learn out=mycas.mylearn;
  filter out=mycas.myfilter;
  smooth out=mycas.mysmooth;
  decode out=mycas.mydecode;
  evaluate out=mycas.myeval;
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The estimates of all parameters are shown in [Figure 13.44](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 13.44 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.950779	0.006259	151.91	<.0001
TPM1_2	0.049221	0.006259	7.86	<.0001
TPM2_1	0.057042	0.007335	7.78	<.0001
TPM2_2	0.942958	0.007335	128.56	<.0001
CONST1_1	-0.058860	0.079933	-0.74	0.4616
CONST2_1	4.105147	0.113280	36.24	<.0001
XL1_0_1_1	0.986513	0.016304	60.51	<.0001
XL2_0_1_1	1.513899	0.022254	68.03	<.0001
COV1_1_1	2.563899	0.094248	27.20	<.0001
COV2_1_1	3.839914	0.143172	26.82	<.0001

The accuracy of classification through the regime-switching regression model is calculated by the following statements:

```
data rsregCheck;
  merge rsregdgp(in=a) mycas.mydecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t x y s correct ds;
run;

%let qFlipState = -1;
data rsregAccuracy;
  set rsregCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T.-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data rsregCheck;
  set rsregCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;
```

```
proc print data = rsregAccuracy noobs; run;
```

The accuracy of the regime-switching regression model, which is shown in [Figure 13.45](#), is close to 90%. This means that the regime-switching regression model successfully distinguishes the two regimes.

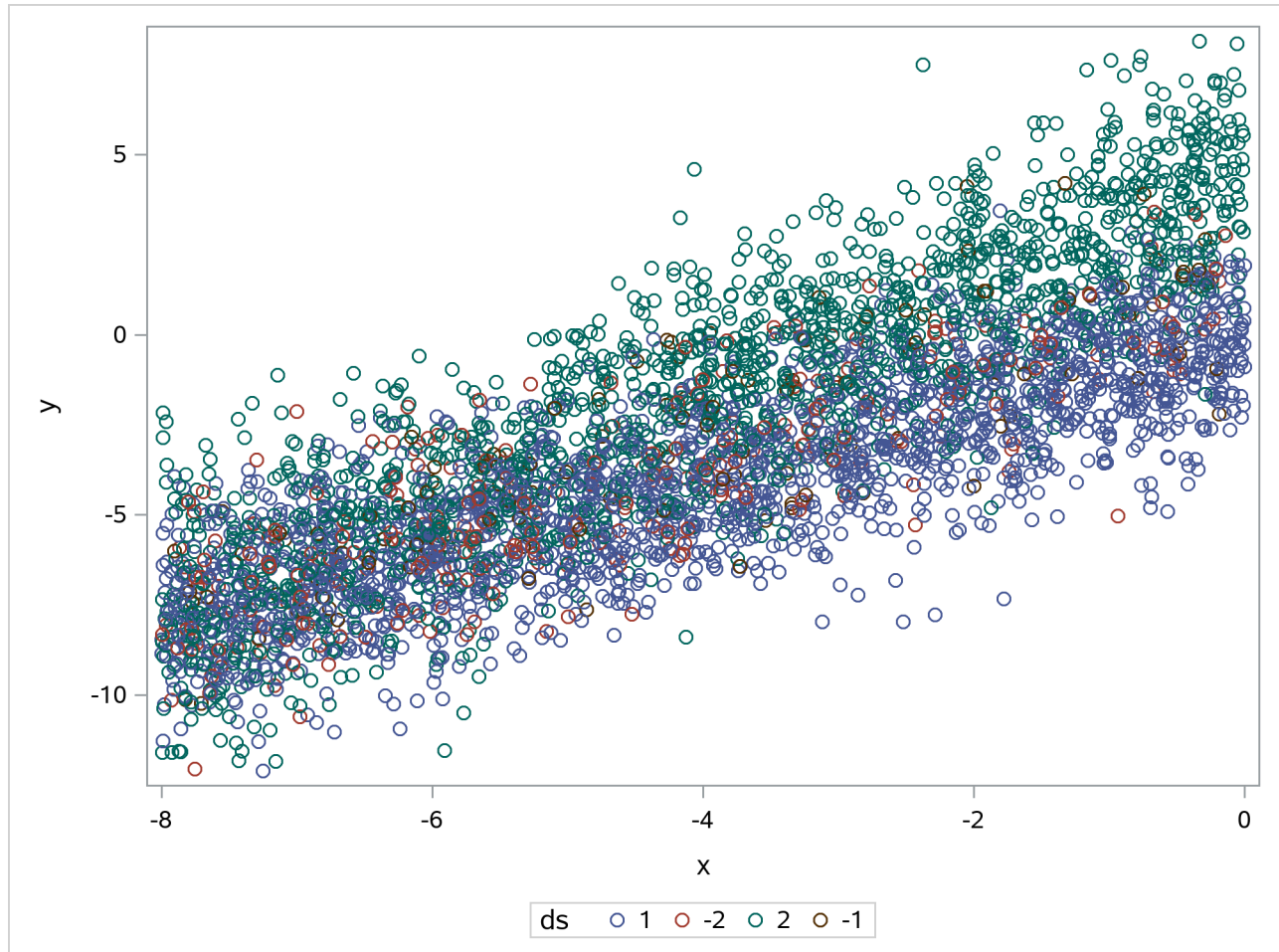
Figure 13.45 Accuracy of Decoding by the Regime-Switching Regression Model

accuracy
0.89925

The following statements create the scatter plot in [Figure 13.46](#) to show how well the data points are classified:

```
proc sgplot data=rsregCheck;
  scatter y=y x=x / group=ds;
run;
```

Figure 13.46 Scatter Plot of Data Points



Regime-Switching Autoregression Model

Let $\mathbf{Y}_t = (y_{1t}, \dots, y_{pt})'$, $t = 1, \dots, T$, denote a p -dimensional time series vector of random variables. The $\mathbf{Y}_t$ can be modeled in the regression

$$\mathbf{Y}_t = \beta_{\mathbf{S}_t} z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, the regressor z_t is observable and contains the lagged dependent variables, the latent variable $\mathbf{S}_t$ is the so-called state, and $\beta_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters whose values depend on the state $\mathbf{S}_t$. The variable $\mathbf{S}_t$ follows the first-order Markov chain; that is,

$$p(\mathbf{S}_t | \mathbf{S}_{t-1}, \mathbf{S}_{t-2}, \dots, \mathbf{S}_1) = p(\mathbf{S}_t | \mathbf{S}_{t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. The range of $\mathbf{S}_t$ is a finite set, $\{1, \dots, K\}$. The transition probability from state i to state j is expressed as

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

The $K \times K$ matrix $\mathbf{A} = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), π , of the first state $\mathbf{S}_1$:

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

Because the lagged dependent variables are included in the regressors and the autoregression parameters depend on the state, the $\mathbf{Y}_t$ follows different autoregressions in different regimes; hence, this type of model is called the regime-switching autoregression model.

Consider a univariate regime-switching autoregression model that has two regimes:

$$y_t = \begin{cases} 0.8 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } \mathbf{S}_t = 1 \\ -0.7 * y_{t-1} + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } \mathbf{S}_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

```
%let pi1 = 0.5;
%let a11 = 0.95;
%let a22 = 0.95;
%let ar1_1_1_1 = 0.8;
%let cov1_11 = 2.56;
%let ar2_1_1_1 = -0.7;
```

```

%let cov2_11 = 4;
%let T = 1000;
%let seed = 1234;

data rsardgp;
  retain cd1_11 cd2_11;
  ylag = 0;
  do t = 1 to &T.;
    if(t=1) then do;
      /* initial probability distribution */
      p = &p11.;
      /* Cholesky decomposition of COV1 */
      cd1_11 = sqrt(&cov1_11.);
      /* Cholesky decomposition of COV2 */
      cd2_11 = sqrt(&cov2_11.);
    end;
    else do;
      /* transition probability matrix */
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e = normal(&seed.);
    if(s=1) then do;
      /* y ~ N(beta1*ylag, Sigma1) at state 1 */
      y = &ar1_1_1_1 * ylag + cd1_11*e;
    end;
    else do;
      /* y ~ N(beta2*ylag, Sigma2) at state 2 */
      y = &ar2_1_1_1 * ylag + cd2_11*e;
    end;
    output;
    lags = s;
    ylag = y;
  end;
run;

data rsar;
  set rsardgp;
  keep t y;
run;

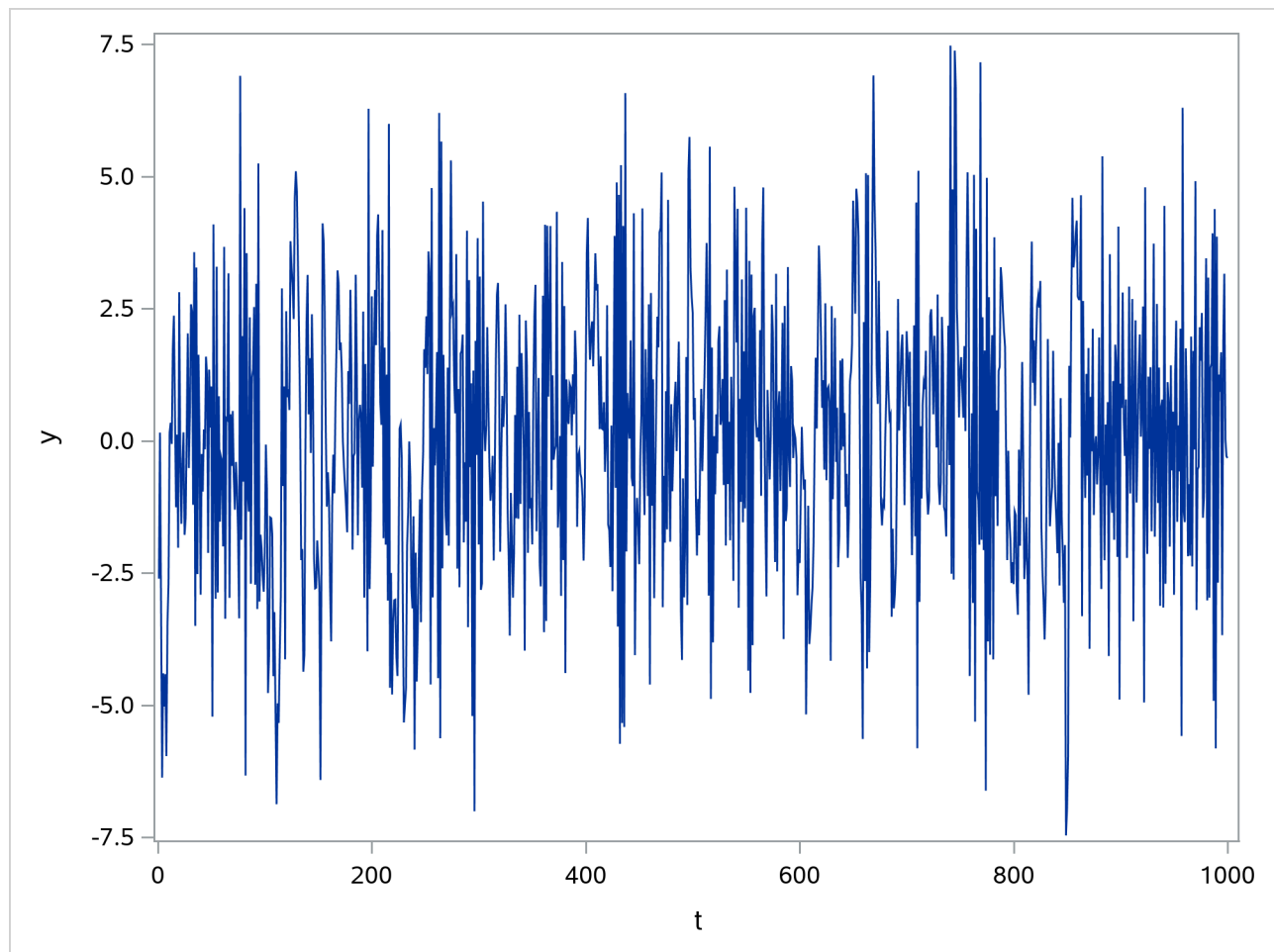
```

The following statements create the series plot in [Figure 13.47](#). It is difficult to tell directly from the plot that the series are from two different regimes.

```

proc sgplot data=rsar;
  series y=y x=t;
run;

```

Figure 13.47 Series Plot of Data Points

The following code uploads the data table and estimates the regime-switching autoregression model:

```
data mycas.rsar; set rsar; run;
proc hmm data=mycas.rsar;
  id time=t;
  model y / type=ar noint ylag=1 nstate=2 method=ml;
  optimize algorithm=activeset printLevel=3 printIterFreq=1;
  learn out=mycas.mylearn(promote=no replace=yes);
  filter out=mycas.myfilter(promote=no replace=yes);
  smooth out=mycas.mysmooth(promote=no replace=yes);
  decode out=mycas.mydecode(promote=no replace=yes);
  evaluate out=mycas.myeval(promote=no replace=yes);
run;
```

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

The estimates of all parameters are shown in [Figure 13.48](#), which displays columns for parameter name, estimate value, standard error, t value, and p -value. The parameter estimates are very close to the true parameter values that are used in the data generating process.

Figure 13.48 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.937990	0.013985	67.07	<.0001
TPM1_2	0.062010	0.013985	4.43	<.0001
TPM2_1	0.056595	0.013425	4.22	<.0001
TPM2_2	0.943405	0.013425	70.27	<.0001
AR1_1_1_1	0.759087	0.032724	23.20	<.0001
AR2_1_1_1	-0.630710	0.037261	-16.93	<.0001
COV1_1_1	2.687805	0.198060	13.57	<.0001
COV2_1_1	4.225692	0.280319	15.07	<.0001

The accuracy of classification through the regime-switching autoregression model is calculated by the following statements:

```

data rsarCheck;
  merge rsardgp(in=a) mycas.mydecode(in=b);
  by t;
  if(s=state) then do; correct=1; ds = s; end;
  else do; correct=0; ds = -s; end;
  if(a=b);
  keep t y s correct ds;
run;

%let qFlipState = -1;
data rsarAccuracy;
  set rsarCheck;
  retain count 0 correctCount 0;
  count = count + 1;
  correctCount = correctCount + correct;
  if(count>&T.-0.5) then do;
    accuracy = correctCount / count;
    if(accuracy<0.5) then call symputx("qFlipState",1,'G');
    else call symputx("qFlipState",0,'G');
    if(accuracy<0.5) then accuracy = 1 - accuracy;
    output;
  end;
  keep accuracy;
run;

data rsarCheck;
  set rsarCheck;
  if(&qFlipState.=1) then do;
    ds = -ds;
    correct = 1 - correct;
  end;
run;

proc print data = rsarAccuracy noobs; run;

```

The accuracy of the regime-switching autoregression model, which is shown in [Figure 13.49](#), is more than 92%. This means that the regime-switching autoregression model successfully distinguishes the two regimes.

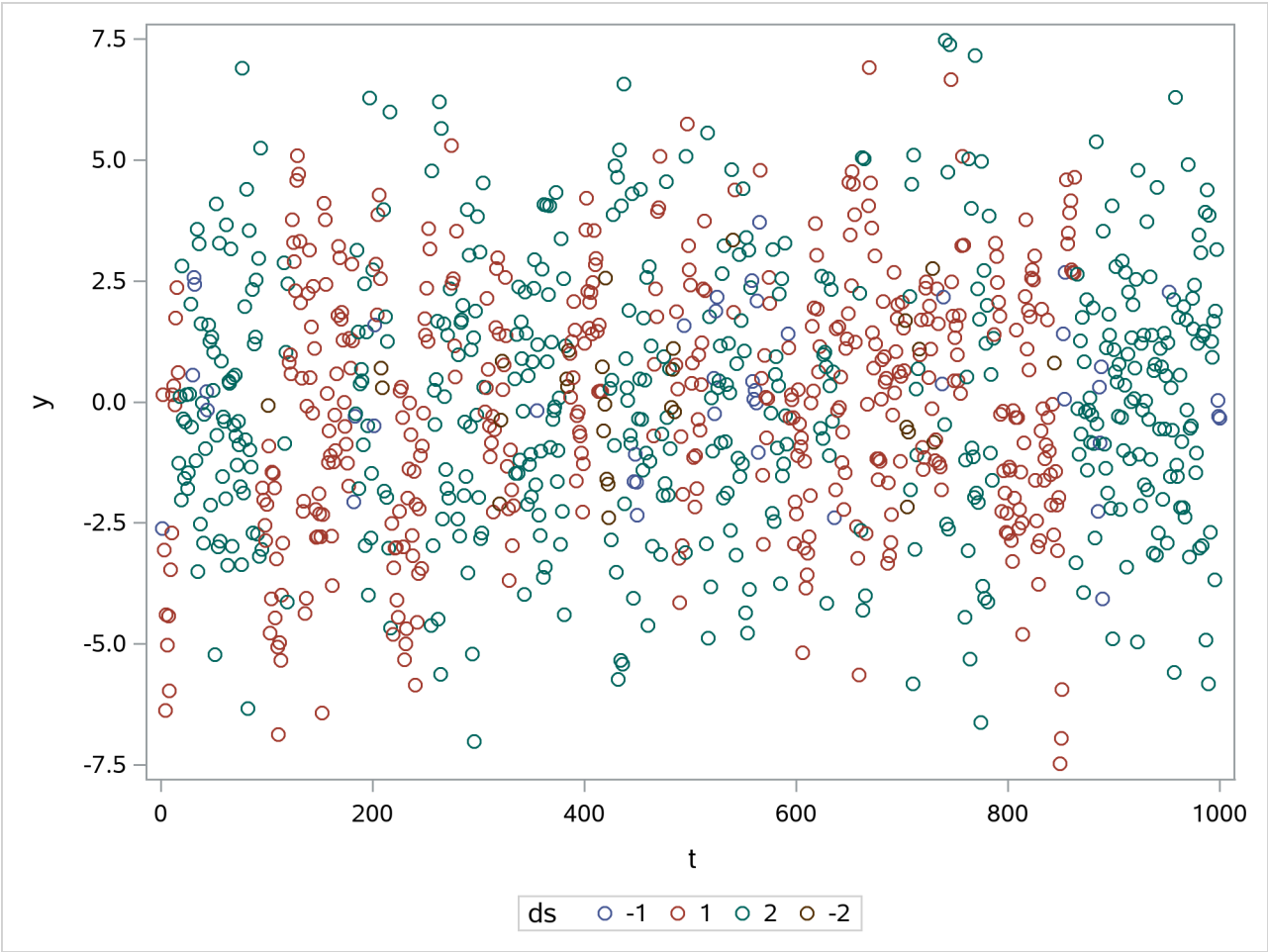
Figure 13.49 Accuracy of Decoding by the Regime-Switching Autoregression Model

accuracy
0.927

The following statements create the scatter plot in [Figure 13.50](#) to show how well the data points are classified:

```
proc sgplot data=rsarCheck;  
  scatter y=y x=t / group=ds;  
run;
```

Figure 13.50 Scatter Plot of Data Points



Syntax: HMM Procedure

```

PROC HMM DATA=CAS-libref.data-table < options > ;
  DECODE < option > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  ESTIMATE < options > ;
  EVALUATE < option > ;
  FILTER < option > ;
  FORECAST < options > ;
  ID TIME= variable < SECTION= variable > ;
  INITIAL equation < , ... , equation > ;
  MODEL dependents < = regressors > < / options > ;
  OPTIMIZE < options > ;
  PRIOR distribution < , ... , distribution > ;
  SCORE < options > ;
  SMOOTH < option > ;
  STORE < OUT= > CAS-libref.data-table ;

```

Functional Summary

The statements and options available in the HMM procedure are summarized in [Table 13.1](#).

Table 13.1 Functional Summary

Description	Statement	Option
Input Data Table Options		
Specifies the input data table	PROC HMM	DATA=
Specifies the input data table for parameter estimation	ESTIMATE	IN=
Specifies the input data table for model information	SCORE	INMODEL=
Output Data Table Options		
Writes the log likelihood and information criteria to the specified output data table	PROC HMM	OUTSTAT=
Writes the parameter estimates to the specified output data table	ESTIMATE	OUT=
Writes the parameter estimates and their standard errors and covariance matrix to the specified output data table	ESTIMATE	OUTALL=
Writes the evaluation results to the specified output data table	EVALUATE	OUT=
Writes the decoding results to the specified output data table	DECODE	OUT=
Writes the filtering results to the specified output data table	FILTER	OUT=

Table 13.1 *continued*

Description	Statement	Option
Writes the forecasting results to the specified output data table	FORECAST	OUT=
Writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table	FORECAST	OUTALL=
Writes the model information to the specified output data table	SCORE	OUTMODEL=
Writes the smoothing results to the specified output data table	SMOOTH	OUT=
Writes the model information to the specified output data table that can be processed by the ASTORE procedure	STORE	OUT=
ID Variable		
Specifies the variable that identifies the time or sequence	ID	TIME=
Specifies the variable that identifies the section	ID	SECTION=
Model Options		
Specifies the mean form in a regime-switching autoregression model	MODEL	ARMEAN=
Estimates the initial state probability vector (ISPV)	MODEL	ESTISPV
Specifies the estimation method	MODEL	METHOD=
Specifies the number of components in a state in a Gaussian mixture HMM (GM HMM)	MODEL	NCOMPONENT
Suppresses the present values of the exogenous variables	MODEL	NOCURRENTX
Suppresses the constant (intercept) parameters	MODEL	NOINT
Specifies the number of seasonal periods	MODEL	NSEASON=
Specifies the number or the range of numbers of hidden states	MODEL	NSTATE=
Centers seasonal dummies	MODEL	SCENTER
Specifies the state-independent parameters	MODEL	STATEINDEPENDENT=
Specifies the degree of deterministic time trend	MODEL	TREND=
Specifies the type of model of interest	MODEL	TYPE=
Specifies the lags of exogenous (independent) variables	MODEL	XLAG=
Specifies the order or the range of orders of the autoregressive process	MODEL	YLAG=
Optimization Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=

Table 13.1 *continued*

Description	Statement	Option
Specifies the initial parameter values for optimization	INITIAL	
Specifies the priors for parameters	PRIOR	
Specifies the optimization algorithm	OPTIMIZE	ALGORITHM=
Specifies the maximum number of iterations in the optimization process	OPTIMIZE	MAXITER=
Specifies the maximum time (in seconds) allowed for optimization	OPTIMIZE	MAXTIME=
Specifies the print iteration frequency	OPTIMIZE	PRINTITERFREQ=
Specifies the print level	OPTIMIZE	PRINTLEVEL=
Optimization Suboptions for the Active Set or Interior Point Algorithm		
Specifies the range from which each variable can take values during the sampling process in multistart mode	OPTIMIZE	MSRANGE=
Specifies whether to enable multistart mode	OPTIMIZE	MULTISTART=
Specifies the upper limit on the magnitude of the objective value	OPTIMIZE	OBJLIMIT=
Specifies the tolerance for infeasibility	OPTIMIZE	FEASTOL=
Specifies the tolerance for the optimality error	OPTIMIZE	OPTTOL=
Specifies whether the optimizer should return only a solution that is locally optimal	OPTIMIZE	SOLTYPE=
Optimization Suboptions for the Stochastic Gradient Descent (SGD) Algorithm		
Specifies the annealing rate	OPTIMIZE	ANNEALINGRATE=
Specifies the learning rate	OPTIMIZE	LEARNINGRATE=
Specifies the size of the minibatches to use	OPTIMIZE	MINIBATCHSIZE=
Specifies the value for momentum	OPTIMIZE	MOMENTUM=
Suppresses the adaptive gradient descent algorithm	OPTIMIZE	NONADUPDATE
Specifies the value of adaptive rate	OPTIMIZE	ADAPTIVEDECAY=
Requests that computational threads share a common weight vector and that they update the weight vector without race conditions	OPTIMIZE	USELOCKING=
Specifies the length of a chain in a minibatch	OPTIMIZE	CHAIN=(LENGTH=)
Specifies the buffer length on the left side of a chain in a minibatch	OPTIMIZE	CHAIN=(BUFFER=)
Forecasting Options		
Specifies a nonnegative integer to use as the seed for generating random number sequences	PROC HMM	SEED=
Specifies the size of confidence limits for forecasting	FORECAST	ALPHA=
Starts forecasting before the end of the input data	FORECAST	BACK=
Specifies how many periods to forecast	FORECAST	LEAD=

Table 13.1 *continued*

Description	Statement	Option
Specifies the number of paths to simulate in the forecasting of a regime-switching autoregression model	FORECAST	NSIM=
Specifies whether to perform forecasts after each observation	FORECAST	ONLINE
Output Control Options		
Specifies the order in which to label the states (and components)	PROC HMM	LABELSWITCH=
Specifies the ODS tables to display	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	

PROC HMM Statement

PROC HMM DATA=*CAS-libref.data-table* < options > ;

The PROC HMM statement invokes the HMM procedure. You must specify the following *option*:

DATA=*CAS-libref.data-table*

names the input data table for PROC HMM to use. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>CAS-libref</i> , see the section “Using CAS Sessions and CAS Engine Librefs” on page 599.
<i>data-table</i>	specifies the name of the input data table.

You can also specify the following *options*:

LABELSWITCH=(*options*)

specifies the order in which to label the states (and components). The label switching problem is a common issue for HMMs because any two HMMs are mathematically equivalent if the only difference between two models is that the states are labeled in different ways. For example, consider the following two univariate bivariate Gaussian HMMs, which are described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels “1” and “2” are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. The LABELSWITCH= option helps you to define the order of the state (and component) labels. You can specify the following *options*:

OUT=CAS-libref.data-table

writes the changes of state labels (and component labels in the case of GM HMMs) to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

REORDER=(number-list)

specifies the new order of the state labels, where the *number-list* directly specifies the new order of the states. The *number-list* must be a permutation of numbers 1 to K , where K is the number of states; that is, the size of the *number-list* must be K , and each element in the *number-list* must be unique and between 1 and K . For example, REORDER=(3 1 2) implies that the original state 1 become state 3, state 2 become state 1, and state 3 become state 2. If you specify this option, the SORT= option is ignored.

REORDERCOMPONENT=(number-list)

specifies the new order of the component labels in the case of GM HMMs, where the *number-list* directly specifies the new order of the components for each state. For a K -state M -component GM HMM, the size of the *number-list*, $\{n_{11}, \dots, n_{1M}, \dots, n_{K1}, \dots, n_{KM}\}$, must be KM , and each sublist $\{n_{i1}, \dots, n_{iM}\}$, $i = 1, \dots, K$, must be a permutation of numbers 1 to M —that is, each element in the sublist $\{n_{i1}, \dots, n_{iM}\}$ must be unique and between 1 and M . For example, REORDERCOMPONENT=(3 2 1 2 1 3) for a bivariate tricomponent GM HMM implies that the original components 1, 2, and 3 for the original state 1 become components 3, 2, and 1, respectively, for the original state 1, and the original components 1, 2, and 3 for the original state 2 become components 2, 1, and 3, respectively, for the original state 2. If you specify this option, the labels for components in each state are not sorted.

SORT=type <(name-list)>

specifies whether to sort the labels of states (and components) and in which type of order. You can specify the following *types*:

ASC <(name-list)> sorts the labels in ascending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meaning of those names are defined in Table 13.4. Any parameters that are not in the model are ignored.

DESC *<(name-list)>* sorts the labels in descending order. You can add a *name-list* in parentheses in order to sort the labels of states (and components). The possible names that can appear in the *name-list* are ISPV, TPM, MCP, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV; the meaning of those names are defined in [Table 13.4](#). Any parameters that are not in the model are ignored.

NONE does not sort the labels.

By default, if the INMODEL= option in the SCORE statement is not specified, SORT=ASC(SIGMA); otherwise, SORT=NONE. For example, when the ESTISPV option is not specified, the ISPV is not in the stationary HMM; if you include ISPV in the *name-list*, it is ignored. For a GM HMM, the labels of components for each state are first sorted by the parameters state-by-state if the REORDERCOMPONENT= option is not specified; then the labels of states are sorted by the parameters that are stacked in the new order of components.

For more details and examples, see the section “[Label Switching Problem](#)” on page 694.

OUTSTAT=CAS-libref.data-table

writes the log likelihood and information criteria to the specified output data table. The information criteria include Akaike’s information criterion (AIC), the corrected Akaike’s information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 599.

SEED=number

specifies a nonnegative integer to be used as the seed for generating random number sequences. You can use this option to replicate results from different runs. By default, SEED=1.

DECODE Statement

DECODE *<option>* ;

The DECODE statement solves the decoding problem by finding the best possible state path for the observations. You can specify the following *option*:

OUT=CAS-libref.data-table

writes the decoding results to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 599.

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[ODS Table Names](#)” on page 715. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[ODS Table Names](#)” on page 715. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

ESTIMATE Statement

ESTIMATE < options > ;

LEARN < options > ;

The ESTIMATE statement estimates the parameters. You can specify the following *options*:

IN=*CAS-libref.data-table*

specifies the input data table for parameter estimation. In general, the data table is an output data table that is produced by the OUT= option in the ESTIMATE statement in a previous call of the HMM procedure. To estimate each model that is specified in the MODEL statement, the HMM procedure first checks whether an INITIAL statement is specified for the model. If the INITIAL statement exists for the model, the initial values that are specified in the INITIAL statement are used; otherwise, the HMM procedure searches the IN= data table to see whether it contains a model that has the same model specification. If the IN= data table contains a matched model, its values are used as the initial values for the model that is specified in the MODEL statement.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

OUT=*CAS-libref.data-table*

writes the parameter estimates to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

OUTALL=*CAS-libref.data-table*

writes the parameter estimates and their standard errors and covariance matrix to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

EVALUATE Statement

EVALUATE < option > ;

The EVALUATE statement evaluates the model by calculating the log likelihood. You can specify the following *option*:

OUT=*CAS-libref.data-table*

writes the evaluation results to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

FILTER Statement

FILTER < option > ;

The FILTER statement solves the filtering problem, which is to find the probability distribution of the current state, conditional on the observations that precede and include the current observation. You can specify the following *option*:

OUT=CAS-libref.data-table

writes the filtering results to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

FORECAST Statement

FORECAST < options > ;

The FORECAST statement finds the probability distribution of the future states and the dependent variables. You can specify the following *options*:

ALPHA=number

specifies the forecast confidence limit size, where *number* is between 0 and 1. When you specify this option, the upper and lower confidence limits define the $100\alpha\%$ confidence interval. By default, ALPHA=0.95, which produces 95% confidence intervals.

BACK=number

specifies the number of observations before the end of the data at which the multistep forecasts begin, where *number* must be less than or equal to the number of observations minus the number of lagged regressors in the model. By default, BACK=0, which means that the forecasts start at the end of the available data.

LEAD=number

specifies the number of multistep forecast values to compute, where *number* must be a nonnegative integer. By default, LEAD=1, which means that a one-step-ahead forecast is performed.

NSIM=number

specifies the number of paths to be simulated in the forecasting of a regime-switching autoregression model, where *number* must be a positive integer. By default, NSIM=100.

ONLINE

requests that the (multistep) forecasts be done after each observation. When you specify this option, the BACK= option is ignored.

OUT=CAS-libref.data-table

writes the forecasting results to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

OUTALL=CAS-libref.data-table

writes the state distributions, means, standard errors, confidence intervals, medians, and covariance matrices of forecasts for dependent variables to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

ID Statement

ID TIME=*variable* < **SECTION=***variable* > ;

The ID statement identifies observations in the input data table by specifying a variable for the time series data or two variables for the cross-sectional time series data.

You must specify the following option:

TIME=*variable*

specifies the temporal or sequential order of the observations. The *variable* cannot have missing values.

You can also specify the following option:

SECTION=*variable*

identifies the section of each observation for modeling cross-sectional time series data. The *variable* cannot have missing values.

INITIAL Statement

INITIAL *equation* < , . . . , *equation* > ;

The INITIAL statement specifies the initial parameter values for nonlinear optimization when the maximum likelihood method or the maximum a posteriori method is applied to the estimation of the HMMs (that is, METHOD=ML or METHOD=MAP, respectively, in the MODEL statement). Only one INITIAL statement is allowed. If you specify more than one *equation*, separate them with commas. The INITIAL statement supports only equations; hence, inequality comparison operators (<, <=, >, and >=) and the distribution operator (~) are not supported in the INITIAL statement.

If you specify the NSTATE= n_1 : n_2 option or the YLAG= m_1 : m_2 option (or both) in the MODEL statement to estimate multiple models, the INITIAL statement is applied to the first model that corresponds to the specification of NSTATE= n_1 or YLAG= m_1 (or that corresponds to both NSTATE= n_1 and YLAG= m_1 if both options are specified).

To use the INITIAL statement, you need to know the form of the model: different sets of parameters are available for different types of HMMs. Nonlinear equations on parameters are not supported.

The *equation* is in the form of a matrix expression. For more information about the matrix expression, see the section “[Matrix Expression](#)” on page 683.

The initial parameter values are values that solve the specified linear equations. If you do not specify initial values for all parameters, the default initial value for any parameter that is not specified in the INITIAL statement is 0, except for the following:

- The diagonal elements of the **SIGMA** or **COV** parameter matrix are set to ones if the **SIGMA** or **COV** parameter matrix is to be estimated.
- If a row of the **TPM** parameter matrix contains all zeros, each element in the row is set to $1/K$, where K is the number of states; if a row of the **TPM** parameter matrix does not add up to one, the row is normalized by dividing each element in the row by the sum of the row.
- If all elements of the **ISPV** parameter vector are zeros, each element of the **ISPV** parameter vector is set to $1/K$, where K is the number of states; if the sum of the **ISPV** parameter vector is not equal to one, it is normalized by dividing each element of the **ISPV** parameter vector by the sum of the **ISPV** parameter vector.

The following example uses the INITIAL statement for a bivariate three-state Gaussian HMM, which PROC HMM estimates by the maximum likelihood method by default:

```
proc hmm data=One;
  id time = t;
  model y1 y2 / type=gaussian nstate=3;
  initial TPM={0.8 0.1 0.1, 0.2 0.7 0.1, 0.9 0.05 0.05},
           MU={-1 -1, 0 0, 1 1}` ,
           SIGMA(1)=4*I(2),
           SIGMA(2)=I(2),
           SIGMA(3)=4*I(2)+2;
run;
```

MODEL Statement

MODEL *dependents* < = *regressors* > < / *options* > ;

The MODEL statement specifies dependent (endogenous) variables and regressors (the independent or exogenous variables) for the HMM model. Only one MODEL statement is allowed.

You can specify the following *options* after a forward slash (/):

ARMEAN=ADJUSTED | STANDARD

specifies the mean form in a regime-switching autoregression model. The ARMEAN= option applies only when you specify the TYPE=AR option in the MODEL statement. You can specify the following values:

ADJUSTED	specifies the mean-adjusted form of regime-switching autoregression model.
STANDARD	specifies the standard regime-switching autoregression model.

By default, METHOD=STANDARD.

ESTISPV

estimates the initial state probability vector (ISPV). By default, the ISPV is fixed to the stationary distribution of the Markov chain. For the finite state space and homogeneous HMM, the stationary distribution of the Markov chain is the π such that $\pi' A = \pi'$, where A is the transition probability matrix (TPM); that is, π is the leading left eigenvector of A . If you do not specify this option but you do specify the ISPV function in the INITIAL or PRIOR statement, the HMM procedure issues errors and then stops.

METHOD=MAP | ML

requests the type of estimates to be computed. You can specify the following values:

MAP	specifies the maximum a posteriori method.
ML	specifies the maximum likelihood method.

By default, METHOD=ML.

NCOMPONENT=number

specifies the number of components in a state for a Gaussian mixture HMM (GM HMM), where *number* must be a positive integer. This option applies only when you specify TYPE=GAUSSIANMIXTURE. By default, NCOMPONENT=1.

NOCURRENTX

suppresses the present values of the exogenous variables. This option applies only when you specify TYPE=AR or TYPE=REG.

NOINT

suppresses the constant (intercept) parameters. This option applies only when you specify TYPE=AR or TYPE=REG.

NSEASON=s

specifies the number of seasonal periods (where *s* must be a positive integer) and adds (*s*−1) seasonal dummies to the regressors. This option applies only when you specify TYPE=AR or TYPE=REG and do not specify the NOINT option.

NSTATE=number1 < : number2 >

specifies the number or the range of numbers of hidden states, where *number1* and *number2* must be positive integers. If you specify NSTATE=*number1*, the number of hidden states is *number1*; if you specify NSTATE=*number1:number2*, multiple models whose number of hidden states is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. By default, NSTATE=2.

SCENTER

centers seasonal dummies that are implied by the NSEASON= option. The centered seasonal dummies are generated by $c - \frac{1}{s}$, where *c* is a seasonal dummy that the NSEASON=*s* option generates. This option applies only when you specify the NSEASON= option and TYPE=AR or TYPE=REG.

STATEINDEPENDENT=(name-list)

specifies the parameters to be estimated independent of state. The names of the independent parameters are entered in the *name-list* in parentheses. The names that can appear in the *name-list* are ISPV, TPM, MCP, MU, SIGMA, CONST, SDUMMY, LTREND, QTREND, XL, AR, and COV. Table 13.4 gives the meanings of these names. This option does not support inputting the component index after the parameter names. For example, it supports COV but not COV(1,1,1). The name ISPV is ignored unless ESTISPV is also specified. Any parameters that are not in the model are ignored. Currently this option does not support ALGORITHM=SGD in the OPTIMIZE statement. The estimates are ordered by the names specified in the *name-list*.

To see how this option works, consider the following univariate bivariate Gaussian HMM, which is described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, μ_1 and Σ_1 are the mean and covariance parameters for state 1, and μ_2 and Σ_2 are the mean and covariance parameters for state 2. STATEINDEPENDENT=(SIGMA) estimates the model with $\Sigma_1 = \Sigma_2$. The following statement specifies this option:

```
model y / stateindependent=(sigma);
```

For more details and an example, see “Example 13.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model” on page 768.

TREND=LINEAR | QUAD

specifies the degree of deterministic time trend that the model includes. You can specify the following values:

LINEAR	includes a linear time trend as a regressor.
QUAD	includes linear and quadratic time trends as regressors.

This option applies only when you specify TYPE=AR or TYPE=REG.

TYPE=AR | GAUSSIAN | GAUSSIANMIXTURE | REG

specifies the type of the model of interest to use. You can specify the following values:

AR	uses the regime-switching autoregression model (RS-AR, also known as the autoregressive hidden Markov model, AR HMM).
GAUSSIAN	uses the Gaussian hidden Markov model (Gaussian HMM).
GAUSSIANMIXTURE	uses the Gaussian mixture hidden Markov model (GM HMM).
REG	uses the regime-switching regression model (RS-REG, also known as the regression hidden Markov model, REG HMM).

By default, TYPE=GAUSSIAN.

XLAG=number

specifies the lags of exogenous (independent) variables. To exclude the present values of exogenous variables from the model, you must specify the NOCURRENTX option. This option applies only when you specify TYPE=AR or TYPE=REG. By default, XLAG=0.

YLAG=number1 < : number2 >

specifies the order or the range of orders of the autoregressive process, where *number1* and *number2* must be nonnegative integers. If you specify YLAG=*number1*, the order of the autoregressive process is *number1*; if you specify YLAG=*number1:number2*, multiple models whose order of the autoregressive process is inclusively between *number1* and *number2* are estimated and inferred. If you specify the same number for both *number1* and *number2*, *number2* is ignored. This option applies only when you specify TYPE=AR. By default, YLAG=0.

OPTIMIZE Statement

OPTIMIZE < options > ;

The OPTIMIZE statement optimizes the parameter estimation.

You can specify the following *options*:

ALGORITHM=type < (suboptions) >

specifies the optimization algorithm to use. You can specify the following *types*:

ACTIVESET < (suboptions) >

uses the active set algorithm. You can specify one or more of the following *suboptions* in parentheses:

FEASTOL=number

defines the feasible tolerance. The optimizer exits if the constraint violation is less than *number* and the scaled optimality conditions are less than the value of the OPTTOL= suboption. By default, FEASTOL= 10^{-6} .

MSRANGE=number

defines the range from which each variable can take values during the sampling process in multistart mode, where *number* must be a positive number. This option is ignored if MULTISTART=0. This option affects only the sampling process that determines starting points for the local solver. It does not affect the bounds of the original nonlinear optimization problem. By default, MSRANGE=2.

MULTISTART=0 | 1

specifies whether to enable multistart mode. You can specify the following values:

0 disables multistart mode.

1 solves the problem from multiple starting points, possibly finding a better local optimum as a result. The computing cost in multistart mode might be huge.

By default, MULTISTART=0.

OBJLIMIT=number

specifies an upper limit on the magnitude of the objective value, where *number* must be greater than or equal to 10^8 . If **OBJLIMIT=M**, the algorithm terminates when the objective value becomes less than $-M$ for a minimization problem or when the objective value exceeds M for a maximization problem. The termination of the algorithm implies that either the problem is unbounded or the algorithm diverges. If optimization were allowed to continue, numerical difficulty might be encountered. By default, **OBJLIMIT=10²⁰**. If *number* is less than 10^8 , it is reset to the default value of 10^{20} .

OPTTOL=ε

specifies the measure by which the current iterate is determined to be an acceptable approximation of a local optimum, where ϵ is a positive real number. The optimizer determines that the current iterate is a local optimum when the norm of the scaled vector of the optimality conditions is less than ϵ and the true constraint violation is less than the value of the **FEASTOL=** option. By default, **OPTTOL=10⁻⁶**.

SOLTYPE=0 | 1

specifies the type of solution that the optimizer should return. You can specify the following values:

- 0** requests that the optimizer return a locally optimal solution, provided that it locates one.
- 1** requests that the optimizer return the best feasible solution that was found, provided that its objective value is better than that of the locally optimal solution that was found.

By default, **SOLTYPE=1**.

INTERIORPOINT <(suboptions)>

uses the interior point algorithm. You can specify the same *suboptions* in parentheses as you can specify for **ALGORITHM=ACTIVESET**.

SGD <(suboptions)>

uses the stochastic gradient descent algorithm. You can specify one or more of the following *suboptions* in parentheses:

NONADUPDATE

suppresses the adaptive gradient descent algorithm.

ADAPTIVEDECAY=number

specifies the adaptive rate for the gradient descent algorithm when it is used, where the algorithm becomes **ADAGRAD** if **ADAPTIVEDECAY=1** and becomes **ADADELTA** otherwise. By default, **ADAPTIVEDECAY=0.95**.

ANNEALINGRATE=number

specifies the annealing parameter, β , where *number* must be a nonnegative double. Annealing is a way to automatically reduce the learning rate as SGD progresses, causing smaller steps as SGD approaches a solution. Effectively, this option replaces the learning rate parameter, η , with

$$\eta' = \frac{\eta}{1 + \beta t}$$

where t is the number of iterations that SGD has performed. By default, ANNEALINGRATE= 10^{-6} .

CHAIN=(options)

specifies the options for the sequence data (which constitute a chain). You can specify the following *options* within parentheses:

LENGTH=number

specifies the length of a chain in a minibatch in optimization process, where *number* must be a positive integer. By default, LENGTH=1000.

BUFFER=number

specifies the buffer length on the left side of a chain in a minibatch in optimization process, where *number* must be a nonnegative integer. By default, BUFFER=20.

LEARNINGRATE=number

specifies the learning rate parameter, η , for SGD, where *number* must be a positive double. New iterates for SGD are found by using

$$\theta_{k+1} = \theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i)$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k , and $L(\theta_k; x_i, y_i)$ is the negative log likelihood that is associated with the i th chain of observation. If you see a huge objective value with SGD (especially with a small data table), it is likely that too large a value was specified for this option. By default, LEARNINGRATE=0.001.

MINIBATCHSIZE=number

specifies the size of the minibatches used in SGD, where *number* must be a positive integer. By default, MINIBATCHSIZE=1.

MOMENTUM=number

specifies the value for momentum parameter α , where *number* must be greater than or equal to 0 and less than or equal to 1. New iterates for SGD that uses the momentum method are found by using

$$\begin{aligned} \theta_{k+1} &= \theta_k + \nabla \theta_k \\ \nabla \theta_k &= \alpha \nabla \theta_k - \frac{\eta}{\|I_k\|} \sum_{(x_i, y_i) \in I_k} \nabla L(\theta_k; x_i, y_i) \end{aligned}$$

where θ_k is the current weight vector, θ_{k+1} is the new weight vector, I_k is the minibatch that is used during iteration k , and $L(\theta_k; x_i, y_i)$ is the negative log-likelihood that is associated with the i th chain of observation. By default, MOMENTUM=0.

USELOCKING

requests that computational threads share a common weight vector and that they update the weight vector without race conditions. By default, computational threads update a single weight vector simultaneously. This causes intentional race conditions and nondeterministic behavior, but increases performance significantly.

By default, ALGORITHM=ACTIVESET.

MAXITER=number

specifies the maximum number of iterations in the optimization process, where *number* must be a nonnegative integer. By default, MAXITER=128.

MAXTIME=t

specifies an upper limit of *t* seconds for the optimization process, including problem generation time and solution time, where *t* can be any positive number. The default value of *t* is the positive number that has the largest absolute value that your operating environment can represent. If you do not specify this option, the optimizer does not stop on the basis of the amount of time elapsed.

PRINTITERFREQ=number

specifies how frequently to output the optimization process history, where *number* must be a nonnegative integer. If PRINTITERFREQ=0, no iterations are output. If *number*=*n* and *n* is positive, the optimization process history is output every *n* iterations. By default, PRINTITERFREQ=0.

PRINTLEVEL=0 | 1 | 2 | 3

specifies the print level. You can specify the following values:

- 0** suppresses output of the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.
- 1** outputs the initial parameter values and the initial objective value.
- 2** outputs the initial parameter values, the initial objective value, and the algorithm information.
- 3** outputs the initial parameter values, the initial objective value, the algorithm information, the final parameter values, and the final objective value.

By default, PRINTLEVEL=0.

PRIOR Statement

PRIOR *distribution* < , . . . , *distribution* > ;

The PRIOR statement specifies the initial hyperparameter values in the prior distributions of the parameters when the maximum a posteriori method is applied to the estimation of the HMM (that is, when METHOD=MAP in the MODEL statement). If you specify the maximum likelihood (ML) method (METHOD=ML), the PRIOR statement is ignored. You can specify only one PRIOR statement. If you specify more than one *distribution*, separate them with commas.

If you specify the NSTATE=*n*₁ : *n*₂ option or the YLAG=*m*₁ : *m*₂ option (or both) in the MODEL statement to estimate multiple models, the PRIOR statement is applied to the first model that corresponds to the

specification of $\text{NSTATE}=n_1$ or $\text{YLAG}=m_1$ (or that corresponds to both $\text{NSTATE}=n_1$ and $\text{YLAG}=m_1$ if both options are specified).

To use the PRIOR statement, you need to know the type of the model: different sets of prior distributions are available for different types of HMMs.

The *distribution* is in the following format:

parameter-name $\sim$ *distribution-function*(arguments) ;

You can specify the following *parameter-names*:

ISPV	indicates the initial state probability vector. The prior distribution must be the Dirichlet distribution.
MCP	indicates the mixture component probabilities in the Gaussian mixture HMM (GM HMM). The prior distribution must be the Dirichlet distribution.
MUSIGMA	indicates the mean and covariance parameters for the observation distributions. For the Gaussian HMM, regime-switching regression model, or regime-switching autoregression model, the prior distribution of the mean and covariance parameters for each state must be the normal-inverse-Wishart (NIW) distribution. For the GM HMM, the prior distribution of the mean and covariance parameters for each component at each state must be the normal-inverse-Wishart (NIW) distribution.
TPM	indicates the transition probability matrix. The prior distribution must be the Dirichlet distribution.

You can specify the following *distribution-functions*:

DIR	specifies the Dirichlet distribution. There is one <i>argument</i> for the Dirichlet distribution, and it must be a valid matrix expression of the proper size. For example, if the Dirichlet distribution is the prior distribution of the ISPV, MCP, CPM or TPM, the size of the <i>argument</i> must be the same as the size of the ISPV, MCP, CPM or TPM. All elements of the <i>argument</i> should be positive. If you specify any nonpositive element in the <i>argument</i> , the nonpositive element is changed to the machine precision. For information about how to define the matrix expressions, see the section “ Matrix Expression ” on page 683. For more information about the Dirichlet distribution, see the section “ Maximum a Posteriori (MAP) Method ” on page 678.
NIW	specifies the normal-inverse-Wishart (NIW) distribution. There are four <i>arguments</i> for the NIW distribution ($\tilde{\mu}$, $\tilde{\Sigma}$, $\tilde{\kappa}$, and $\tilde{\nu}$), and they must be valid matrix expressions of the proper size. If the NIW distribution is the prior distribution of the MUSIGMA for Gaussian HMM, then the following statements are true: • $\tilde{\mu}$ is a $K \times k_y$ matrix, where K is the number of states, and k_y is the number of dependent variables. • $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix. • $\tilde{\kappa}$ is a $K \times 1$ vector. • $\tilde{\nu}$ is a $K \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for GM HMM, then the following statements are true:

- $\tilde{\mu}$ is a $KM \times k_y$ matrix, where K is the number of states, M is the number of components at a state, and k_y is the number of dependent variables.
- $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix.
- $\tilde{\kappa}$ is a $KM \times 1$ vector.
- $\tilde{\nu}$ is a $KM \times 1$ vector.

If the NIW distribution is the prior distribution of the MUSIGMA for regime-switching regression or regime-switching autoregression, then the following statements are true:

- $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, where K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors in each equation. (For more information about how to calculate k_z , see the sections “[Regime-Switching Regression Model](#)” on page 673 and “[Regime-Switching Autoregression Model](#)” on page 674.)
- $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix.
- $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix.
- $\tilde{\nu}$ is a $K \times 1$ vector.

For information about how to define the matrix expressions, see the section “[Matrix Expression](#)” on page 683. For more information about the normal-inverse-Wishart (NIW) distribution, see the section “[Maximum a Posteriori \(MAP\) Method](#)” on page 678.

The following statements specify the PRIOR statement for the ISPV, TPM, and MUSIGMA in a bivariate three-state Gaussian HMM:

```
proc hmm data=mycas.One;
  id time=t;
  model x y / type=gaussian nstate=3 estispv method=map;
  prior ISPV ~ dir(J(3,1,1)),
        TPM ~ dir(I(3)*100+J(3,3,1)),
        MUSIGMA ~ niw(J(1,2,-1) // J(1,2,0) // J(1,2,1),
                      J(3,1,1)@ (I(2)*10),
                      5*J(3,1,1),
                      10*J(3,1,1));
run;
```

SCORE Statement

SCORE *options* ;

The SCORE statement specifies options that are related to scoring data. You can specify the following *options*:

INMODEL=CAS-libref.data-table

specifies the input data table for model information. The data table must be an output data table that is produced by the OUTMODEL= option in a previous call of the HMM procedure. When you specify this option, the IN= option in the ESTIMATE statement and the ID, INITIAL, MODEL, OPTIMIZE, and PRIOR statements are ignored. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

OUTMODEL=CAS-libref.data-table

writes the model information to the specified output data table. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

SMOOTH Statement

SMOOTH *option* ;

The SMOOTH statement specifies options that are related to the smoothing problem, which is to find the probability distribution of the state, conditional on all available observations. You can specify the following *option*:

OUT=CAS-libref.data-table

writes the smoothing results to the specified output data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

STORE Statement

STORE <OUT=>*CAS-libref.data-table* ;

The STORE statement saves the model information to the specified output data table that can be processed by the ASTORE procedure. An example of using analytic store technology is to perform a time-consuming model learning (that is, parameter estimation) and to store the learned model by using the STORE statement. Later you can perform the inference (namely, evaluating, filtering, smoothing, decoding, and forecasting) based on the saved model from the previous analysis, without having to learn the model again. For more information about the ASTORE procedure, see the chapter “The ASTORE Procedure” in *SAS Visual Data Mining and Machine Learning: Procedures*. The output data table consists of binary large object columns to store the binary data. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters, because the data type of each column is binary.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section “Using CAS Sessions and CAS Engine Librefs” on page 599.

Details: HMM Procedure

Hidden Markov Model

This section is a brief review of the hidden Markov model. For more information, see Rabiner (1989), Krolzig (1997), Frühwirth-Schnatter (2006), and Cappé, Moulines, and Rydén (2010).

The hidden Markov model (HMM) is a bivariate discrete time process $\{\mathbf{S}_t, \mathbf{Y}_t\}$, where $\{\mathbf{S}_t\}$ is a (hidden) Markov chain; conditional on $\{\mathbf{S}_t\}$, the observable process $\{\mathbf{Y}_t\}$ is a sequence of independent random variables such that the conditional distribution of $\mathbf{Y}_t$ depends only on $\mathbf{S}_t$. The $\mathbf{S}_t$ is often called the state. In general, the HMM can be expressed in three equations:

1. Initialization equation: $\mathbf{S}_1 \sim h_\theta(\cdot)$
2. Transition equation: $\mathbf{S}_t \sim f_\theta(\cdot|\mathbf{S}_{t-1})$
3. Observation equation: $\mathbf{Y}_t \sim g_\theta(\cdot|\mathbf{S}_t)$

Here the initial probability distribution function $h_\theta(\cdot)$ is the probability distribution function for initial state $\mathbf{S}_1$; the transition probability distribution function $f_\theta(\cdot|\mathbf{S}_{t-1})$ is the probability distribution function of current state $\mathbf{S}_t$ conditional on the past state $\mathbf{S}_{t-1}$; the observation probability distribution function $g_\theta(\cdot|\mathbf{S}_t)$ is the probability distribution function of current observation $\mathbf{Y}_t$ conditional on the current state $\mathbf{S}_t$; and all probability distribution functions might depend on the parameter θ .

There is no restriction on the dimensionality of $\mathbf{S}_t$ and $\mathbf{Y}_t$; that is, they might be scalars, vectors, or matrices.

If the transition probability distribution function $f_\theta(\cdot|\cdot)$ is time-independent (that is, the Markov chain is time-homogeneous), then the HMM is called the (time-)homogeneous HMM; otherwise, the HMM is called the (time-)inhomogeneous HMM.

The state $\mathbf{S}_t$ could take discrete or continuous values or even a mix of discrete and continuous values.² In the simplest and most popular case, $\mathbf{S}_t$ is a scalar and takes finite discrete values; that is, $\mathbf{S}_t \in \{1, \dots, K\}$. In this case, the HMM could be called the finite-state-space HMM.

For the finite-state-space HMM, the initial state distribution function $h_\theta(\cdot)$ can be expressed as a $K \times 1$ vector π , which is called the initial state probability vector (ISPV); that is,

$$\pi = \{\pi_i = p(\mathbf{S}_1 = i), i = 1, \dots, K\}$$

For the finite-state-space and homogeneous HMM, the transition probability distribution function $f_\theta(\cdot|\cdot)$ can be expressed as a $K \times K$ matrix $\mathbf{A}$, which is called the transition probability matrix (TPM), where an element

²In some literature, the HMM is referred to only as the discrete-state-space HMM (that is, the state can take only discrete values). The name “state space model” is used for the continuous-state-space HMM (that is, the state can take only continuous values).

a_{ij} of $\mathbf{A}$ denotes the probability transitioning from past state i to current state j ; that is,

$$a_{ij} = p(\mathbf{S}_t = j | \mathbf{S}_{t-1} = i)$$

There are six common problems to solve for an HMM:

- Filtering problem: What is $p(\mathbf{S}_t | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$?
- Smoothing problem: What is $p(\mathbf{S}_t | \mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the probability distribution of state $\mathbf{S}_t$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?
- Forecasting problem: What is $p(\mathbf{S}_{t+h} | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$, $h > 0$, the probability distribution of state $\mathbf{S}_{t+h}$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$?
- Evaluating problem: What is $p(\mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the probability (or the likelihood) of observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$?
- Decoding problem: What is $\arg \max_{\mathbf{S}_1, \dots, \mathbf{S}_T} p(\mathbf{S}_1, \dots, \mathbf{S}_T, \mathbf{Y}_1, \dots, \mathbf{Y}_T)$, the most likely sequence of hidden states $\mathbf{S}_1, \dots, \mathbf{S}_T$ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?
- Learning problem: How do you estimate the unknown parameter θ given observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$, where T is the sample size?

In most cases, the learning problem is the most difficult one. You might apply the maximum likelihood (ML) method or the maximum a posteriori (MAP) method to get the point estimate of parameter θ . For more information, see the section “[Parameter Estimation Methods](#)” on page 678.

Several algorithms are helpful in solving the problems for the HMMs. Three of them are the forward algorithm, the backward algorithm, and the Viterbi algorithm. They might be generalized to support many types of HMMs. The following sections focus on applying them to the finite-state-space and homogeneous HMM.

Forward Algorithm

The forward algorithm recursively calculates the joint probability distribution of a state at time t , $\mathbf{S}_t$, and all observations up to time t , $\mathbf{Y}_1, \dots, \mathbf{Y}_t$. First, define $\alpha_t(i)$, $i = 1, \dots, K$, as the joint probability of $(\mathbf{S}_t = i)$ and $\mathbf{Y}_1, \dots, \mathbf{Y}_t$:

$$\alpha_t(i) \equiv p(\mathbf{S}_t = i, \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

Then, $\alpha_t(i)$ can be recursively calculated by the following steps:

1. Calculate $\alpha_1(i)$, $i = 1, \dots, K$, by

$$\alpha_1(i) = \pi_i g_\theta(\mathbf{Y}_1 | \mathbf{S}_1 = i)$$

2. Calculate $\alpha_t(i), i = 1, \dots, K, t = 2, \dots, T$, by

$$\alpha_t(i) = \left(\sum_{j=1}^K \alpha_{t-1}(j) a_{ji} \right) g_{\theta}(\mathbf{Y}_t | \mathbf{S}_t = i)$$

You can solve the evaluating problem by

$$L_t \equiv p(\mathbf{Y}_1, \dots, \mathbf{Y}_t) = \sum_{j=1}^K \alpha_t(j)$$

Because the likelihood of all observations, L_T , can be calculated, you can apply the maximum likelihood method to estimate the unknown parameter θ ; if the prior distribution of the parameter θ is given, you can also apply the MAP method. That is, you might use the forward algorithm in the learning problem.

You can solve the filtering problem by

$$p_t^{(f)}(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \dots, \mathbf{Y}_t) = \alpha_t(i) / L_t$$

When the filtering problem is solved, you can solve the forecast problem, because the h -step-ahead prediction of the state probability can be calculated by

$$p_{t+h|t}^{(p)} = (\mathbf{A}')^h p_t^{(f)}$$

where $p_t^{(f)}$ is the $K \times 1$ vector with the i th element as $p_t^{(f)}(i)$, and $p_{t+h|t}^{(p)}$ is the $K \times 1$ vector with the i th element as $p_{t+h|t}^{(p)}(i)$:

$$p_{t+h|t}^{(p)}(i) \equiv p(\mathbf{S}_{t+h} = i | \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

The distribution of $\mathbf{Y}_{t+h}$ conditional on observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$ is the following mixture distribution:

$$\mathbf{Y}_{t+h} | \mathbf{Y}_1, \dots, \mathbf{Y}_t \sim \sum_{i=1}^K p_{t+h|t}^{(p)}(i) g_{\theta}(\cdot | \mathbf{S}_{t+h} = i)$$

Backward Algorithm

To solve the smoothing problem, in addition to the forward algorithm, you also need the backward algorithm. Define $\beta_t(i)$ as the probability of observing the future observations $\mathbf{Y}_{t+1}, \dots, \mathbf{Y}_T$ conditional on the current state $\mathbf{S}_t = i$:

$$\beta_t(i) \equiv p(\mathbf{Y}_{t+1}, \dots, \mathbf{Y}_T | \mathbf{S}_t = i)$$

The backward algorithm recursively calculates $\beta_t(i)$ as follows:

1. Set $\beta_T(i) = 1, i = 1, \dots, K$.
2. Calculate $\beta_t(i), i = 1, \dots, K, t = T - 1, \dots, 1$, by

$$\beta_t(i) = \sum_{j=1}^K a_{ij} g_{\theta}(\mathbf{Y}_{t+1} | \mathbf{S}_{t+1} = j) \beta_{t+1}(j)$$

When you calculate both $\alpha_t(i)$ and $\beta_t(i), i = 1, \dots, K, t = 1, \dots, T$, by using the forward algorithm and backward algorithm, the smoothing problem is solved, because you can calculate the probability of state $\mathbf{S}_t$ taking value i given all observations $\mathbf{Y}_1, \dots, \mathbf{Y}_T$ as follows:

$$\gamma_t(i) \equiv p(\mathbf{S}_t = i | \mathbf{Y}_1, \dots, \mathbf{Y}_T) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^K \alpha_t(j) \beta_t(j)}$$

Viterbi Algorithm

You use the Viterbi algorithm to solve the decoding problem. First, define the highest probability of a single state path ending in state $\mathbf{S}_t = i$ and observations $\mathbf{Y}_1, \dots, \mathbf{Y}_t$ as follows:

$$V_t(i) = \max_{\mathbf{S}_1, \dots, \mathbf{S}_{t-1}} p(\mathbf{S}_1, \dots, \mathbf{S}_{t-1}, \mathbf{S}_t = i, \mathbf{Y}_1, \dots, \mathbf{Y}_t)$$

To keep track of the best path, for $t = 2, \dots, T$, define

$$v_t(i) = \arg \max_j V_{t-1}(j) a_{ji}$$

Then, $V_t(i)$ and $v_t(i)$ can be recursively calculated as follows:

1. Calculate $V_1(i), i = 1, \dots, K$, by

$$V_1(i) = \pi_i g_{\theta}(\mathbf{Y}_1 | \mathbf{S}_1 = i)$$

2. Calculate $V_t(i)$ and $v_t(i), i = 1, \dots, K, t = 2, \dots, T$, by

$$\begin{aligned} V_t(i) &= \left(\max_j V_{t-1}(j) a_{ji} \right) g_{\theta}(\mathbf{Y}_t | \mathbf{S}_t = i) \\ v_t(i) &= \arg \max_j V_{t-1}(j) a_{ji} \end{aligned}$$

Then, the probability of the best path is

$$V^* = \max_i V_T(i)$$

Define $v_T^* = \arg \max_i V_T(i)$, and the best state path can be backtracked by

$$v_t^* = v_{t+1}(v_{t+1}^*), t = T - 1, \dots, 1$$

That is, $\{v_t^*\}_{t=1, \dots, T}$ is the best state path. The decoding problem is solved.

Gaussian Hidden Markov Model

The Gaussian hidden Markov model (Gaussian HMM) is a type of finite-state-space and homogeneous HMM where the observation probability distribution is the normal distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim N(\mu_{\mathbf{S}_t}, \Sigma_{\mathbf{S}_t})$$

where $\mu_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t$, $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mu_i, \Sigma_i\}_{i=1, \dots, K}$, which consists of mean and covariance parameters) together specify the Gaussian HMM; that is, the parameter θ of the Gaussian HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the Gaussian HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 668. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Gaussian Mixture Hidden Markov Model

The Gaussian mixture hidden Markov model (GM HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the Gaussian mixture distribution,

$$\mathbf{Y}_t | \mathbf{S}_t \sim \text{GM}(\{w_{\mathbf{S}_t,1}, \dots, w_{\mathbf{S}_t,M}\}, \{\mu_{\mathbf{S}_t,1}, \dots, \mu_{\mathbf{S}_t,M}\}, \{\Sigma_{\mathbf{S}_t,1}, \dots, \Sigma_{\mathbf{S}_t,M}\})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\text{GM}(\dots)$ represents the Gaussian mixture distribution; M is the number of components at each state; and the $w_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_{\mathbf{S}_t,j} \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_{\mathbf{S}_t,j} = 1$). The $\mu_{\mathbf{S}_t,j}$ and $\Sigma_{\mathbf{S}_t,j}$, $j = 1, \dots, M$, are the mean and covariance parameters for the j th Gaussian component at state $\mathbf{S}_t$, and their values depend on the variable $\mathbf{S}_t$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \dots, K, j = 1, \dots, M\}$, which consists of the mixture component probabilities, the mean and covariance parameters for all components) together specify the GM HMM; that is, the parameter θ of the GM HMM is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Because the GM HMM is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 668. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Regression Model

The regime-switching regression model (RS REG, also known as the regression hidden Markov model or REG HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are the mean and covariance parameters, respectively, at state $\mathbf{S}_t$, $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the observation parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1,\dots,K}$, which consists of mean and covariance parameters) together specify the RS REG; that is, the parameter θ of the RS REG is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, and exogenous variables and their lagged values. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \dots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}$, $i = 1, \dots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:

- If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \dots x_{k_x,t} \ x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

The statement

```
model y = x1 x2 x3 / type=reg nstate=2;
```

defines the regression for state $i, i = 1, \dots, K$ ($K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x1_t + b_{i,3}x2_t + b_{i,4}x3_t + \varepsilon_t \\ &= (b_{i,1} \ b_{i,2} \ b_{i,3} \ b_{i,4})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1 \ x1_t \ x2_t \ x3_t)'$, and $k_z = 4$.

In the following statement, $z_t = (1 \ sd_t^{(1)} \ sd_t^{(2)} \ sd_t^{(3)} \ t \ t^2 \ x1_t \ x2_t \ x3_t \ x1_{t-1} \ x2_{t-1} \ x3_{t-1})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time t and where $k_z = 12$:

```
model y = x1 x2 x3 / trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1} \ x2_{t-1} \ x3_{t-1})'$ and $k_z = 3$:

```
model y = x1 x2 x3 / noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y , can be greater than 1; in that case, $\mathbf{B}_i, i = 1, \dots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, and $z_t^{(\text{XL})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \ \dots \ \Sigma_K)'$, where K is the number of states.

The Gaussian HMM can be regarded as a special case of the RS REG when there is only one regressor constant.

Because the RS REG is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 668. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Autoregression Model

The regime-switching autoregression model (RS AR, also known as the autoregressive hidden Markov model or AR HMM) is a type of finite-state-space and homogeneous HMM in which the observation probability distribution is the normal distribution conditional on the specified regressors. For a standard form of RS AR,

$$\mathbf{Y}_t | z_t, \mathbf{S}_t \sim N(\mathbf{B}_{\mathbf{S}_t} z_t, \Sigma_{\mathbf{S}_t})$$

where the variable $\mathbf{S}_t$ is the (hidden) state; $\mathbf{B}_{\mathbf{S}_t}$ and $\Sigma_{\mathbf{S}_t}$ are mean and covariance parameters at state $\mathbf{S}_t, \mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B}$ ($\equiv \{\mathbf{B}_i, \Sigma_i\}_{i=1, \dots, K}$, which consists of mean and covariance parameters) together specify the RS AR; that is, the parameter θ of the RS AR is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables and their lagged values, and the lagged values of the dependent variables. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \left(z_t^{(\text{AR})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \dots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}$, $i = 1, \dots, n_s - 1$, denote the values of seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model, or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:
 - If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \dots x_{k_x,t} \ x_{1,t-1} \dots x_{k_x,t-1} \dots x_{1,t-s} \dots x_{k_x,t-s})'$$

- $z_t^{(\text{AR})}$ is empty if the YLAG= option is not specified or YLAG=0 is specified; otherwise, $z_t^{(\text{AR})}$ is determined as follows, where YLAG= p and k_y is the number of dependent variables:

$$z_t^{(\text{AR})} = (y_{1,t-1} \dots y_{k_y,t-1} \dots y_{1,t-p} \dots y_{k_y,t-p})'$$

The statement

```
model y = x1 x2 x3 / ylag=2 type=ar nstate=2;
```

defines the autoregression for state i , $i = 1, \dots, K$ ($K = 2$ in this case), as

$$\begin{aligned} y_t &= b_{i,1} + b_{i,2}x_{1t} + b_{i,3}x_{2t} + b_{i,4}x_{3t} + b_{i,5}y_{t-1} + b_{i,6}y_{t-2} + \varepsilon_t \\ &= (b_{i,1} \ b_{i,2} \ b_{i,3} \ b_{i,4} \ b_{i,5} \ b_{i,6})z_t + \varepsilon_t \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_i)$, $z_t = (1 \ x_{1t} \ x_{2t} \ x_{3t} \ y_{t-1} \ y_{t-2})'$, and $k_z = 6$.

In the following statement, $z_t = (1 \ sd_t^{(1)} \ sd_t^{(2)} \ sd_t^{(3)} \ t \ t^2 \ x_{1t} \ x_{2t} \ x_{3t} \ x_{1,t-1} \ x_{2,t-1} \ x_{3,t-1} \ y_{t-1} \ y_{t-2})'$, where $sd_t^{(i)}$, $i = 1, 2, 3$, denote the values of seasonal dummies at time t , and $k_z = 14$:

```
model y = x1 x2 x3 / ylag=2 trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x1_{t-1} \ x2_{t-1} \ x3_{t-1} \ y1_{t-1} \ y2_{t-1} \ y1_{t-2} \ y2_{t-2})'$ and $k_z = 7$:

```
model y1 y2 = x1 x2 x3 / ylag=2 noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, k_y , can be greater than 1 (for example, $k_y = 2$ in the previous statement); in that case, $\mathbf{B}_i, i = 1, \dots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, **XL**, and **AR** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, $z_t^{(\text{XL})}$, and $z_t^{(\text{AR})}$, respectively; the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \ \dots \ \Sigma_K)'$, where K is the number of states.

The RS REG can be regarded as a special case of the RS AR when YLAG=0 is specified.

Because the RS AR is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms introduced in the section “[Hidden Markov Model](#)” on page 668. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Regime-Switching Mean-Adjusted Autoregression Model

The regime-switching mean-adjusted autoregression model (also known as the mean-adjusted autoregressive hidden Markov model) is a type of finite-state-space and homogeneous HMM. For a regime-switching mean-adjusted autoregression model, the observation $\mathbf{Y}_t$ follows the normal distribution as

$$\mathbf{Y}_t | \mathbf{Y}_t, \dots, \mathbf{Y}_{t-p}, z_t, \dots, z_{t-p}, \mathbf{S}_t, \mathbf{S}_{t-1}, \dots, \mathbf{S}_{t-p} \sim N(\mathbf{B}_{\mathbf{S}_t}^{(R)} z_t + \sum_{\tau=1}^p \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(A)} (\mathbf{Y}_{t-\tau} - \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(R)} z_{t-p}), \Sigma_{\mathbf{S}_t})$$

where p is the autoregressive order, the variable $\mathbf{S}_t$ is the (hidden) state, $\mathbf{B}_{\mathbf{S}_{t-\tau}}^{(A)}$ is the autoregressive part of mean parameters at state $\mathbf{S}_t$, $\mathbf{B}_{\mathbf{S}_t}^{(R)}$ is the remaining mean parameters at state $\mathbf{S}_t$, and $\Sigma_{\mathbf{S}_t}$ is the covariance parameters at state $\mathbf{S}_t$ for $\mathbf{S}_t = 1, \dots, K$. Hence, the initial state probability vector (ISPV) π , the transition probability matrix (TPM) $\mathbf{A}$, and the parameter $\mathbf{B} (\equiv \{\mathbf{B}_i^{(A)}, \mathbf{B}_i^{(R)}, \Sigma_i\}_{i=1, \dots, K})$ together specify the regime-switching mean-adjusted autoregression model; that is, the parameter θ of the regime-switching mean-adjusted autoregression model is $\{\pi, \mathbf{A}, \mathbf{B}\}$.

Let k_z denote the number of regressors. The regressors, a $k_z \times 1$ vector z_t , might contain the constant, seasonal dummies, linear time trend, quadratic time trend, exogenous variables. That is,

$$z_t = \left(\left(z_t^{(\text{CONST})} \right)' \left(z_t^{(\text{SD})} \right)' \left(z_t^{(\text{LTREND})} \right)' \left(z_t^{(\text{QTREND})} \right)' \left(z_t^{(\text{XL})} \right)' \right)'$$

where

- $z_t^{(\text{CONST})}$ is empty if the NOINT option is specified; otherwise, $z_t^{(\text{CONST})} = (1)$.
- $z_t^{(\text{SD})}$ is empty if the NSEASON= option is not specified; otherwise, $z_t^{(\text{SD})} = (sd_t^{(1)} \cdots sd_t^{(n_s-1)})'$, where $sd_t^{(i)}, i = 1, \dots, n_s - 1$, denote the values of the seasonal dummies at time t and where n_s is the value specified in the NSEASON= n_s option.
- $z_t^{(\text{LTREND})}$ is empty if the TREND= option is not specified; otherwise, $z_t^{(\text{LTREND})} = (t)$.
- $z_t^{(\text{QTREND})}$ is empty if the TREND=QUAD option is not specified; otherwise, $z_t^{(\text{QTREND})} = (t^2)$.
- $z_t^{(\text{XL})}$ is empty if no exogenous variables are specified in the model or if the NOCURRENTX option is specified when the XLAG option is not specified or XLAG=0; otherwise, $z_t^{(\text{XL})}$ is determined as follows, where XLAG= s and k_x is the number of exogenous variables:

– If the NOCURRENTX option is specified,

$$z_t^{(\text{XL})} = (x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$

– If the NOCURRENTX option is not specified,

$$z_t^{(\text{XL})} = (x_{1,t} \cdots x_{k_x,t} \ x_{1,t-1} \cdots x_{k_x,t-1} \cdots x_{1,t-s} \cdots x_{k_x,t-s})'$$

The ARMEAN=ADJUSTED option in the following statement specifies the following regime-switching mean-adjusted autoregression model:

```
model y = x / armean=adjusted ylag=1 type=ar nstate=2;
```

$$\begin{aligned} y_t &= c_{\mathbf{S}_t} + b_{\mathbf{S}_t} x_t + a_{\mathbf{S}_t} (y_{t-1} - c_{\mathbf{S}_{t-1}} - b_{\mathbf{S}_{t-1}} x_{t-1}) + \varepsilon_t \\ &= (c_{\mathbf{S}_t} \ b_{\mathbf{S}_t}) z_t + a_{\mathbf{S}_t} (y_{t-1} - (c_{\mathbf{S}_{t-1}} \ b_{\mathbf{S}_{t-1}}) z_{t-1}) + \varepsilon_t \\ &= \begin{cases} c_1 + b_1 x_t + a_1 (y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 1, \\ c_2 + b_2 x_t + a_2 (y_{t-1} - c_1 - b_1 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 1, \\ c_1 + b_1 x_t + a_1 (y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 1, \mathbf{S}_{t-1} = 2, \\ c_2 + b_2 x_t + a_2 (y_{t-1} - c_2 - b_2 x_{t-1}) + \varepsilon_t, & \mathbf{S}_t = 2, \mathbf{S}_{t-1} = 2. \end{cases} \end{aligned}$$

where $\varepsilon_t \sim N(0, \Sigma_{\mathbf{S}_t})$, $z_t = (1 \ x_t)'$, and $k_z = 2$.

The number of dependent variables, k_y , can be greater than 1; in that case, $\Sigma_i, i = 1, \dots, K$, is the $k_y \times k_y$ symmetric positive definite matrix. When $k_y > 1$, the model is called the vector autoregressive hidden Markov model (VAR HMM) in some literature.

In the INITIAL statement, the **CONST**, **SD**, **LTREND**, **QTREND**, and **XL** functions refer to the mean parameters that correspond to $z_t^{(\text{CONST})}$, $z_t^{(\text{SD})}$, $z_t^{(\text{LTREND})}$, $z_t^{(\text{QTREND})}$, $z_t^{(\text{XL})}$, respectively; the **AR** function refers to the mean parameters that correspond to demeaned lagged dependent variables $(\mathbf{Y}_{t-\tau} - \mathbf{B}_{\mathbf{S}_{t-\tau}}^{(R)} z_{t-\tau}), \tau = 1, \dots, p$; and the **COV** function refers to the covariance parameters $\Sigma = (\Sigma_1 \cdots \Sigma_K)'$, where K is the number of states.

Because the regime-switching mean-adjusted autoregression model is a type of finite-state-space and homogeneous HMM, the six common problems—the filtering, smoothing, forecasting, evaluating, decoding, and learning problems—can be solved using the three algorithms that are introduced in the section “[Hidden Markov Model](#)” on page 668. That is, you can solve the evaluating, filtering, and forecasting problems by using the forward algorithm; the smoothing problem by using the forward algorithm and backward algorithm; the decoding problem by using the Viterbi algorithm; and the learning problem, if solved through the maximum likelihood or maximum a posteriori method, by using the forward algorithm to calculate the likelihood.

Parameter Estimation Methods

For the parameter estimation (the learning problem) of the HMM, the HMM procedure supports two methods: the maximum likelihood (ML) method and the maximum a posteriori (MAP) method.

Maximum Likelihood (ML) Method

For the ML method, the parameter estimator is obtained as

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(\mathbf{Y}_1, \dots, \mathbf{Y}_T; \theta))$$

where $L_T(\cdot; \theta)$ is the likelihood of all observations for a particular parameter θ . The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

Maximum a Posteriori (MAP) Method

For the MAP method, the parameter estimator is obtained as

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(\mathbf{Y}_1, \dots, \mathbf{Y}_T; \theta)) + \log (p(\theta))$$

where $p(\theta)$ is the prior distribution function of parameter θ . The prior distribution can be specified in the PRIOR statement. The $\hat{\theta}$ is obtained through gradient-based optimization; the covariance matrix estimator of the $\hat{\theta}$ is obtained through the inverse of the negative Hessian.

In theory, the likelihood of the hidden Markov model (HMM) is unbounded and “the ML estimator as a global maximizer of the likelihood function does not exist” (Frühwirth-Schnatter 2006, page 173). The introduction of the proper prior distribution of the parameters in the MAP method can solve the unboundedness problem.

Prior Distribution of the Initial State Probability Vector

The prior distribution of the initial state probability vector (ISPV), π , is the Dirichlet distribution,

$$\pi \sim \text{Dir}(\alpha)$$

where the hyperparameter α is the $K \times 1$ vector and K is the number of states. The probability density function (PDF) of the Dirichlet distribution is

$$f(x, \alpha) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i-1}$$

where $x = (x_1 \cdots x_K)'$, $x_i > 0, i = 1, \dots, K$, and $\sum_{i=1}^K x_i = 1$; $\alpha = (\alpha_1 \cdots \alpha_K)'$ and $\alpha_i > 0, i = 1, \dots, K$; $B(\alpha)$ is the beta function,

$$B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}$$

Prior Distribution of the Transition Probability Matrix

The prior distribution of each row of the transition probability matrix (TPM), $\mathbf{A}$, is the Dirichlet distribution,

$$a_i' \sim \text{Dir}(\alpha^{(i)}), i = 1, \dots, K$$

where a_i is the i th row of the TPM $\mathbf{A}$, $\alpha^{(i)}$ is the $K \times 1$ vector, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{A} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \cdots \alpha^{(K)})'$, which is the $K \times K$ matrix that results from stacking the transpose of $\alpha^{(i)}, i = 1, \dots, K$.

Prior Distribution of the Categorical Probability Matrix

The prior distribution of each row of the categorical probability matrix (CPM), $\mathbf{B}$, is the Dirichlet distribution,

$$b_i' \sim \text{Dir}(b^{(i)}), i = 1, \dots, K$$

where b_i is the i th row of the CPM, $\mathbf{B}$; $b^{(i)}$ is the $M \times 1$ vector; M is the number of categories at a state; and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{B} \sim \text{Dir}(\tilde{\mathbf{B}})$$

where the hyperparameter $\tilde{\mathbf{B}} = (b^{(1)} \cdots b^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $b^{(i)}, i = 1, \dots, K$.

Prior Distribution of the Mixture Component Probabilities

The prior distribution of each row of the mixture component probabilities (MCP), $\mathbf{W}$, is the Dirichlet distribution,

$$w'_i \sim \text{Dir}(\alpha^{(i)}), i = 1, \dots, K$$

where w_i is the i th row of the MCP $\mathbf{W}$, $\alpha^{(i)}$ is the $M \times 1$ vector, M is the number of components at a state, and K is the number of states. To simplify notation, the following matrix form of the Dirichlet distribution is used,

$$\mathbf{W} \sim \text{Dir}(\tilde{\mathbf{A}})$$

where the hyperparameter $\tilde{\mathbf{A}} = (\alpha^{(1)} \dots \alpha^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $\alpha^{(i)}$, $i = 1, \dots, K$.

Prior Distribution of the Observation Parameters

Gaussian HMM In the Gaussian HMM, the observation parameters are $\{\mu_i, \Sigma_i\}$, $i = 1, \dots, K$, where μ_i and Σ_i are the mean and covariance parameters for state i . The prior distribution of μ_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\mu_i, \Sigma_i \sim \text{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \dots, K$$

which is equivalent to

$$\begin{aligned} \mu_i | \Sigma_i &\sim N(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \Sigma_i) \\ \Sigma_i &\sim \text{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i) \end{aligned}$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a positive scalar; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned} f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1} \Sigma|}} \exp\left(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1} \Sigma)^{-1}(\mu - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k\left(\frac{\tilde{\nu}}{2}\right)} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\tilde{\Sigma}\Sigma^{-1})\right) \end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function, which is defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}\tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\ \tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\ \tilde{\nu} &= (\tilde{\nu}_1 \cdots \tilde{\nu}_K)'\end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $K \times 1$ vector, $\tilde{\nu}$ is a $K \times 1$ vector, K is the number of states, and k_y is the number of dependent variables.

GM HMM In the GM HMM, the mean and covariance parameters for component j at state i are $\{\mu_{ij}, \Sigma_{ij}\}$, $i = 1, \dots, K$, $j = 1, \dots, M$, where M is the number of components at a state and K is the number of states.

The prior distribution of μ_{ij} and Σ_{ij} is the normal-inverse-Wishart (NIW) distribution,

$$\mu_{ij}, \Sigma_{ij} \sim \text{NIW}(\tilde{\mu}_{ij}, \tilde{\Sigma}_{ij}, \tilde{\kappa}_{ij}, \tilde{\nu}_{ij}), i = 1, \dots, K, j = 1, \dots, M$$

which is equivalent to

$$\begin{aligned}\mu_{ij} | \Sigma_{ij} &\sim N(\tilde{\mu}_{ij}, \tilde{\kappa}_{ij}^{-1} \Sigma_{ij}) \\ \Sigma_{ij} &\sim \text{IW}(\tilde{\Sigma}_{ij}, \tilde{\nu}_{ij})\end{aligned}$$

where the hyperparameter $\tilde{\mu}_{ij}$ is a $k_y \times 1$ vector; the hyperparameter $\tilde{\Sigma}_{ij}$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_{ij}$ is a positive scalar; the hyperparameter $\tilde{\nu}_{ij}$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; and k_y is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) &= \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1} \Sigma|}} \exp\left(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1} \Sigma)^{-1}(\mu - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu}k}{2}} \Gamma_k(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu}+k+1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\tilde{\Sigma} \Sigma^{-1})\right)\end{aligned}$$

where the parameter μ is a $k \times 1$ vector; the parameter Σ is a $k \times k$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a positive scalar; and the hyperparameter $\tilde{\nu}$ is a positive scalar, $\tilde{\nu} > k + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}\tilde{\mu} &= (\tilde{\mu}_{11} \cdots \tilde{\mu}_{1M} \tilde{\mu}_{21} \cdots \tilde{\mu}_{2M} \cdots \tilde{\mu}_{K1} \cdots \tilde{\mu}_{KM})' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_{11} \cdots \tilde{\Sigma}_{1M} \tilde{\Sigma}_{21} \cdots \tilde{\Sigma}_{2M} \cdots \tilde{\Sigma}_{K1} \cdots \tilde{\Sigma}_{KM})' \\ \tilde{\kappa} &= (\tilde{\kappa}_{11} \cdots \tilde{\kappa}_{1M} \tilde{\kappa}_{21} \cdots \tilde{\kappa}_{2M} \cdots \tilde{\kappa}_{K1} \cdots \tilde{\kappa}_{KM})' \\ \tilde{\nu} &= (\tilde{\nu}_{11} \cdots \tilde{\nu}_{1M} \tilde{\nu}_{21} \cdots \tilde{\nu}_{2M} \cdots \tilde{\nu}_{K1} \cdots \tilde{\nu}_{KM})'\end{aligned}$$

where $\tilde{\mu}$ is a $KM \times k_y$ matrix, $\tilde{\Sigma}$ is a $k_y KM \times k_y$ matrix, $\tilde{\kappa}$ is a $KM \times 1$ vector, $\tilde{\nu}$ is a $KM \times 1$ vector, M is the number of components at a state, K is the number of states, and k_y is the number of dependent variables.

Regime-Switching Regression Model and Regime-Switching Autoregression Model In regime-switching regression models and regime-switching autoregression models, the observation parameters are $\{\mathbf{B}_i, \Sigma_i\}, i = 1, \dots, K$, where $\mathbf{B}_i$ and Σ_i are the mean and covariance parameters for state i . Because $\mathbf{B}_i z_t = (z_t' \otimes I_{k_y}) \text{vec}(\mathbf{B}_i) = (z_t' \otimes I_{k_y}) \beta_i$, where vec is the column stacking operator, the observation parameters in this section are denoted by $\{\beta_i, \Sigma_i\}, i = 1, \dots, K$. The prior distribution of β_i and Σ_i is the normal-inverse-Wishart (NIW) distribution,

$$\beta_i, \Sigma_i \sim \text{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\nu}_i), i = 1, \dots, K$$

which is equivalent to

$$\begin{aligned}\beta_i | \Sigma_i &\sim N(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \otimes \Sigma_i) \\ \Sigma_i &\sim \text{IW}(\tilde{\Sigma}_i, \tilde{\nu}_i)\end{aligned}$$

where the hyperparameter $\tilde{\mu}_i$ is a $k_y k_z \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k_y \times k_y$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a $k_z \times k_z$ symmetric positive definite matrix; the hyperparameter $\tilde{\nu}_i$ is a positive scalar, $\tilde{\nu}_i > k_y + 3$; K is the number of states; k_y is the number of dependent variables; and k_z is the number of regressors. For information about how to calculate k_z , see the sections “[Regime-Switching Regression Model](#)” on page 673 and “[Regime-Switching Autoregression Model](#)” on page 674.

The probability density function (PDF) of the NIW distribution is

$$\begin{aligned}f_{\text{NIW}}(\beta, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\nu}) &= f_N(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) \\ f_N(\beta, \tilde{\mu}, \tilde{\kappa}^{-1} \otimes \Sigma) &= \frac{1}{\sqrt{(2\pi)^{k_1 k_2} |\tilde{\kappa}^{-1} \otimes \Sigma|}} \exp\left(-\frac{1}{2}(\beta - \tilde{\mu})'(\tilde{\kappa}^{-1} \otimes \Sigma)^{-1}(\beta - \tilde{\mu})\right) \\ f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\nu}) &= \frac{|\tilde{\Sigma}|^{\frac{\tilde{\nu}}{2}}}{2^{\frac{\tilde{\nu} k_1}{2}} \Gamma_{k_1}(\frac{\tilde{\nu}}{2})} |\Sigma|^{-\frac{\tilde{\nu} + k_1 + 1}{2}} \exp\left(-\frac{1}{2} \text{tr}(\tilde{\Sigma} \Sigma^{-1})\right)\end{aligned}$$

where the parameter β is a $k_1 k_2 \times 1$ vector; the parameter Σ is a $k_1 \times k_1$ symmetric positive matrix; the hyperparameter $\tilde{\mu}$ is a $k_1 k_2 \times 1$ vector; the hyperparameter $\tilde{\Sigma}$ is a $k_1 \times k_1$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}$ is a $k_2 \times k_2$ symmetric positive definite matrix; and the hyperparameter $\tilde{\nu}$ is a positive scalar, where $\tilde{\nu} > k_1 + 3$. The $\Gamma_k(\cdot)$ is the multivariate gamma function defined as

$$\Gamma_k(x) = \pi^{\frac{k(k-1)}{4}} \prod_{j=1}^k \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\begin{aligned}\tilde{\mu} &= (\tilde{\mu}_1 \cdots \tilde{\mu}_K)' \\ \tilde{\Sigma} &= (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)' \\ \tilde{\kappa} &= (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)' \\ \tilde{v} &= (\tilde{v}_1 \cdots \tilde{v}_K)'\end{aligned}$$

where $\tilde{\mu}$ is a $K \times k_y k_z$ matrix, $\tilde{\Sigma}$ is a $k_y K \times k_y$ matrix, $\tilde{\kappa}$ is a $k_z K \times k_z$ matrix, $\tilde{v}$ is a $K \times 1$ vector, K is the number of states, k_y is the number of dependent variables, and k_z is the number of regressors.

Optimization Algorithms

The HMM procedure supports three types of optimization algorithms: the active set algorithm, the interior point algorithm, and the stochastic gradient descent (SGD) algorithm. For more information about the active set and interior point algorithms and their related options, see the chapter “The Nonlinear Programming Solver” in *SAS/OR User’s Guide: Mathematical Programming*. For more information about the SGD algorithm, see the section “OPTIMIZE Statement” on page 661.

Matrix Expression

The INITIAL and PRIOR statements operate on matrices. That is, you can specify the parameter matrices or constant matrices through the INITIAL and PRIOR statements’ built-in operators and functions. You can add elements of the matrices **A** and **B** by using the expression **A+B**, perform matrix multiplication by using the expression **A*B**, and perform elementwise multiplication by using the expression **A#B**. You can get the diagonal elements of the matrix **A** by using the function **DIAG(A)**, and you can get the $n \times n$ identity matrix by using the function **I(n)**.

Each equation is written as a matrix expression that is composed of constants, operators, and functions.

Constants

Constants are either scalar constants (such as -1.2, 0.3, and so on) or matrix constants enclosed in braces (such as the 2×2 matrix {1 2, 3 4} or the 1×3 row vector {-0.2 5.3 12}).

The matrix constant cannot be the first item in the INITIAL statement unless you enclose it in parentheses. For example, you cannot specify the following statement:

```
initial {-0.1 -0.2, -0.3 -0.4} = MU;
```

You can specify the preceding example by enclosing the first matrix constant in parentheses, as follows:

```
initial ({-0.1 -0.2, -0.3 -0.4}) = MU;
```

Operators

Operators define the operations on operands. [Table 13.2](#) lists all built-in operators that the INITIAL statement supports.

Table 13.2 Operators

Operator	Name	Description
+	Addition	Adds corresponding matrix elements
=	Comparison, equal	Compares matrix elements
<	Comparison, less than	Compares matrix elements
<=	Comparison, not greater than	Compares matrix elements
>	Comparison, greater than	Compares matrix elements
>=	Comparison, not less than	Compares matrix elements
	Concatenation, horizontal	Concatenates matrices horizontally
//	Concatenation, vertical	Concatenates matrices vertically
@	Direct product	Takes the direct product of two matrices
~	Distribution	Links the distribution function to the matrix elements
:	Index creation	Creates an index vector
#	Multiplication, elementwise	Performs elementwise multiplication
*	Multiplication, matrix	Performs matrix multiplication
—	Sign reverse	Reverses the signs of elements
[]	Subscripts	Selects submatrices
—	Subtraction	Subtracts corresponding matrix elements
`	Transpose	Transposes a matrix

For more information about each operator, see the section “[Details of Operators](#)” on page 688.

[Table 13.3](#) shows the precedence of matrix operators.

Table 13.3 Precedence of Operators

Priority Group	Operators
I (highest)	[] (subscripts) ` (transpose)
II	— (sign reverse)
III	* # @
IV	— (subtraction) +
V	// :
VI (lowest)	= < <= > >= ~

Each equation can be a compound expression that involves several matrix operators and operands. The rules for evaluating compound expressions are as follows:

- Evaluation follows the order of operator precedence as described in [Table 13.3](#). Group I has the highest priority; that is, Group I operators are evaluated first. Group II operators are evaluated after Group I operators, and so on. For example, $1 + 2 * 3$ returns 7.

- If neighboring operators in an expression have equal precedence, the expression is evaluated from left to right, except for the Group I operators. For example, $1 - 2 - 3$ returns -4 .
- All expressions in parentheses are evaluated first, following the two preceding rules. For example, $3 * (2 + 1)$ returns 9.

Functions

Functions are divided into two types. One type of function refers to parameters to be estimated; examples are **MU**(**K**, **I**) and **SIGMA**(**K**, **I**, **J**). The other type does not; examples are **I**(**n**) and **DIAG**(**A**).

Functions that refer to parameters are listed in Table 13.4. The arguments of functions can be matrices. The simplest case, scalar arguments, is discussed first. For convenience, the scalar indices **i** and **j** refer to the position of the element in the coefficient matrix, the scalar **l** refers to the lag value, the scalar **c** refers to the component index, and the scalar **k** refers to the state value.

Table 13.4 Functions That Refer to Parameters

Function	Description
AR (k , l , i , j)	Autoregressive parameter of the lag l of the j th dependent variable, $y_{j,t-l}$, to the i th dependent variable at time t , y_{it} , for state k in regime-switching autoregression
CONST (k , i)	Intercept parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression, or the alias of MU for Gaussian HMM or GM HMM
COV (k , i , j)	Covariance of innovations parameter between the i th and j th error processes for state k in regime-switching regression or regime-switching autoregression, or the alias of SIGMA for Gaussian HMM or GM HMM
ISPV (i)	The i th element in the initial state probability vector
LTREND (k , i)	Linear trend parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression when the TREND = option is specified in the MODEL statement
MCP (i , j)	The (i, j) th element in the mixture component probabilities in the GM HMM
MU (k , i)	Mean parameter of the i th time series, y_{it} , for state k in the Gaussian HMM, or alias for CONST for regime-switching regression or regime-switching autoregression
MU (k , c , i)	Mean parameter of the i th time series, y_{it} , for the c th component at state k in the GM HMM
QTREND (k , i)	Quadratic trend parameter of the i th time series, y_{it} , for state k in regime-switching regression or regime-switching autoregression when TREND = QUAD is specified in the MODEL statement
SD (k , i , j)	Alias of SDUMMY
SDUMMY (k , i , j)	The j th seasonal dummy of the i th time series at time t , y_{it} , for state k in regime-switching regression or regime-switching autoregression, where $j = 1, \dots, (nseason-1)$, where $nseason$ is the value of the NSEASON = option in the MODEL statement

Table 13.4 continued

Function	Description
SIGMA (<i>k</i> , <i>i</i> , <i>j</i>)	Covariance parameter between the <i>i</i> th and <i>j</i> th time series, y_{it} and y_{jt} , for state <i>k</i> in the Gaussian HMM, or the alias of COV for regime-switching regression or regime-switching autoregression
SIGMA (<i>k</i> , <i>c</i> , <i>i</i> , <i>j</i>)	Covariance parameter between the <i>i</i> th and <i>j</i> th time series, y_{it} and y_{jt} , for <i>c</i> th component at state <i>k</i> in the GM HMM
TPM (<i>i</i> , <i>j</i>)	The (<i>i</i> , <i>j</i>)th element in the transition probability matrix
XL (<i>k</i> , <i>l</i> , <i>i</i> , <i>j</i>)	Exogenous parameter of the lag <i>l</i> of the <i>j</i> th exogenous variable, $x_{j,t-l}$, to the <i>i</i> th dependent variable at time <i>t</i> , y_{it} , for state <i>k</i> in regime-switching regression or regime-switching autoregression

The functions that refer to parameters, as shown in Table 13.4, are valid only when the parameters are included in the model. For example, the **ISPV** function is valid only when you specify the ESTISPV option in the MODEL statement; otherwise, the HMM procedure issues errors and then stops.

The functions that refer to parameters, as shown in Table 13.4, accept vector and matrix arguments and return the matrix that is constructed by the corresponding parameters. According to the number of arguments, the following list shows what matrix a function returns when the arguments are vectors:

- A function, **FUNC**₁, that has one vector argument *I*, where $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, returns a vector $R = (r_1 \ r_2 \ \dots \ r_{n_I})'$, where $r_k = \text{FUNC}_1(i_k)$, $k = 1, \dots, n_I$. **ISPV** is a type of **FUNC**₁.
- A function, **FUNC**₂, that has two vector arguments *I* and *J*, where $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$ and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \cdots & & & \\ r_{n_I,1} & r_{n_I,2} & \cdots & r_{n_I,n_J} \end{pmatrix}$$

where $r_{k,m} = \text{FUNC}_2(i_k, j_m)$, $k = 1, \dots, n_I$, $m = 1, \dots, n_J$. **CONST**, **LTREND**, **MU**, **QTREND**, and **TPM** are types of **FUNC**₂.

- A function, **FUNC**_N, that has *N* vector arguments $I_1, I_2, \dots$, and I_N , where $N > 2$ and *N* is odd and where $I_n = (i_{n,1} \ i_{n,2} \ \dots \ i_{n,j_n})'$, $n = 1, \dots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_1 \\ B_2 \\ \cdots \\ B_{j_n} \end{pmatrix}$$

where $B_k = \text{FUNC}_N(i_{1,k}, I_2, \dots, I_N)$, $k = 1, \dots, j_1$, and B_k is constructed in the same way as **FUNC**_{N-1}($I_2, \dots, I_N$).

For example, a function, **FUNC**₃, that has three vector arguments K , I , and J , where $K = (k_1 \ k_2 \ \dots \ k_{n_K})'$, $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n_{K n_I},1} & r_{n_{K n_I},2} & \cdots & r_{n_{K n_I},n_J} \end{pmatrix}$$

where $r_{u,v} = \mathbf{FUNC}_3(k_{d_1(u,n_I)}, i_{d_2(u,n_I)}, j_v)$, $u = 1, \dots, n_{K n_I}$, $v = 1, \dots, n_J$, and $d_1(m, n)$ is the integer such that $d_1(m, n)n \geq m > (d_1(m, n) - 1)n$ and $d_2(m, n) = m - (d_1(m, n) - 1)n$. **COV** and **MU** for GM HMM and **SIGMA** for Gaussian HMM are types of **FUNC**₃.

- A function, **FUNC**_N, that has N vector arguments $I_1, I_2, \dots$, and I_N , where $N > 2$ and N is even and where $I_n = (i_{n,1} \ i_{n,2} \ \dots \ i_{n,j_n})'$, $n = 1, \dots, N$, returns a block matrix,

$$R = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,j_2} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,j_2} \\ \vdots & \vdots & \ddots & \vdots \\ B_{j_1,1} & B_{j_1,2} & \cdots & B_{j_1,j_2} \end{pmatrix}$$

where $B_{k,l} = \mathbf{FUNC}_N(i_{1,k}, i_{2,l}, I_3, \dots, I_N)$, $k = 1, \dots, j_1$, $l = 1, \dots, j_2$, and $B_{k,l}$ is constructed in the same way as **FUNC**_{N-2}($I_3, \dots, I_N$).

For example, a function, **FUNC**₄, that has four vector arguments K , L , I , and J , where $K = (k_1 \ k_2 \ \dots \ k_{n_K})'$, $L = (l_1 \ l_2 \ \dots \ l_{n_L})'$, $I = (i_1 \ i_2 \ \dots \ i_{n_I})'$, and $J = (j_1 \ j_2 \ \dots \ j_{n_J})'$, returns a matrix,

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n_L n_J} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n_L n_J} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n_{K n_I},1} & r_{n_{K n_I},2} & \cdots & r_{n_{K n_I},n_L n_J} \end{pmatrix}$$

where $r_{u,v} = \mathbf{FUNC}_4(k_{d_1(u,n_I)}, l_{d_1(v,n_J)}, i_{d_2(u,n_I)}, j_{d_2(v,n_J)})$, $u = 1, \dots, n_{K n_I}$, $v = 1, \dots, n_L n_J$, where $d_1(m, n)$ is the integer such that $d_1(m, n)n \geq m > (d_1(m, n) - 1)n$ and $d_2(m, n) = m - (d_1(m, n) - 1)n$. **AR** and **SIGMA** for GM HMM and **XL** are types of **FUNC**₄.

The functions that refer to parameters can accept empty arguments or omit any number of last arguments. The empty or omitted arguments are replaced by all possible values for those arguments. For example, PROC HMM is used to fit a bivariate three-state Gaussian HMM as follows:

```
model y1 y2 / type=gaussian nstate=3;
```

In order to initialize the second row of **TPM** to be {0.1 0.7 0.2}, you can use the following statement:

```
initial tpm(2,{1 2 3}) = {0.1 0.7 0.2};
```

Taking advantage of empty arguments, you can specify the preceding example as follows:

```
initial tpm(2,) = {0.1 0.7 0.2};
```

To get all coefficients of the covariance matrix for state 2, you can use **SIGMA**(2, {1 2}, {1 2}), or **SIGMA**(2, ,), or **SIGMA**(2). To get all coefficients of exogenous variables on dependent variables, you can use **SIGMA**({1 2 3}, {1 2}, {1 2}), or **SIGMA**(, ,), or **SIGMA**(), or even just **SIGMA**.

Another type of function does not refer to parameters but generates useful matrices. Table 13.5 lists all built-in functions.

Table 13.5 Functions That Do Not Refer to Parameters

Function	Description
DIAG (A)	Creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix
I (n)	Creates an $n \times n$ identity matrix
J (m,n,elem)	Creates an $m \times n$ matrix whose elements are all equal to elem
SHAPE (A,m,n)	Creates an $m \times n$ matrix whose elements are from matrix A

For more information about each function in Table 13.5, see the section “Details of Functions” on page 692.

Details of Operators

This section describes all operators that are available in the HMM procedure. Each subsection describes the operator and shows how it is used.

Addition Operator: +

The addition operator (+) computes a new matrix whose elements are the sums of corresponding elements of the two operands. Following are examples of how the addition operator is used:

matrix1 + matrix2	adds the element in the i th row and j th column of the first matrix to the element in the i th row and j th column of the second matrix, for $i = 1, \dots, n$, $j = 1, \dots, p$, where matrix1 and matrix2 are both $n \times p$ matrices. For example, {1 2 3, 4 5 6} + {7 8 9, 10 11 12} results in {8 10 12, 14 16 18}.
matrix + scalar	adds the scalar value to each element of the matrix . For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + 1.
matrix + vector	adds the vector value to each row or column of the $n \times p$ matrix : • If you add an $n \times 1$ column vector, each row of the vector is added to each row of the matrix. For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + {1, 1}. • If you add a $1 \times p$ row vector, each column of the vector is added to each column of the matrix. For example, you can obtain {2 3 4, 5 6 7} from {1 2 3, 4 5 6} + {1 1 1}.

Comparison Operators: =, <, <=, >, >=

The comparison operators (=, <, <=, >, >=) compare two matrices element by element and return a list of equivalent restrictions on only scalar constants and parameters.

```
matrix1 = matrix2
matrix1 < matrix2
matrix1 <= matrix2
matrix1 > matrix2
matrix1 >= matrix2
```

For example, the INITIAL statement with matrix expressions

```
initial SIGMA(1, {1,2}, {1,2}) = SIGMA(2, {3,4}, {3,4});
```

is transformed into the following equivalent INITIAL statement with scalar parameters:

```
initial SIGMA(1,1,1) = SIGMA(2,3,3),
      SIGMA(1,1,2) = SIGMA(2,3,4),
      SIGMA(1,2,1) = SIGMA(2,4,3),
      SIGMA(1,2,2) = SIGMA(2,4,4);
```

You can also use the comparison operators to conveniently compare all elements of a matrix to a scalar. If either argument is a scalar, then the HMM procedure performs an elementwise comparison between each element of the matrix and the scalar.

You can also compare an $n \times p$ matrix to a row or column vector:

- If the comparison is to an $n \times 1$ column vector, the HMM procedure compares each row of the vector to each row of the matrix.
- If the comparison is to a $1 \times p$ row vector, the HMM procedure compares each column of the vector to each column of the matrix.

For example, the following statements are equivalent:

```
initial SIGMA(1,4:5,1:3) = 0.2;
initial SIGMA(1,4:5,1:3) = {0.2, 0.2};
initial SIGMA(1,4:5,1:3) = {0.2 0.2 0.2};
```

Concatenation Operator, Horizontal: ||

The horizontal concatenation operator (||) produces a new matrix by horizontally joining **matrix1** and **matrix2**.

```
matrix1 || matrix2
```

The matrices must have the same number of rows, which is also the number of rows in the new matrix. The number of columns in the new matrix is the number of columns in **matrix1** plus the number of columns in **matrix2**.

For example, {1 1 1, 7 7 7} || {0 0 0, 8 8 8} returns {1 1 1 0 0 0, 7 7 7 8 8 8}.

Concatenation Operator, Vertical: //

The vertical concatenation operator (//) produces a new matrix by vertically joining **matrix1** and **matrix2**.

matrix1 // matrix2

The matrices must have the same number of columns, which is also the number of columns in the new matrix. The number of rows in the new matrix is the number of rows in **matrix1** plus the number of rows in **matrix2**.

For example, {1 1 1} // {0 0 0, 8 8 8} returns {1 1 1, 0 0 0, 8 8 8}.

Direct Product Operator: @

The direct product operator (@) computes a new matrix that is the direct product (also called the Kronecker product) of **matrix1** and **matrix2**.

matrix1 @ matrix2

For matrices **A** and **B**, the direct product is denoted by $\mathbf{A} \otimes \mathbf{B}$. The number of rows in the new matrix equals the product of the number of rows in **matrix1** and the number of rows in **matrix2**; the number of columns in the new matrix equals the product of the number of columns in **matrix1** and the number of columns in **matrix2**.

Specifically, if **A** is an $n \times p$ matrix and **B** is an $m \times q$ matrix, then the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ is the following $nm \times pq$ block matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}B & \cdots & A_{1p}B \\ \vdots & \ddots & \vdots \\ A_{n1}B & \cdots & A_{np}B \end{bmatrix}$$

For example, {1 2, 3 4} @ {0 2} returns {0 2 0 4, 0 6 0 8}, and {0 2} @ {1 2, 3 4} returns {0 0 2 4, 0 0 6 8}. Note that the direct product of two matrices is not commutative.

Distribution Operator: ~

The distribution operator links the matrix to its distribution function.

matrix ~ distribution-function

This operator is used in the PRIOR statement. For more information, see the section “PRIOR Statement” on page 664.

For example, **ISPV ~ DIR({1,1,1})** means that the initial state probability vector (ISPV) follows a Dirichlet distribution with an argument of a 3×1 vector {1,1,1}.

Index Creation Operator: :

The index creation operator (:) creates a column vector whose first element is **value1**, second element is **value1+1**, and so on, until the last element, which is less than or equal to **value2**.

value1 : value2

For example, 3 : 6 returns {3 4 5 6}.

If **value1** is greater than **value2**, a reverse-order index is created. For example, 6 : 3 returns {6 5 4 3}.

Neither **value1** nor **value2** is required to be an integer.

Multiplication Operator, Elementwise: #

The elementwise multiplication operator (#) computes a new matrix whose elements are the products of the operands. Following are examples of how the elementwise multiplication operator is used:

matrix1 # matrix2	computes a new matrix whose elements are the products of the corresponding elements of matrix1 and matrix2 . For example, {1 2, 3 4} # {4 8, 0 5} returns {4 16, 0 20}.
matrix # scalar	multiplies each element in matrix by the scalar value.
matrix # vector	multiplies each row or column of the $n \times p$ matrix by a corresponding element of the vector: • If you multiply by an $n \times 1$ column vector, each row of the matrix is multiplied by the corresponding row of the vector. • If you multiply by a $1 \times p$ row vector, each column of the matrix is multiplied by the corresponding column of the vector.

For example, a 2×3 matrix can be multiplied on either side by a 2×3 , 1×3 , 2×1 , or 1×1 scalar.

The product of elementwise multiplication is also known as the Schur or Hadamard product. Elementwise multiplication (which uses the # operator) should not be confused with matrix multiplication (which uses the * operator).

Multiplication Operator, Matrix: *

The matrix multiplication operator (*) computes a new matrix by performing matrix multiplication.

matrix1 * matrix2

The first matrix must have the same number of columns as the second matrix has rows. The new matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. That is, if **A** is an $n \times p$ matrix and **B** is a $p \times m$ matrix, then the product **A * B** is an $n \times m$ matrix. The (i, j) element of the product is the sum $\sum_{k=1}^p A_{ik} B_{kj}$.

For example, {1 2, 3 4} * {1, 2} returns {5, 11}.

Sign Reversal Operator: -

The sign reversal operator (−) computes a new matrix whose elements are formed by reversing the sign of each element in **matrix**. The sign reversal operator is also called the unary minus operator.

- matrix

For example, −{−1 7 6, 2 0 −8} returns {1 −7 −6, −2 0 8}.

Subscripts: []

Subscripts are used with matrices to select submatrices, where **rows**, **columns**, and **elements** are expressions that evaluate to scalars or vectors. If these expressions are numeric, they must contain valid subscript values of rows and columns, or the indices, in the argument matrix.

matrix[rows, columns]

matrix[elements]

For example, `{1 2 3, 4 5 6, 7 8 9}[2,3]` returns 6; `{1 2 3, 4 5 6, 7 8 9}[2,1:3]` returns `{4 5 6}`; and `{1 2 3, 4 5 6, 7 8 9}[,3]` returns `{3, 6, 9}`. Because the HMM procedure stores matrices in row-major order, `{11 22 33, 44 55 66, 77 88 99}[{3 5 9}]` returns `{33, 55, 99}`.

Subtraction Operator: `-`

The subtraction operator (`-`) computes a new matrix whose elements are formed by subtracting the corresponding elements of the second operand from the first operand.

matrix1 - matrix2 computes a new matrix whose elements are formed by subtracting the corresponding elements of **matrix2** from those of **matrix1**.

For example, `{1 2 3, 4 5 6} - {1 1 1, 1 1 1}` returns `{0 1 2, 3 4 5}`.

matrix - scalar subtracts the **scalar** value from each element of the **matrix**.

For example, `{1 2 3, 4 5 6} - 1` returns `{0 1 2, 3 4 5}`.

matrix - vector subtracts the **vector** from each element of the **matrix**:

- If you subtract an $n \times 1$ column vector, each row of the vector is subtracted from each row of the matrix. For example, `{1 2 3, 4 5 6} - {1, 1}` returns `{0 1 2, 3 4 5}`.
- If you subtract a $1 \times p$ row vector, each column of the vector is subtracted from each column of the matrix. For example, `{1 2 3, 4 5 6} - {1 1 1}` returns `{0 1 2, 3 4 5}`.

Transpose Operator: ```

The transpose operator, denoted by the backquote (```), exchanges the rows and columns of **matrix**, producing the transpose of **matrix**.

matrix`

If v is the value in the i th row and j th column of **matrix**, then the transpose of **matrix** contains v in the j th row and i th column. If **matrix** contains n rows and p columns, the transpose has p rows and n columns.

For example, `{1 2, 3 4, 5 6}`` returns `{1 3 5, 2 4 6}`.

Details of Functions

DIAG Function

The **DIAG** function creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix.

DIAG(matrix)

The **matrix** argument can be either a square matrix or a vector:

- If **matrix** is a square matrix, the **DIAG** function creates a vector from the diagonal elements of the matrix.

For example, `DIAG({1 2 3, 4 5 6, 7 8 9})` returns `{1, 5, 9}`.

- If **matrix** is a vector, the **DIAG** function creates a matrix whose diagonal elements are the values in the vector. All off-diagonal elements are zeros.

For example, **DIAG**({1 5 9}) or **DIAG**({1, 5, 9}) returns {1 0 0, 0 5 0, 0 0 9}.

You can call the **DIAG** function repeatedly. For example, **DIAG**(**DIAG**({1 2 3, 4 5 6, 7 8 9})) returns {1 0 0, 0 5 0, 0 0 9}.

I Function

The **I** function creates an identity matrix that contains **dim** rows and columns.

I(**dim**)

The diagonal elements of an identity matrix are ones; all other elements are zeros. The value of **dim** must be an integer greater than or equal to 1. Noninteger operands are rounded to the nearest integer.

For example, **I**(3) returns {1 0 0, 0 1 0, 0 0 1}.

J Function

The **J** function creates a matrix that contains **nrow** rows and **ncol** columns, in which all elements are equal to **value**.

J(**nrow**, **ncol**, **value**)

The arguments **nrow** and **ncol** are both integers; **value** can be any expression that returns a linear combination of scalar constants and parameters.

For example, **J**(2, 3, 1) returns {1 1 1, 1 1 1}.

SHAPE Function

The **SHAPE** function creates a new matrix from data in **matrix**.

SHAPE(**matrix**, **nrow**, **ncol**)

The values **nrow** and **ncol** specify the number of rows and columns, respectively, in the new matrix. The **SHAPE** function produces the new matrix by traversing the argument matrix in row-major order until it reaches the specified number of elements. If necessary, the **SHAPE** function reuses elements.

For example, **SHAPE**({1 2 3, 4 5 6}, 3, 2) returns {1 2, 3 4, 5 6}; **SHAPE**({1 2 3, 4 5 6}, 5, 2) returns {1 2, 3 4, 5 6, 1 2, 3 4}; and **SHAPE**({1 2 3, 4 5 6}, 1, 4) returns {1 2 3 4}.

Label Switching Problem

Label switching is an important and severe problem for all kinds of HMMs. Nonidentifiability is caused by invariance to relabeling the states of the hidden Markov chain (or the components of the mixture distribution in the case of GH HMMs); for further discussion and illustration, see Frühwirth-Schnatter (2006, page 15 and 173) and the references therein.

Any two HMMs are mathematically equivalent if the only difference between them is that the states or components are labeled in different ways. For example, consider the following models, which are both univariate bivariate Gaussian HMMs described by the parameters $\{\pi, \mathbf{A}, \mathbf{B}\}$, where π is the initial state probability vector, $\mathbf{A}$ is the transition probability matrix, $\mathbf{B} \equiv \{\mu_1, \Sigma_1, \mu_2, \Sigma_2\}$, and μ_1 and Σ_1 are mean and covariance parameters for state 1, and μ_2 and Σ_2 are mean and covariance parameters for state 2:

$$\text{model 1: } \pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix}, \mu_1 = 1, \Sigma_1 = 1, \mu_2 = -1, \Sigma_2 = 4$$

$$\text{model 2: } \pi = \begin{pmatrix} 0.25 \\ 0.75 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{pmatrix}, \mu_1 = -1, \Sigma_1 = 4, \mu_2 = 1, \Sigma_2 = 1$$

In theory, these two models are equivalent, and just the state labels “1” and “2” are switched. When the labels are switched, it is inconvenient to compare any state-related results, such as filtered or smoothed probabilities or decoded path. Because of the randomness in the optimization, you might even get “different” estimation results every time because of label switching.

You can use the LABELSWITCH= option in the PROC HMM statement to solve the label switching problem. When the INMODEL= option in the SCORE statement is not specified, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default; that is, the states are labeled in the ascending order of the covariance parameter estimates. When the INMODEL= option in the SCORE statement is specified, the LABELSWITCH=(SORT=NONE) option is applied by default; that is, there is no relabeling. You can specify additional suboptions in the LABELSWITCH= option to label the states (and components) in whichever way you prefer.

An Example to Illustrate the LABELSWITCH= Option

The following data are simulated for this example. The data generating process (DGP) follows a bivariate tricomponent GM HMM.

```
%let a11 = 0.98;
%let a22 = 0.98;
%let c1_1 = 0.3;
%let c1_2 = 0.4;
%let c1_3 = 0.3;
%let c2_1 = 0.3;
%let c2_2 = 0.4;
%let c2_3 = 0.3;
%let mu1_1_1 = -1;
%let mu1_2_1 = -6;
%let mu1_3_1 = -11;
%let sigma1_1_11 = 1;
```



```

%let sigma1_2_11 = 2;
%let sigma1_3_11 = 3;
%let mu2_1_1 = 6;
%let mu2_2_1 = 12;
%let mu2_3_1 = 20;
%let sigma2_1_11 = 4;
%let sigma2_2_11 = 5;
%let sigma2_3_11 = 6;
%let seed = 1234;
%let T = 400000;
%let nSections = 200;

data labelSwitchDGP;
  do t = 1 to &T.;
    sec = ceil(t*&nSections./&T.);
    if(t=1) then do;
      * initial probability distribution;
      p = (1-&a22.)/((1-&a11.)+(1-&a22.));
    end;
    else do;
      * transition probability matrix;
      if(lags=1) then p = &a11.;
      else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.);
    if(s=1) then do;
      * choose component;
      u = uniform(&seed.);
      if(u<=&c1_1.) then do;
        * x ~ N(mu, Sigma) at state 1, component 1;
        x = &mu1_1_1. + sqrt(&sigma1_1_11.)*e1;
      end;
      else do;
        if(u<=&c1_1.+&c1_2.) then do;
          * x ~ N(mu, Sigma) at state 1, component 2;
          x = &mu1_2_1. + sqrt(&sigma1_2_11.)*e1;
        end;
        else do;
          * x ~ N(mu, Sigma) at state 1, component 3;
          x = &mu1_3_1. + sqrt(&sigma1_3_11.)*e1;
        end;
      end;
    end;
    else do;
      * choose component;
      u = uniform(&seed.);
      if(u<=&c2_1.) then do;
        * x ~ N(mu, Sigma) at state 2, component 1;
        x = &mu2_1_1. + sqrt(&sigma2_1_11.)*e1;
      end;
      else do;

```

```

        if(u<=&c1_1.+&c1_2.) then do;
            * x ~ N(mu, Sigma) at state 2, component 2;
            x = &mu2_2_1. + sqrt(&sigma2_2_11.)*e1;
        end;
        else do;
            * x ~ N(mu, Sigma) at state 2, component 3;
            x = &mu2_3_1. + sqrt(&sigma2_3_11.)*e1;
        end;
    end;
end;
output;
lags = s;
end;
run;

```

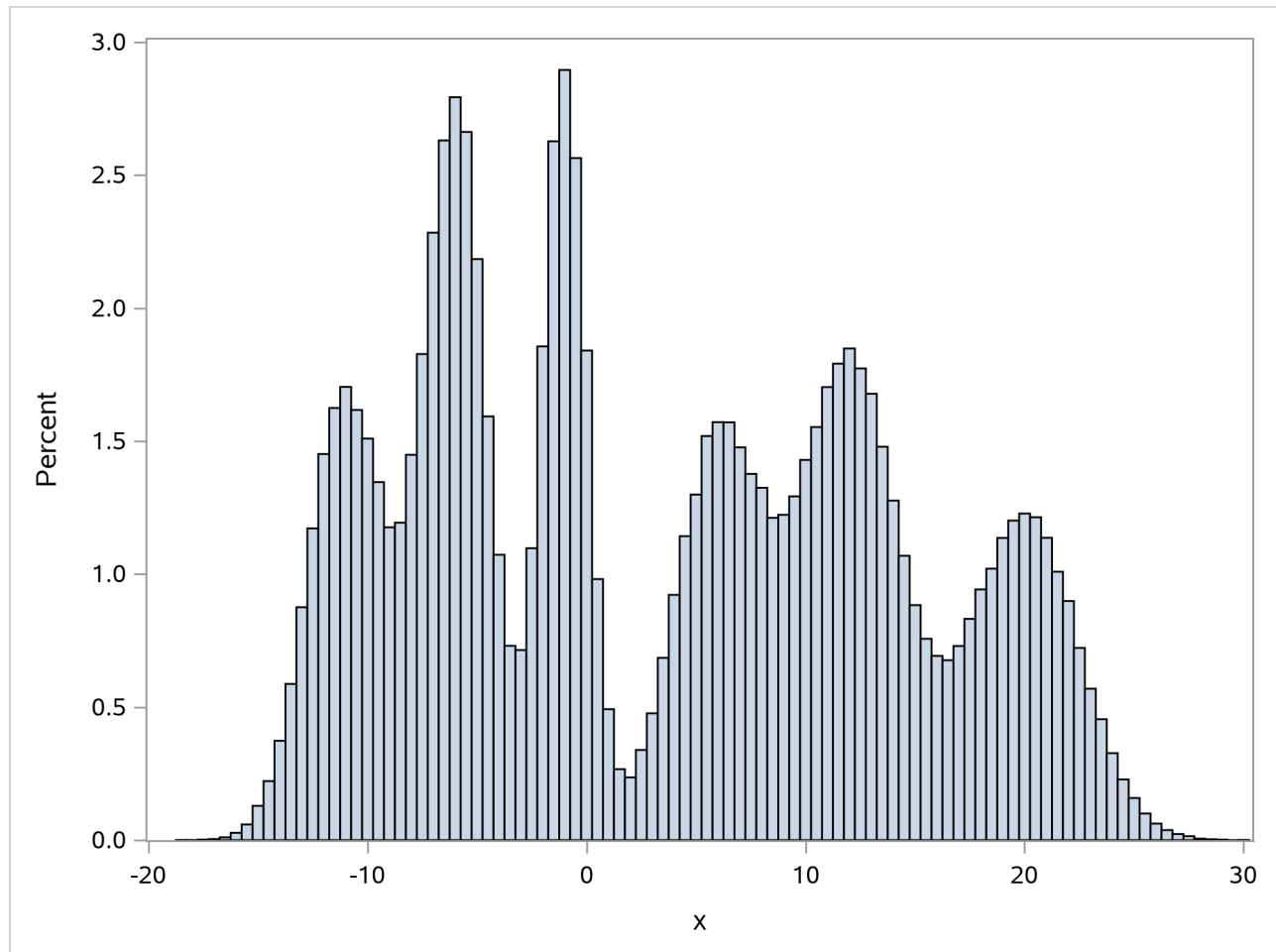
The following statements plot the histogram of the simulated x by ignoring the sectional and sequential information.

```

proc sgplot data=labelSwitchDGP;
    histogram x;
run;

```

Figure 13.51 shows the histogram of x , in which there are six modes around 20, 12, 6, -1 , -6 , and -11 , as expected.

Figure 13.51 Plot of Histogram of x 

The following statements make a copy of the client-side data on the server:

```
data mycas.labelSwitchDGP; set labelSwitchDGP; run;
```

The following statements estimate the six-state Gaussian HMM by using the initial values {20, 12, 6, -1, -6, -11} for the mean parameter:

```
proc hmm data=mycas.labelSwitchDGP;
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;
```

The final parameter estimates after optimization, which are shown in [Figure 13.52](#), are totally different from the reported parameter estimates in [Figure 13.53](#). In [Figure 13.52](#), the six states, described as Gaussian distribution, are in order $N(20, 6)$, $N(12, 5)$, $N(6, 4)$, $N(-1, 1)$, $N(-6, 2)$, and $N(-11, 3)$; however, in [Figure 13.53](#), the six states are in order $N(-1, 1)$, $N(-6, 2)$, $N(-11, 3)$, $N(6, 4)$, $N(12, 5)$, and $N(20, 6)$. The difference is caused by the states being relabeled, by default, according to the estimates of the covariance matrix parameter of each state in ascending order; that is, the LABELSWITCH=(SORT=ASC(SIGMA)) option is applied by default.

Figure 13.52 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	Estimate
TPM1_1	0.292461
TPM1_2	0.392902
TPM1_3	0.294507
TPM1_4	0.005954
TPM1_5	0.007939
TPM1_6	0.006237
TPM2_1	0.294241
TPM2_2	0.387707
TPM2_3	0.298507
TPM2_4	0.005716
TPM2_5	0.007708
TPM2_6	0.006121
TPM3_1	0.295218
TPM3_2	0.387390
TPM3_3	0.297737
TPM3_4	0.006181
TPM3_5	0.007747
TPM3_6	0.005726
TPM4_1	0.006300
TPM4_2	0.008122
TPM4_3	0.006358
TPM4_4	0.292704
TPM4_5	0.390938
TPM4_6	0.295579
TPM5_1	0.005966
TPM5_2	0.007509
TPM5_3	0.006372
TPM5_4	0.294351
TPM5_5	0.390263
TPM5_6	0.295538
TPM6_1	0.005998
TPM6_2	0.008252
TPM6_3	0.005837
TPM6_4	0.292779
TPM6_5	0.392673
TPM6_6	0.294461
MU1_1	19.994009
MU2_1	12.015924
MU3_1	6.036520
MU4_1	-1.002846
MU5_1	-5.994957
MU6_1	-10.977249
SIGMA1_1_1	6.082480
SIGMA2_1_1	4.881128
SIGMA3_1_1	4.037455
SIGMA4_1_1	1.007495
SIGMA5_1_1	1.987657
SIGMA6_1_1	3.042159

Figure 13.53 Parameter Estimates

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.292704	0.001985	147.45	<.0001
TPM1_2	0.390938	0.002676	146.08	<.0001
TPM1_3	0.295579	0.002419	122.18	<.0001
TPM1_4	0.006358	0.000369	17.23	<.0001
TPM1_5	0.008122	0.000429	18.93	<.0001
TPM1_6	0.006300	0.000352	17.91	<.0001
TPM2_1	0.294351	0.001805	163.09	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.295538	0.002251	131.31	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.292779	0.002057	142.35	<.0001
TPM3_2	0.392673	0.002761	142.23	<.0001
TPM3_3	0.294461	0.002483	118.61	<.0001
TPM3_4	0.005837	0.000367	15.91	<.0001
TPM3_5	0.008252	0.000435	18.99	<.0001
TPM3_6	0.005998	0.000350	17.14	<.0001
TPM4_1	0.006181	0.000354	17.46	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.005726	0.000356	16.10	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.005716	0.000305	18.73	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.006121	0.000320	19.13	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.005954	0.000337	17.67	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.006237	0.000354	17.64	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-1.002846	0.005048	-198.67	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-10.977249	0.014627	-750.46	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	1.007495	0.007790	129.34	<.0001
SIGMA2_1_1	1.987657	0.023807	83.49	<.0001
SIGMA3_1_1	3.042159	0.033805	89.99	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082480	0.063053	96.47	<.0001

The following statements explicitly show how the states are relabeled by using the LABELSWITCH=(OUT=) option:

```
proc hmm data=mycas.labelSwitchDGP labelSwitch=(out=mycas.olabel1);
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;

data olabel1; set mycas.olabel1; run;
proc sort data=olabel1; by oldStateLabel; run;
proc print data=olabel1 noobs label; run;
```

The first row in Figure 13.54 shows that the old state label “1” is relabeled as the new state “6”; that is, $N(20, 6)$ was labeled as state 1 but is relabeled as state 6. The same logic applies to other rows. The second row in Figure 13.54 shows that the old state label “2” is relabeled as the new state “5”; the third row shows the old state label “3” is relabeled as the new state “4”; and so on. Note that all parameter estimates (ISPV, TPM, and so on) are permuted accordingly after the states are relabeled.

Figure 13.54 Old and New Labels of States

Old State Label	New State Label
1	6
2	5
3	4
4	1
5	2
6	3

The following statements request that the states be labeled in ascending order of the mean parameter estimates. As shown in Figure 13.55, all parameters are permuted according to the new state labels.

```
proc hmm data=mycas.labelSwitchDGP labelSwitch=(sort=asc(mu) out=mycas.olabel2);
  id time=t section=sec;
  model x / type=gaussian nstate=6;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
run;
```

Figure 13.55 Parameter Estimates

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.294461	0.002483	118.61	<.0001
TPM1_2	0.392673	0.002761	142.23	<.0001
TPM1_3	0.292779	0.002057	142.35	<.0001
TPM1_4	0.005837	0.000367	15.91	<.0001
TPM1_5	0.008252	0.000435	18.99	<.0001
TPM1_6	0.005998	0.000350	17.14	<.0001
TPM2_1	0.295538	0.002251	131.31	<.0001
TPM2_2	0.390263	0.002506	155.74	<.0001
TPM2_3	0.294351	0.001805	163.09	<.0001
TPM2_4	0.006372	0.000331	19.24	<.0001
TPM2_5	0.007509	0.000368	20.40	<.0001
TPM2_6	0.005966	0.000304	19.62	<.0001
TPM3_1	0.295579	0.002419	122.18	<.0001
TPM3_2	0.390938	0.002676	146.08	<.0001
TPM3_3	0.292704	0.001985	147.45	<.0001
TPM3_4	0.006358	0.000369	17.23	<.0001
TPM3_5	0.008122	0.000429	18.93	<.0001
TPM3_6	0.006300	0.000352	17.91	<.0001
TPM4_1	0.005726	0.000356	16.10	<.0001
TPM4_2	0.007747	0.000411	18.83	<.0001
TPM4_3	0.006181	0.000354	17.46	<.0001
TPM4_4	0.297737	0.003360	88.62	<.0001
TPM4_5	0.387390	0.004081	94.94	<.0001
TPM4_6	0.295218	0.002374	124.34	<.0001
TPM5_1	0.006121	0.000320	19.13	<.0001
TPM5_2	0.007708	0.000360	21.42	<.0001
TPM5_3	0.005716	0.000305	18.73	<.0001
TPM5_4	0.298507	0.003211	92.97	<.0001
TPM5_5	0.387707	0.003928	98.70	<.0001
TPM5_6	0.294241	0.002163	136.02	<.0001
TPM6_1	0.006237	0.000354	17.64	<.0001
TPM6_2	0.007939	0.000400	19.87	<.0001
TPM6_3	0.005954	0.000337	17.67	<.0001
TPM6_4	0.294507	0.003291	89.49	<.0001
TPM6_5	0.392902	0.004027	97.58	<.0001
TPM6_6	0.292461	0.002318	126.17	<.0001
MU1_1	-10.977249	0.014627	-750.46	<.0001
MU2_1	-5.994957	0.008460	-708.59	<.0001
MU3_1	-1.002846	0.005048	-198.67	<.0001
MU4_1	6.036520	0.023829	253.33	<.0001
MU5_1	12.015924	0.019460	617.48	<.0001
MU6_1	19.994009	0.018676	1070.59	<.0001
SIGMA1_1_1	3.042159	0.033805	89.99	<.0001
SIGMA2_1_1	1.987657	0.023807	83.49	<.0001
SIGMA3_1_1	1.007495	0.007790	129.34	<.0001
SIGMA4_1_1	4.037455	0.053806	75.04	<.0001
SIGMA5_1_1	4.881128	0.097545	50.04	<.0001
SIGMA6_1_1	6.082480	0.063053	96.47	<.0001

The following statements estimate the bistate tricomponent GM HMM. If you compare the final parameter estimates after optimization that are shown in [Figure 13.56](#) and the parameter estimates that are reported in [Figure 13.57](#), you see that the states and components are relabeled. State labels are exchanged, and the labels of component 1 and 3 for old state 1 are also exchanged, which is clearly shown in [Figure 13.58](#).

```
proc hmm data=mycas.labelSwitchDGP labelSwitch=(out=mycas.olabel3);
  id time=t section=sec;
  model x / type=gaussianMixture nstate=2 ncomponent=3;
  optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint;
  initial mu={20, 12, 6, -1, -6, -11};
  score outmodel=mycas.amodel;
run;

data olabel3; set mycas.olabel3; run;
proc sort data=olabel3; by oldStateLabel oldComponentLabel; run;
proc print data=olabel3 noobs label; run;
```

Figure 13.56 Final Parameter Estimates after Optimization

Optimization Results	
Parameter	Estimate
TPM1_1	0.980249
TPM1_2	0.019751
TPM2_1	0.020196
TPM2_2	0.979804
MCP1_1	0.299927
MCP1_2	0.396951
MCP1_3	0.303122
MCP2_1	0.299444
MCP2_2	0.399150
MCP2_3	0.301406
MU1_1_1	19.994225
MU1_2_1	12.015837
MU1_3_1	6.036274
MU2_1_1	-1.002864
MU2_2_1	-5.994918
MU2_3_1	-10.977185
SIGMA1_1_1_1	6.081829
SIGMA1_2_1_1	4.882496
SIGMA1_3_1_1	4.037829
SIGMA2_1_1_1	1.007494
SIGMA2_2_1_1	1.987574
SIGMA2_3_1_1	3.042284

Figure 13.57 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036274	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994225	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037829	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882496	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081829	0.063045	96.47	<.0001

Figure 13.58 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	3
1	2	2	2
1	3	2	1
2	1	1	1
2	2	1	2
2	3	1	3

The following statements specify the INMODEL= option in the SCORE statement, which causes the LABELSWITCH=(SORT=NONE) option to be applied by default. It is not surprising to see that the parameter estimates in Figure 13.59 are exactly same as those in Figure 13.57; that is, labels of states and components are not reordered, as confirmed by Figure 13.60.

```
proc hmm data=mycas.labelSwitchDGP
    labelSwitch=(out=mycas.olabel4);;
    score inmodel=mycas.omodel;
run;
```

```
data olabel4; set mycas.odelabel4; run;
proc sort data=odelabel4; by oldStateLabel oldComponentLabel; run;
proc print data=odelabel4 noobs label; run;
```

Figure 13.59 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.979804	0.000315	3107.71	<.0001
TPM1_2	0.020196	0.000315	64.06	<.0001
TPM2_1	0.019751	0.000308	64.04	<.0001
TPM2_2	0.980249	0.000308	3178.10	<.0001
MCP1_1	0.299444	0.001126	265.87	<.0001
MCP1_2	0.399150	0.001943	205.44	<.0001
MCP1_3	0.301406	0.001712	176.01	<.0001
MCP2_1	0.303122	0.002797	108.36	<.0001
MCP2_2	0.396951	0.003540	112.13	<.0001
MCP2_3	0.299927	0.001544	194.29	<.0001
MU1_1_1	-1.002864	0.005048	-198.67	<.0001
MU1_2_1	-5.994918	0.008460	-708.63	<.0001
MU1_3_1	-10.977185	0.014627	-750.48	<.0001
MU2_1_1	6.036274	0.023837	253.23	<.0001
MU2_2_1	12.015837	0.019464	617.34	<.0001
MU2_3_1	19.994225	0.018675	1070.63	<.0001
SIGMA1_1_1_1	1.007494	0.007789	129.35	<.0001
SIGMA1_2_1_1	1.987574	0.023804	83.50	<.0001
SIGMA1_3_1_1	3.042284	0.033806	89.99	<.0001
SIGMA2_1_1_1	4.037829	0.053783	75.08	<.0001
SIGMA2_2_1_1	4.882496	0.097590	50.03	<.0001
SIGMA2_3_1_1	6.081829	0.063045	96.47	<.0001

Figure 13.60 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	1	1
1	2	1	2
1	3	1	3
2	1	2	1
2	2	2	2
2	3	2	3

However, you can reorder the labels of states and components in whichever way you prefer by using the LABELSWITCH=(REORDER=) and LABELSWITCH=(REORDERCOMPONENT=) options, as illustrated in the following statements:

```

proc hmm data=mycas.labelSwitchDGP
    labelSwitch=(reorder=(2 1) reorderComponent=(2 1 3 2 3 1)
        out=mycas.olabel5);
    score inmodel=mycas.omodel;
run;

data olabel5; set mycas.olabel5; run;
proc sort data=olabel5; by oldStateLabel oldComponentLabel; run;
proc print data=olabel5 noobs label; run;

```

The parameter estimates are shown in Figure 13.61, and the label comparison is shown in Figure 13.62.

Figure 13.61 Parameter Estimates

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.980249	0.000308	3178.10	<.0001
TPM1_2	0.019751	0.000308	64.04	<.0001
TPM2_1	0.020196	0.000315	64.06	<.0001
TPM2_2	0.979804	0.000315	3107.71	<.0001
MCP1_1	0.299927	0.001544	194.29	<.0001
MCP1_2	0.303122	0.002797	108.36	<.0001
MCP1_3	0.396951	0.003540	112.13	<.0001
MCP2_1	0.399150	0.001943	205.44	<.0001
MCP2_2	0.299444	0.001126	265.87	<.0001
MCP2_3	0.301406	0.001712	176.01	<.0001
MU1_1_1	19.994225	0.018675	1070.63	<.0001
MU1_2_1	6.036274	0.023837	253.23	<.0001
MU1_3_1	12.015837	0.019464	617.34	<.0001
MU2_1_1	-5.994918	0.008460	-708.63	<.0001
MU2_2_1	-1.002864	0.005048	-198.67	<.0001
MU2_3_1	-10.977185	0.014627	-750.48	<.0001
SIGMA1_1_1_1	6.081829	0.063045	96.47	<.0001
SIGMA1_2_1_1	4.037829	0.053783	75.08	<.0001
SIGMA1_3_1_1	4.882496	0.097590	50.03	<.0001
SIGMA2_1_1_1	1.987574	0.023804	83.50	<.0001
SIGMA2_2_1_1	1.007494	0.007789	129.35	<.0001
SIGMA2_3_1_1	3.042284	0.033806	89.99	<.0001

Figure 13.62 Old and New Labels of States and Components

Old State Label	Old Component Label	New State Label	New Component Label
1	1	2	2
1	2	2	1
1	3	2	3
2	1	1	2
2	2	1	3
2	3	1	1

The preceding statements illustrate how to use and interpret the LABELSWITCH= option. The statements use both Gaussian HMMs and GM HMMs, which differ as follows: Although it is trivial to know that a K-state Gaussian HMM can be regarded as a special case of a K-state unicomponent GM HMM, a K-state M-component GM HMM can also be regarded as a special case of a KM-state Gaussian HMM. In most cases, a KM-state Gaussian HMM has many more parameters (because of the huge transition probability matrix) than a K-state M-component GM HMM has. For the preceding statements, all information criteria, although not reported here, prefer the bivariate tricomponent GM HMM (which the true data generating process follows) to the six-state Gaussian HMM.

Analytic Store Technology

Scoring new data is a common and important task in deploying a model. You can score new data by using the SCORE statement in the HMM procedure, or you can do it by taking advantage of the analytic store technology. By using the STORE statement, you can save estimated models in an analytic store table. Then, you can download the table to a local analytic store file. Later, you can load the model and score new data on any platform that supports the analytic store technology—and you do not even need to have the HMM procedure available on those platforms.

The following example illustrates how to use the analytic store technology.

Generate your training data and upload them to the CAS server. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
%let pi1 = 0.5;
%let a11 = 0.2;
%let a22 = 0.2;
%let mu1_1 = 1;
%let mu1_2 = 1;
%let sigma1_11 = 1;
%let sigma1_21 = 0;
%let sigma1_22 = 1;
%let mu2_1 = -1;
%let mu2_2 = -1;
%let sigma2_11 = 1;
%let sigma2_21 = 0;
%let sigma2_22 = 1;

%macro dgpGHMM(table,T,nSections,seed);
  data &table;
    retain cd1_11 cd1_21 cd1_22 cd2_11 cd2_21 cd2_22;
    secLen = ceil(&T./&nSections.);
    do t = 1 to &T.;
      sec = floor((t-1)/secLen)+1;
      if(t=1) then do;
        /* initial probability distribution */
        p = &pi1.;
        /* Cholesky decomposition of sigma1 */
        cd1_11 = sqrt(&sigma1_11.);
        cd1_21 = &sigma1_21./sqrt(&sigma1_11.);
        cd1_22 = sqrt(&sigma1_22.-&sigma1_21.*&sigma1_21./&sigma1_11.);
        /* Cholesky decomposition of sigma2 */
```

```

        cd2_11 = sqrt(&sigma2_11.);
        cd2_21 = &sigma2_21./sqrt(&sigma2_11.);
        cd2_22 = sqrt(&sigma2_22.-&sigma2_21.*&sigma2_21./&sigma2_11.);
    end;
    else do;
        /* transition probability matrix */
        if(lags=1) then p = &a11.;
        else p = 1-&a22.;
    end;
    u = uniform(&seed.);
    if(u<=p) then s=1;
    else s = 2;
    e1 = normal(&seed.); e2 = normal(&seed.);
    if(s=1) then do;
        /* (x,y) ~ N(mu1, Sigma1) at state 1 */
        x = &mu1_1. + cd1_11*e1;
        y = &mu1_2. + cd1_21*e1+cd1_22*e2;
    end;
    else do;
        /* (x,y) ~ N(mu2, Sigma2) at state 2 */
        x = &mu2_1. + cd2_11*e1;
        y = &mu2_2. + cd2_21*e1+cd2_22*e2;
    end;
    output;
    lags = s;
end;
run;
proc sort data=&table.; by sec t; run;
%mend;

* prepare the training data;
%let T = 10000;
%let nSections = 10;
%let seed = 1234;
%dgpGHMM(gHmMTrain,&T.,&nSections.,&seed.);
data mycas.gHmMTrain; set gHmMTrain; run;

```

First, you estimate the model and store it in an analytic store table by using the STORE statement. You must perform this step in the HMM procedure within a CAS session.

```

* estimate the model and save it in an analytic store table;
proc hmm data=mycas.gHmMTrain labelSwitch=(sort=desc(mu));
    id time=t section=sec;
    model x y / type=gaussian nstate=2;
    store out=mycas.ghmmModel;
run;

```

Then, you produce a local analytic store file. You must perform this step in a CAS session that has access to the mycas.ghmmModel data table.

```

* produce a local analytic store file;
proc astore;
    download rstore=mycas.ghmmModel store="ls_ghmm";
run;

```

Now, you can use the local analytic store file to score new data on the platform of interest. For example, you can score new data as in the following code on a SAS client that does not need to connect to a CAS server. In the SETOPTION statements, you set the DECODE option to 1 and the DECODE_WINDOW option to 8 to request the decoding results for an 8-unit time window; you set the EVALUATE option to 1 to request the evaluation up to current observation; that is, the log likelihood; you set the FILTER option to 1 to request the filtered probabilities up to current observation; you set the FORECAST option to 1, the FORECAST_ALPHA to 0.8, the FORECAST_COV to 1, and the FORECAST_LEAD to 4 to request the 4-step-ahead forecasts, their covariance matrices, and their confidence intervals at the significance level 0.80; you set the SMOOTH option to 1 and the SMOOTH_WINDOW option to 12 to request the smoothing results for a 12-unit time window. You use the DESCRIBE statement to output the basic model information and the information about input and output variables. In the SCORE statement, you specify the data to be scored in the DATA= option, the local analytic store file in the STORE= option, and the output table for the scored result in the OUT= option. For more information about the ASTORE procedure, see the chapter “The ASTORE Procedure” in *SAS Visual Data Mining and Machine Learning: Procedures*.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest, &T., &nSections., &seed.);

* score new data;
proc astore;
  setoption decode 1;
  setoption decode_window 8;
  setoption evaluate 1;
  setoption filter 1;
  setoption forecast 1;
  setoption forecast_alpha 0.8;
  setoption forecast_cov 1;
  setoption forecast_lead 4;
  setoption smooth 1;
  setoption smooth_window 12;
  describe store="ls_ghmm";
  score data=gHmmTest store="ls_ghmm" out=ghmmOut;
run;
```

If you need to score new data within a CAS session, you can upload the analytic store file first, and then score the new data by using the RSTORE= option in the ASTORE procedure, as in the following code. Of course, if your CAS session has access to the original analytic store table mycas.ghmmModel, you can skip the uploading step and directly use the option RSTORE=mycas.ghmmModel.

```
* get new data;
%let T = 40;
%let nSections = 2;
%let seed = 12345;
%dgpGHMM(gHmmTest, &T., &nSections., &seed.);
data mycas.gHmmTest; set gHmmTest; run;

* upload the local analytic store to server;
proc astore;
  upload store="ls_ghmm" rstore=mycas.rs_ghmm;
```

```

run;

* score new data;
proc astore;
  setoption decode 1;
  setoption decode_window 8;
  setoption evaluate 1;
  setoption filter 1;
  setoption forecast 1;
  setoption forecast_alpha 0.8;
  setoption forecast_cov 1;
  setoption forecast_lead 4;
  setoption smooth 1;
  setoption smooth_window 12;
  describe rstore=mycas.rs_ghmm;
  score data=mycas.ghmmTest rstore=mycas.rs_ghmm out=mycas.ghmmCasOut;
run;

```

To score new data, you can use the SCORE statement in the HMM procedure, or you can use the STORE statement in the HMM procedure and then use the ASTORE procedure or other products that support the analytic store technology. Sometimes the two methods do not generate the same inference results for the same data set, because these methods treat the data in different ways. That is, using the SCORE statement in the HMM procedure reads all observations in the data set and then scores them; this is the so-called batch mode. In contrast, the analytic store technology deals with observations one by one, which is extremely suitable for streaming analytics; this is the so-called online mode. The following list compares the different kinds of results that these two methods produce:

- For evaluation and filtered results, the two methods are the same.
- For smoothed and decoded results, the two methods are different. Let T be the sample size. For smoothing, in batch mode, $p(S_\tau = i | y_1, \dots, y_T)$, $1 \leq \tau \leq T$, can be calculated; but in online mode, at time t , only $y_1, \dots, y_t$ are available, and then $p(S_\tau = i | y_1, \dots, y_t)$ can be calculated only if $\tau \leq t$. Because of the memory decay of the Markovian structure, $p(S_\tau = i | y_1, \dots, y_t)$ can approximate $p(S_\tau = i | y_1, \dots, y_T)$ very well when $t - \tau$ is large enough. When you set the value of the SMOOTH_WINDOW option to w in the SETOPTION statement in PROC ASTORE, the smoothed probabilities $p(S_\tau = i | y_1, \dots, y_t)$, $\tau = t - w, \dots, t$, $i = 1, \dots, K$, are calculated and output, where K is the number of states. In some literature, the window length is also called the detection delay, because $p(S_{t-w} = i | y_1, \dots, y_t)$ is assumed to be close enough to $p(S_{t-w} = i | y_1, \dots, y_T)$; that is, additional data after time t would not increase the accuracy of the estimation of the smoothed probabilities at time $t - w$. The same logic can be applied to the decoded results.
- For forecasted results, the two methods are the same if there are no exogenous variables in the models; otherwise, the methods are different only if the values of the exogenous variables at time $t + h$, $h > 0$, are needed in some forecast at time t . Let x_t denote the exogenous variables at time t . In batch mode, the h -step-ahead forecast $p(y_{t+h} | y_1, \dots, y_t; x_1, \dots, x_{t+h})$, $h > 0$, can be calculated when $t + h \leq T$ or extra exogenous variable values after T are provided. However, in online mode, x_{t+h} , $h > 0$, is not available at time t , and then some forecasts that need x_{t+h} are not calculable. In practice, in order to use the analytic store technology in online mode, you must make sure that the h -step-ahead forecast of interest is calculable. For example, by using the NOCURRENTX option in the MODEL statement in PROC HMM, and later by setting the FORECAST_LEAD option to 1 in the SETOPTION statement in PROC ASTORE, you can get the one-step-ahead forecast of dependent variables.

Information Criteria

When you compare models, choose the model that has the smallest information criterion values. The information criteria include Akaike's information criterion (AIC), the corrected Akaike's information criterion (AICC), the Bayesian information criterion (BIC, also referred to as the Schwarz Bayesian criterion, SBC), and the Hannan-Quinn criterion (HQC). These criteria are defined as

$$\begin{aligned} \text{AIC} &= -2\ell + 2r \\ \text{AICC} &= -2\ell + 2rT/(T - r - 1) \\ \text{BIC} &= -2\ell + r \log(T) \\ \text{HQC} &= -2\ell + 2r \log(\log(T)) \end{aligned}$$

where ℓ is the log likelihood, r is the total number of parameters in the model, and T is the number of observations that are used to estimate the model.

Missing Values

Exogenous variables cannot have missing values in all types of models. Missing values of dependent variables are treated as follows:

- For Gaussian HMMs and Gaussian mixture HMMs, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on all observations. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized.
- For the regime-switching regression models, all or some of the dependent variables in an observation can be missing. The estimation and inference are based on the observations from the beginning to the last nonmissing or partially nonmissing observation. For example, if the maximum likelihood method is applied, the likelihood of the nonmissing observations and the partially nonmissing observations is calculated and maximized. For the purpose of forecasting, at the end of the data table (or the end of each section if you specify the SECTION= option), the dependent variables can be set as missing values while the exogenous variables are provided.
- For the regime-switching autoregression models, missing values in the dependent variables are not supported. But at the end of the data table (or the end of each section if you specify the SECTION= option), for the purpose of forecasting, the dependent variables can be set as missing values while the exogenous variables are provided.

Data Table Output

The HMM procedure can create the data tables that are specified in the OUT= suboption of the LABELSWITCH= option in the PROC HMM statement; the OUTSTAT= option in the PROC HMM statement; the OUT= options in the DECODE, EVALUATE, FILTER, and SMOOTH statements; the OUT= and OUTALL= options in the ESTIMATE and FORECAST statements; and the OUTMODEL= option in the SCORE statement. The column information for each table is described in the following sections.

OUT= Data Table Generated from the LABELSWITCH= Option in the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= n_1 : n_2 option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= n_1 : n_2 option
OldStateLabel	old label of the state
OldComponentLabel	old label of the component for a GM HMM
NewStateLabel	new label of the state
NewComponentLabel	new label of the component for a GM HMM

OUTSTAT= Data Table Generated from the PROC HMM Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= n_1 : n_2 option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= n_1 : n_2 option
LogLikelihood	log likelihood
AIC	Akaike's information criterion
AICC	corrected Akaike's information criterion
BIC	Bayesian information criterion (also referred to as the Schwarz Bayesian criterion, SBC)
HQC	Hannan-Quinn criterion

OUT= Data Table Generated from the DECODE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State	state value in the most possible path

OUT= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Index	index of the parameter
Type	type of values in the State k variables: EST for estimate and STD for standard error
Parameter	parameter name
State k	value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

OUTALL= Data Table Generated from the ESTIMATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Index	index of the parameter
Parameter	parameter name
Estimate	parameter estimate
StdErr	standard error

COV_ n	value for column n , $n = 1, \dots, N$, in the covariance matrix of parameter estimates, where N is the number of parameters if only one model is estimated or scored; otherwise, N is the maximum of numbers of parameters for all models
----------	---

OUT= Data Table Generated from the EVALUATE Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
LogLikelihood	log likelihood

OUT= Data Table Generated from the FILTER Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State k	filtered probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

OUT= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement

Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h -step-ahead forecast
State k	forecasted probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
{DV p }_Forecast	mean forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q1	first quantile value, $(1 - \alpha)/2$, where α is specified in the ALPHA= α option in the FORECAST statement
{DV p }_Q1	first quantile forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q2	second quantile value, $(1 + \alpha)/2$, where α is specified in the ALPHA= α option in the FORECAST statement
{DV p }_Q2	second quantile forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p
Q3	third quantile value, $1/2$, for median
{DV p }_Q3	median forecast for dependent variable p , $p = 1, \dots, k_y$, where k_y is the number of dependent variables; {DV p } stands for the name of dependent variable p

OUTALL= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
Step	value of h in the h -step-ahead forecast
Index	index of the dependent variable
Variable	dependent variable name
State k	forecasted probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
Forecast	mean forecast for the dependent variable
StdErr	standard error of forecast for the dependent variable
LowerCL	lower confidence limit of forecast for the dependent variable

UpperCL	upper confidence limit of forecast for the dependent variable
Median	median forecast for the dependent variable
COV_{DV p }	value for column p , $p = 1, \dots, k_y$, in the covariance matrix of forecasts of dependent variables, where k_y is the number of dependent variables; {DV p } stands for the name of the dependent variable p

OUTMODEL= Data Table Generated from the SCORE Statement

The output data table consists of binary large object columns to store the binary data, which should be consumed only by the INMODEL= option in the SCORE statement. The number of columns and the definition of columns might be changed without notice in future releases. Do not edit the content of this output data table. If you run the PRINT procedure on this data table, you get a row of zeros or meaningless characters because the data type of each column is binary.

OUT= Data Table Generated from the SMOOTH Statement

The output data table contains the following variables:

ModelIndex	model index, if multiple models are estimated or scored
NState	number of states, if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option
YLAG	order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= $n_1 : n_2$ option
Section ID Variable	values of the variable that is specified in the SECTION= option in the ID statement
Time ID Variable	values of the variable that is specified in the TIME= option in the ID statement
State k	smoothed probability value for state k , $k = 1, \dots, K$, where K is the number of states if you specify the NSTATE= K option; otherwise, $K = \max(n_1, n_2)$, the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE= $n_1 : n_2$ option

ODS Table Names

The HMM procedure assigns a name to each table that it creates. You can use this name to refer to the table when using the Output Delivery System (ODS) to select tables and create output data tables. These names are listed in Table 13.6.

Table 13.6 ODS Tables Produced in the HMM Procedure

ODS Table Name	Description	Option
Algorithm	Option values for the optimization algorithm	PRINTLEVEL= n , where $n = 2$ or 3
ARCoef	Coefficients of autoregressive variables for a regime-switching autoregression model	Default

Table 13.6 *continued*

ODS Table Name	Description	Option
ConstCoef	Coefficients of constants for a regime-switching regression model or regime-switching autoregression model	Default
Cov	Covariance matrices for a regime-switching regression model or regime-switching autoregression model	Default
ExogCoef	Coefficients of exogenous variables for a regime-switching regression model or regime-switching autoregression model	Default
FinalObjectiveFunction	Objective function value at the end of the optimization	PRINTLEVEL=3
FinalParameterEstimates	Final parameter values at the end of the optimization	PRINTLEVEL=3
FitStatistics	Log likelihood and information criteria	Default
InitialObjectiveFunction	Objective function value at the start of the optimization	PRINTLEVEL= <i>n</i> , where <i>n</i> = 1, 2, or 3
InitialParameterEstimates	Initial parameter values at the start of the optimization	PRINTLEVEL= <i>n</i> , where <i>n</i> = 1, 2, or 3
ISPV	Initial state probability vector	Default
IterHistory	Optimization iterations for the active set and interior point algorithms	PRINTITERFREQ= <i>n</i> , <i>n</i> > 0
LinearCoef	Coefficients of linear trends for a regime-switching regression model or regime-switching autoregression model	Default
MCP	Mixture component probabilities for a GM HMM	Default
ModelInfo	Model information	Default
Mu	Mean vectors, μ , for Gaussian or GM HMM	Default
NObs	Observation information	Default
OptIterHistory	Optimization iterations for the SGD algorithm	PRINTITERFREQ= <i>n</i> , <i>n</i> > 0
ParameterEstimates	Model parameter estimates	Default
PriorHyperparm	Prior hyperparameters	METHOD=MAP
QuadCoef	Coefficients of quadratic trends for a regime-switching regression model or regime-switching autoregression model	Default

Table 13.6 *continued*

ODS Table Name	Description	Option
SeasCoef	Coefficients of seasonal dummies for a regime-switching regression model or regime-switching autoregression model	Default
Sigma	Covariance matrices, σ , for a Gaussian or GM HMM	Default
TPM	Transition probability matrix	Default

Several Types of Models as Special Cases of Hidden Markov Models

Several types of models can be treated as special cases of hidden Markov models, including Gaussian mixture models (GMMs), Gaussian mixture regressions (GMRs), regressions, autoregressive error models, vector autoregressions (VARs) with or without exogenous variables in the standard or mean-adjusted form, and so on. In this section, a few of these models are illustrated.

Gaussian Mixture Model and Gaussian Mixture Regressions

Let $\mathbf{Y}_i = (y_{1i}, \dots, y_{pi})'$, $i = 1, \dots, n$, denote a p -dimensional vector of random variables, and $\mathbf{Y}_i$ follows an iid Gaussian mixture distribution,

$$\mathbf{Y}_i \sim \text{GM}(\{w_1, \dots, w_M\}, \{\mu_1, \dots, \mu_M\}, \{\Sigma_1, \dots, \Sigma_M\})$$

where $\text{GM}(\dots)$ represents the Gaussian mixture distribution; M is the number of components; and the w_j , $j = 1, \dots, M$, are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights ($w_j \geq 0$, $j = 1, \dots, M$, and $\sum_{j=1}^M w_j = 1$). The μ_j and Σ_j , $j = 1, \dots, M$, are the mean and covariance parameters, respectively, for the j th Gaussian component. The preceding model is the Gaussian mixture model (GMM). For more information about GMM, see Frühwirth-Schnatter (2006). For more information about Gaussian mixture regression (GMR) and its relationship to GMM, see Sung (2004).

Consider a bivariate Gaussian mixture model that has the following parameter values:

$$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \mu_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -2 \\ -2 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The following statements simulate the bivariate vectors from the previous model to provide training data. In fact, for GMMs, each observation is independent from the others. In order to use the HMM procedure to estimate the GMMs, you must assign each observation in the training data set a unique section ID and a constant time ID; that is, each section contains only one observation.

```

data one;
  call streaminit(12345);
  do sec = 1 to 10000;
    t = 1;
    u = rand('UNIFORM');
    if(u<=0.75) then do;
      s = 1;
      x = 2 + rand('NORMAL');
      y = 2 + rand('NORMAL');
    end;
    else do;
      s = 2;
      x = -2 + rand('NORMAL');
      y = -2 + rand('NORMAL');
    end;
    output;
  end;
run;

```

```
data mycas.one; set one; run;
```

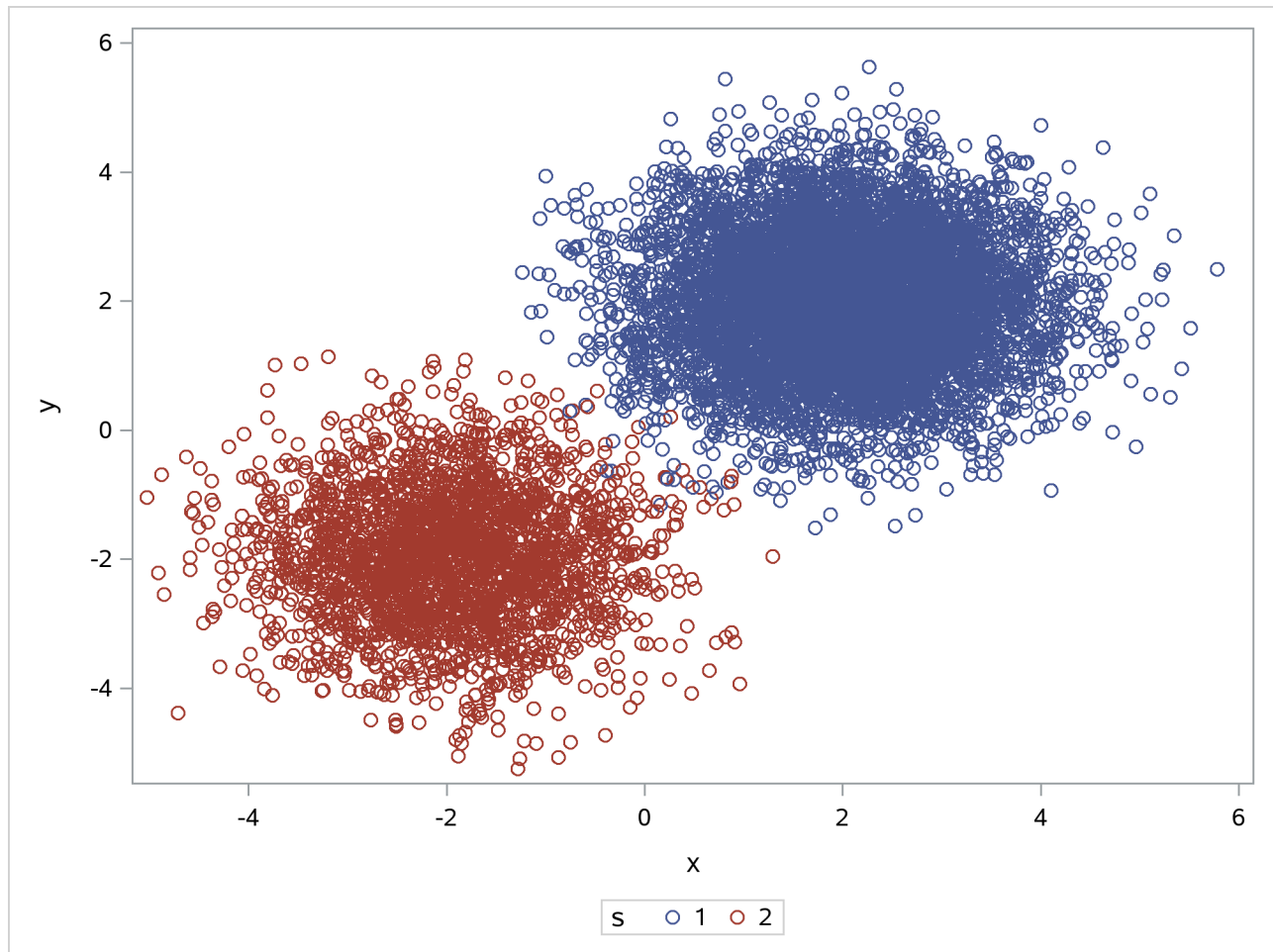
The following statements plot the observed points (x, y) :

```

proc sgplot data=one;
  scatter x=x y=y / group=s;
run;

```

As shown in [Figure 13.63](#), it seems that the points come from two different clusters.

Figure 13.63 Unclassified Observations

The following statements estimate the two-component Gaussian mixture model and then classify the data points through decoding; here decoding means finding the most likely cluster for the observed data. The STORE statement is used to save the model in the analytic store table for future scoring.

```
proc hmm data=mycas.one labelSwitch=(sort=desc(mu));
  id time=t section=sec;
  model x y / nstate=2 type=gaussian estispv;
  decode out=mycas.mydecode;
  store mycas.mygmm;
run;
```

The estimates of all parameters are shown in [Figure 13.64](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value. The TPM parameters are meaningless in this context. The ISPV parameters are the mixture component probabilities. The parameter estimates are very close to the true parameter values in the DGP.

Figure 13.64 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
ISPV1	0.741154	0.004402	168.35	<.0001
ISPV2	0.258846	0.004402	58.80	<.0001
TPM1_1	0.500000	0		
TPM1_2	0.500000	0		
TPM2_1	0.500000	0		
TPM2_2	0.500000	0		
MU1_1	2.017677	0.011520	175.14	<.0001
MU1_2	1.998486	0.011714	170.61	<.0001
MU2_1	-1.972142	0.019170	-102.88	<.0001
MU2_2	-1.971479	0.020118	-98.00	<.0001
SIGMA1_1_1	0.971840	0.016189	60.03	<.0001
SIGMA1_2_1	-0.003386	0.011752	-0.29	0.7732
SIGMA1_2_2	1.002999	0.016772	59.80	<.0001
SIGMA2_1_1	0.918582	0.026555	34.59	<.0001
SIGMA2_2_1	-0.009950	0.019897	-0.50	0.6170
SIGMA2_2_2	1.020699	0.029038	35.15	<.0001

The accuracy of classification through the GMM is calculated by the following statements:

```
data mydecode; set mycas.mydecode; run;
proc sort data=mydecode; by sec; run;

data combine;
  merge mydecode(in=a) one(in=b);
  by sec;
run;

data checkAccuracy;
  set combine;
  retain accuracy 0;
  if(state=s) then accuracy+1;
  if(_N_=10000) then do;
    accuracy = accuracy/_N_;
    output;
  end;
  keep accuracy;
run;

proc print data=checkAccuracy noobs; run;
```

The accuracy of the GMM, which is shown in [Figure 13.65](#), is very high. It implies that you can use the HMM procedure not only to learn but also to infer the GMM.

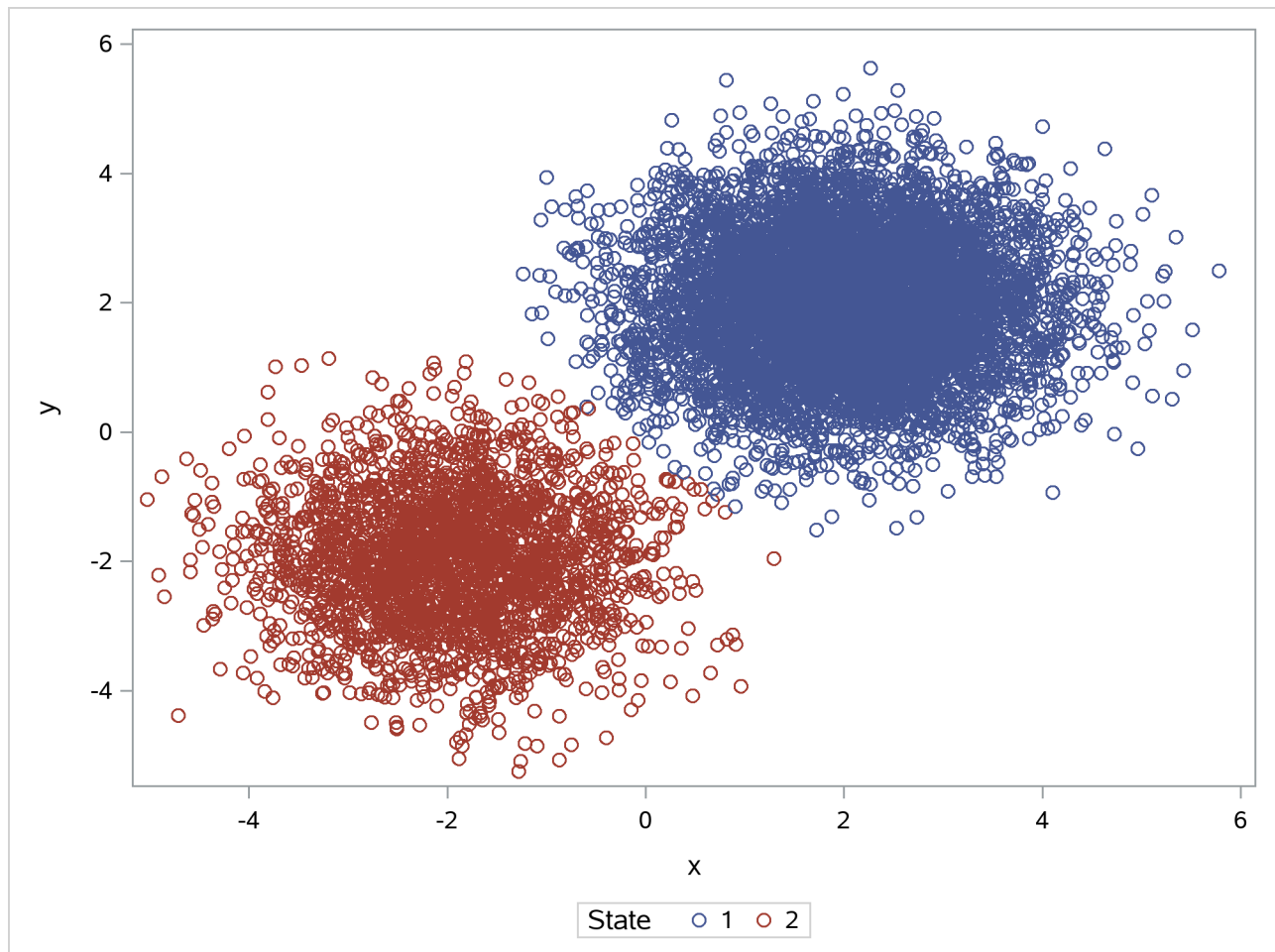
Figure 13.65 Accuracy of GMM on Training Data

accuracy
0.9985

To show how well the data are classified, the following statements plot the classified data points according to the estimated GMM. The plot is shown in [Figure 13.66](#). Comparing it to [Figure 13.63](#), you see that they are almost exactly same.

```
proc sgplot data=combine;
  scatter x=x y=y / group=state;
run;
```

Figure 13.66 Classified Data Points by GMM



It is also easy to apply the estimated GMM to score some new data set by using the analytic store technology. The following statements generate the testing data, apply the ASTORE procedure, and calculate the accuracy of classification. As [Figure 13.67](#) shows, the accuracy is still very high.

```
data two;
  call streaminit(12346);
  do sec = 1 to 10000;
    t = 1;
    u = rand('UNIFORM');
    if(u<=0.75) then do;
      s = 1;
      x = 2 + rand('NORMAL');
    end;
  end;
```

```

        y = 2 + rand('NORMAL');
    end;
    else do;
        s = 2;
        x = -2 + rand('NORMAL');
        y = -2 + rand('NORMAL');
    end;
    output;
end;
run;

proc astore;
    download rstore=mycas.mygmm store='ls_gmm';
    setoption decode 1;
    setoption decode_window 1;
    setoption evaluate 0;
    describe store='ls_gmm';
    score data=two store='ls_gmm' out=gmmout;
run;

data combine2;
    merge gmmout(in=a) two(in=b);
    by sec;
run;

data checkAccuracy2;
    set combine2;
    retain accuracy 0;
    if(DecodedState_BackStep0=s) then accuracy+1;
    if(_N_=10000) then do;
        accuracy = accuracy/_N_;
        output;
    end;
    keep accuracy;
run;

proc print data=checkAccuracy2 noobs; run;

```

Figure 13.67 Accuracy of GMM on Testing Data

accuracy
0.998

Regression and Vector Autoregression

When you specify NSTATE=1 in the MODEL statement for regime-switching regression or autoregression, you are dealing with the regression or vector autoregression (VAR). The autoregressive error model can be treated as a univariate VAR in the mean-adjusted form. For more information about the autoregressive error model, see the chapter “The AUTOREG Procedure” in *SAS/ETS User’s Guide*. For more information about the VARs with or without exogenous variables, see the chapter “The VARMAX Procedure” in *SAS/ETS User’s Guide*.

Examples: HMM Procedure

Example 13.1: Discovering the Hidden Market States by Using the Regime-Switching Autoregression Model

To buy or to sell, that is the question! If you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. However, you cannot directly observe these market states. What you can observe is the stock prices. So, how can you discover the hidden market states from the observable stock price? The question is answered in this example by using regime-switching autoregression models (RS ARs).

Data

Consider the following CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index (Center for Research in Security Prices 2018) from January 19, 1926, to December 29, 2017. You can access the CRSP data through the SASECRSP interface engine; for more information, see the chapter “The SASECRSP Interface Engine” in *SAS/ETS User’s Guide*. You can also directly access the data if you install the CRSP data locally. If you install the CRSP data in the folder XXX, the following statements extract the daily stock prices (that is, market indices) into the data set vwmio:

```
*****;
* DATA;
*****;

title 'Discovering the Hidden Market States';
libname crsplib "XXX";
data vwmio;
    set crsplib.sfz_dind(where=(KYINDNO=1000200
        and caldt>='19JAN1926'd and caldt<='29DEC2017'd));
    format date MMDDYY10.;
    date = caldt; price = aind;
    keep date price;
run;
```

The stock-price series is nonstationary and not suitable to be directly modeled; hence, the stationary weekly-return series is modeled. The following code generates the weekly returns through the daily stock prices and saves them in two data sets. One data set, vwmio, is the full-sample data set, which contains all weekly returns. The other data set, vwmiln, is the in-sample data set, which contains weekly returns before December 31, 2000. The weekly returns after January 1, 2001, are in the out-of-sample period and are used for testing the models.

```
%let ds = vwmio;
%let cutDate = '31DEC2000'd;
data &ds. &ds.In;
    set &ds.O;
    retain cumReturn 0;
    return = (log(price)-log(lag(price)))*100;
    if(return~=. ) then cumReturn + return;
```

```

if(mod(_N_,5)=1 and _N_>1) then do;
    returnw = cumReturn;
    w + 1;
    output &ds.;
    if (date<=&cutDate.) then output &ds.In;
    cumReturn = 0;
end;
keep w date returnw price;
label w="Week Index" date="Date" returnw="Weekly Return"
      price="Market Index";
run;

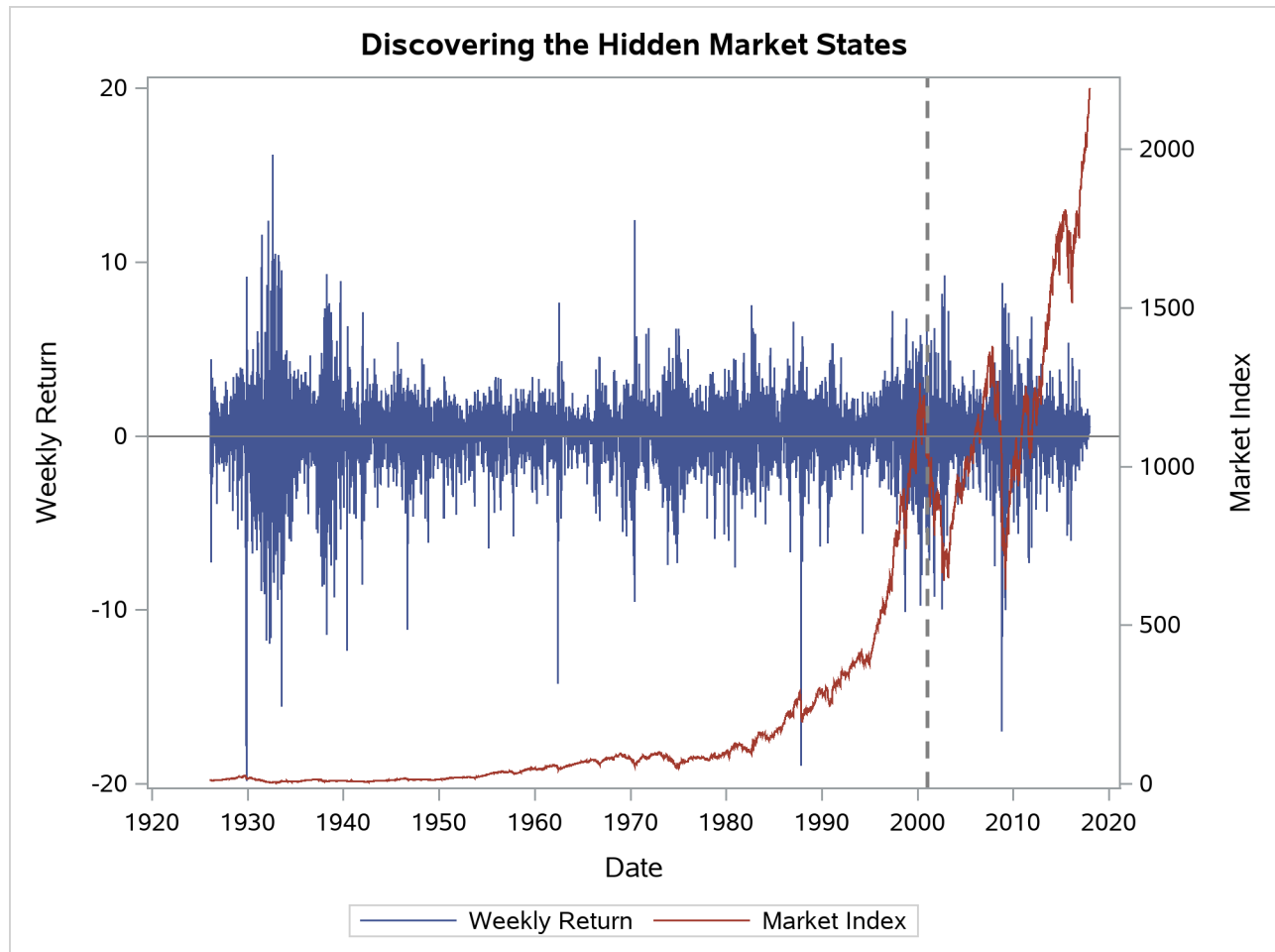
```

The following statements plot the stock prices and weekly returns, as shown in [Output 13.1.1](#). The in-sample and out-of-sample periods are separated by a vertical line.

```

proc sgplot data=&ds.;
    series x=date y=returnw;
    series x=date y=price / y2axis;
    refline 0 / axis=y lineattrs=(color=gray thickness=1);
    refline &cutDate. / axis=x
                        lineattrs=(color=gray thickness=2 pattern=dash);
    yaxis max=20 min=-20;
run;

```

Output 13.1.1 Stock Prices and Weekly Returns

Before you estimate any model, make a copy of the client-side data on the server:

```
data mycas.&ds.In; set &ds.In; run;
data mycas.&ds.; set &ds.; run;
```

Analysis with a Simple Model

Consider a simple case. First, assume that there are two hidden market states, bear market state and bull market state, which can be modeled by exactly two corresponding hidden states in an HMM; that is, consider a bistate HMM. Second, assume that the lagged weekly returns cannot help predict current weekly return at each hidden state; that is, the autoregressive order is zero at each hidden state. Hence, as a start, consider a bistate RS-AR(0) model. The following statements estimate the bistate RS-AR(0) model on weekly returns. The input data table, specified in the required DATA= option in the PROC HMM statement, contains the in-sample data. In order to estimate the model, the ID and MODEL statements must be specified. Since the data are sorted by the variable date, the variable date is assigned to the required TIME= option. However, since the data can also be regarded as being sorted by the variable w, the variable w, instead of the variable date, can be assigned to the TIME= option; if you do so, there is no impact on the model estimation. In the MODEL statement, the dependent variable, the weekly return returnw, is specified first; the TYPE= option is set to AR for the RS-AR model; the NSTATE= option is set to 2 for the bistate model; and the YLAG= option

is set to 0 for an AR(0) model. The SMOOTH, DECODE, and EVALUATE statements specify where to output, respectively, the smoothed probabilities, decoded state path, and log likelihood up to each time point.

```
*****;
* ANALYSIS WITH A SIMPLE MODEL
*****;

proc hmm data=mycas.&ds.In;
  id time=date;
  model returnw / type=ar nstate=2 ylag=0;
  smooth out=mycas.&ds.inSmooth_k2To2_p0To0;
  decode out=mycas.&ds.inDecode_k2To2_p0To0;
  evaluate out=mycas.&ds.inEval_k2To2_p0To0;
run;
```

The number of observations and model information are shown in [Output 13.1.2](#).

Output 13.1.2 Number of Observations and Model Information

Discovering the Hidden Market States

Observation Information	
Number of Observations	3999
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The estimates of the initial state probability vector (ISPV) are shown in [Output 13.1.3](#). In this example, the hidden Markov chain is assumed to be stationary. Hence, the steady-state probability distribution (STPD, which is also called the equilibrium distribution or stationary distribution) of the Markov chain is equal to the ISPV.

Output 13.1.3 Estimates of Initial State Probability Vector

Discovering the Hidden Market States

Initial State Probabilities	
State	Estimation
1	0.76747
2	0.23253

The estimates of the transition probability matrix (TPM) are shown in [Output 13.1.4](#).

Output 13.1.4 Estimates of Transition Probability Matrix

Transition Probabilities		
State	1	2
1	0.98111	0.01889
2	0.06235	0.93765

The estimates of the intercept for each state are shown in [Output 13.1.5](#).

Output 13.1.5 Estimate of Intercept for Each State

Constant	
State	returnw
1	0.28710
2	-0.47243

The estimates of the variance for each state are shown in [Output 13.1.6](#).

Output 13.1.6 Estimate of Variance for Each State

Innovations Covariances		
State	Variable	returnw
1	returnw	2.29936
2	returnw	17.02648

The estimates of all parameters are shown in [Output 13.1.7](#), which displays columns for the parameter name, estimate value, standard error, t value, and p -value.

Output 13.1.7 Parameter Estimates

Parameter Estimates				
Parameter	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.981109	0.003507	279.78	<.0001
TPM1_2	0.018891	0.003507	5.39	<.0001
TPM2_1	0.062352	0.011274	5.53	<.0001
TPM2_2	0.937648	0.011274	83.17	<.0001
CONST1_1	0.287100	0.030130	9.53	<.0001
CONST2_1	-0.472426	0.142740	-3.31	0.0009
COV1_1_1	2.299364	0.090090	25.52	<.0001
COV2_1_1	17.026478	1.052833	16.17	<.0001

The log likelihood and information criteria are shown in [Output 13.1.8](#).

Output 13.1.8 Log Likelihood and Information Criteria

Fit Statistics	
Log Likelihood	-8497.46701
AIC	17006.934019
AICC	17006.955061
BIC	17044.696817
HQC	17020.320118

The smoothed probabilities of states for each observation are output in the data table `mycas.vwmilnSmooth_k2to2_p0to0`. This example discusses one type of inference. The mean and variance (or standard error) of each state can be calculated through the weighted sample, where the smoothed probabilities are the weights. The following statements calculate and print the sample mean and standard error of each state, as shown in [Output 13.1.9](#). In the case of an RS-AR(0) model that has no regressors other than the intercept (which is equivalent to the Gaussian HMM), the sample mean and standard error of each state are almost the same as, respectively, the intercept estimate and the square root of variance estimate of each state.

```

*print sample mean and standard error for each state
  according to the weighted sample,
  where the smoothed probabilities are the weight;
data &ds.inSmooth_k2to2_p0to0;
  set mycas.&ds.inSmooth_k2to2_p0to0;
run;
proc sort data=&ds.inSmooth_k2to2_p0to0; by date; run;
data &ds.inWeighted;
  merge &ds.inSmooth_k2to2_p0to0(in=a) &ds.in(in=b);
  by date;
  if(a=b);
run;
%macro printMeanAndStdErrOfAState(k);
  data &ds.inMeanStdErr;
    set &ds.inWeighted end=eof;
    array cWeights{&k.} _TEMPORARY_;
    array cMeans{&k.} mean1-mean&k.;
    array cVars{&k.} sd1-sd&k.;
    array weights{&k.} state1-state&k.;
    retain cWeights cMeans cVars;
    if(_N_=1) then do;
      do i = 1 to &k.;
        cWeights[i] = weights[i];
        cMeans[i] = weights[i]*returnw;
        cVars[i] = weights[i]*returnw*returnw;
      end;
    end;
  else do;
    do i = 1 to &k.;
      cWeights[i] = cWeights[i] + weights[i];
      cMeans[i] = cMeans[i] + weights[i]*returnw;
      cVars[i] = cVars[i] + weights[i]*returnw*returnw;
    end;
  end;
end;
if eof then do;

```

```

do i = 1 to &k.;
  if(cWeights[i]>0) then do;
    cMeans[i] = cMeans[i] / cWeights[i];
    cVars[i] = cVars[i] / cWeights[i] - cMeans[i]*cMeans[i];
    if(cVars[i]>=0) then cVars[i] = sqrt(cVars[i]);
  end;
  else do;
    cMeans[i] = .;
    cVars[i] = .;
  end;
end;
%do i = 1 %to &k.;
  call symputx("mu&i.",cMeans[&i.], 'G');
  call symputx("sigma&i.",cVars[&i.], 'G');
  state = &i.; mean = mean&i.; stdErr = sd&i.;
  output;
%end;
end;
label state="State" mean="Sample Mean"
      stdErr="Sample Standard Error";
keep state mean stdErr;
run;
proc print data=&ds.inMeanStdErr noobs label; run;
%mend;
%printMeanAndStdErrOfAState(2);

```

Output 13.1.9 Sample Mean and Standard Error of Each State**Discovering the Hidden Market States**

State	Sample	
	Sample Mean	Sample Standard Error
1	0.28710	1.51637
2	-0.47243	4.12632

The following statements plot each state as a Gaussian kernel, which shows how each state looks. As shown in [Output 13.1.10](#), it is obvious that state 1 should represent the bull market state (positive expected return and low risk) and state 2 the bear market state (negative expected return and high risk).

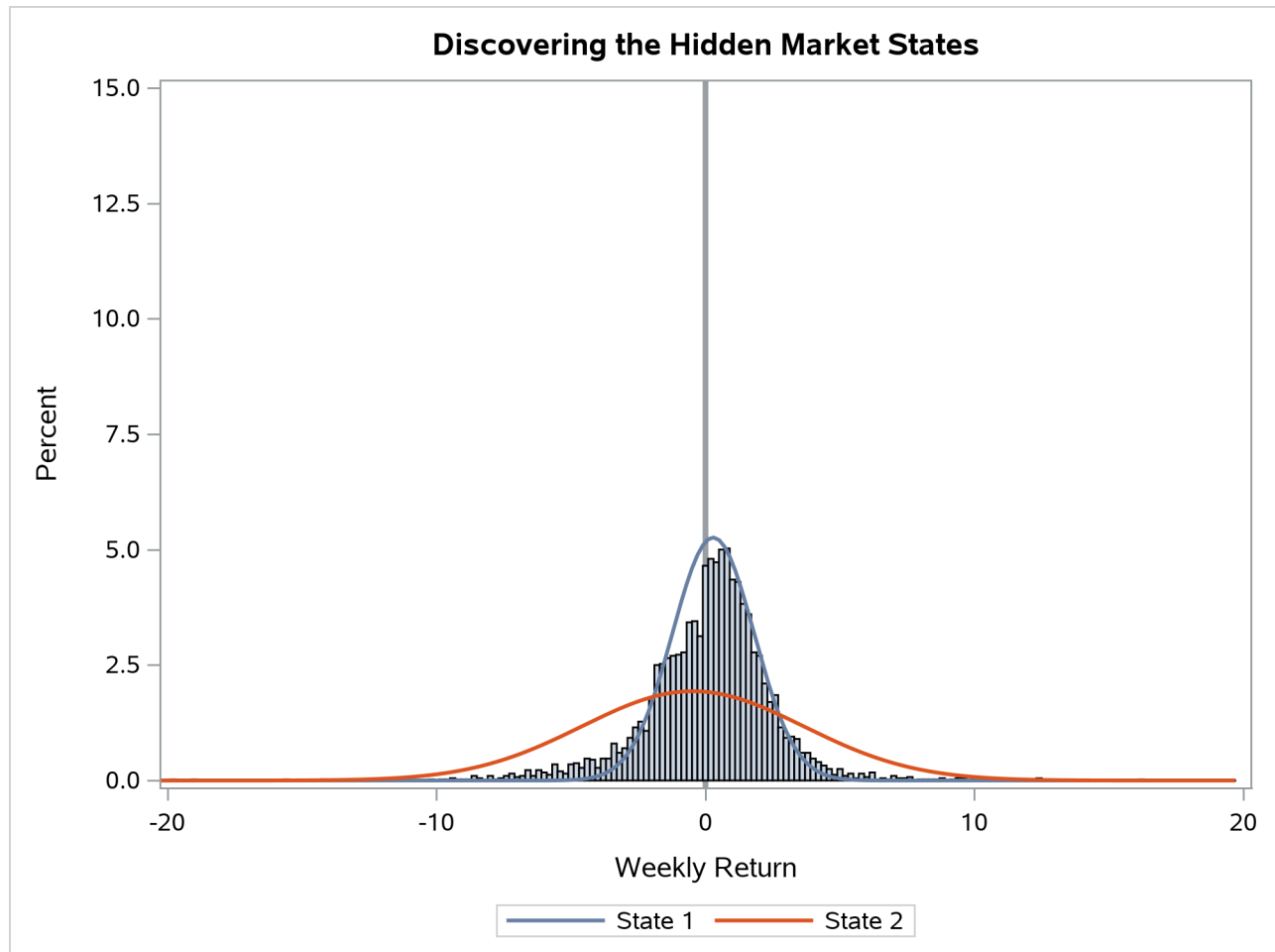
```

%macro plotState(myData,myColumn,ks,ke);
  proc sgplot data=&myData.;
    refline 0 / axis=x lineattrs=(thickness=3);
    histogram &myColumn. / nbins=200;
    %do i = &ks. %to &ke.;
      density &myColumn. / type=normal(mu=&&mu&i. sigma=&&sigma&i.)
        name="state&i."
        legendlabel="State &i.";
    %end;
    keylegend %do i = &ks. %to &ke.; "state&i." %end;;
    xaxis min=-20 max=20 ;
    yaxis min=0 max=15;
  run;
%mend;

```

```
%plotState(&ds.In, returnw, 1, 2);
```

Output 13.1.10 Gaussian Distribution of Each State



The decoded path of states, which provide a historical view of what happened, given all the available data, is output in the data table `mycas.vwmInDecode_k2To2_p0To0`. The decoded path assigns each time point a market state. It seems that the hidden market states have become observable. The following statements plot the decoded market states, as shown in [Output 13.1.11](#):

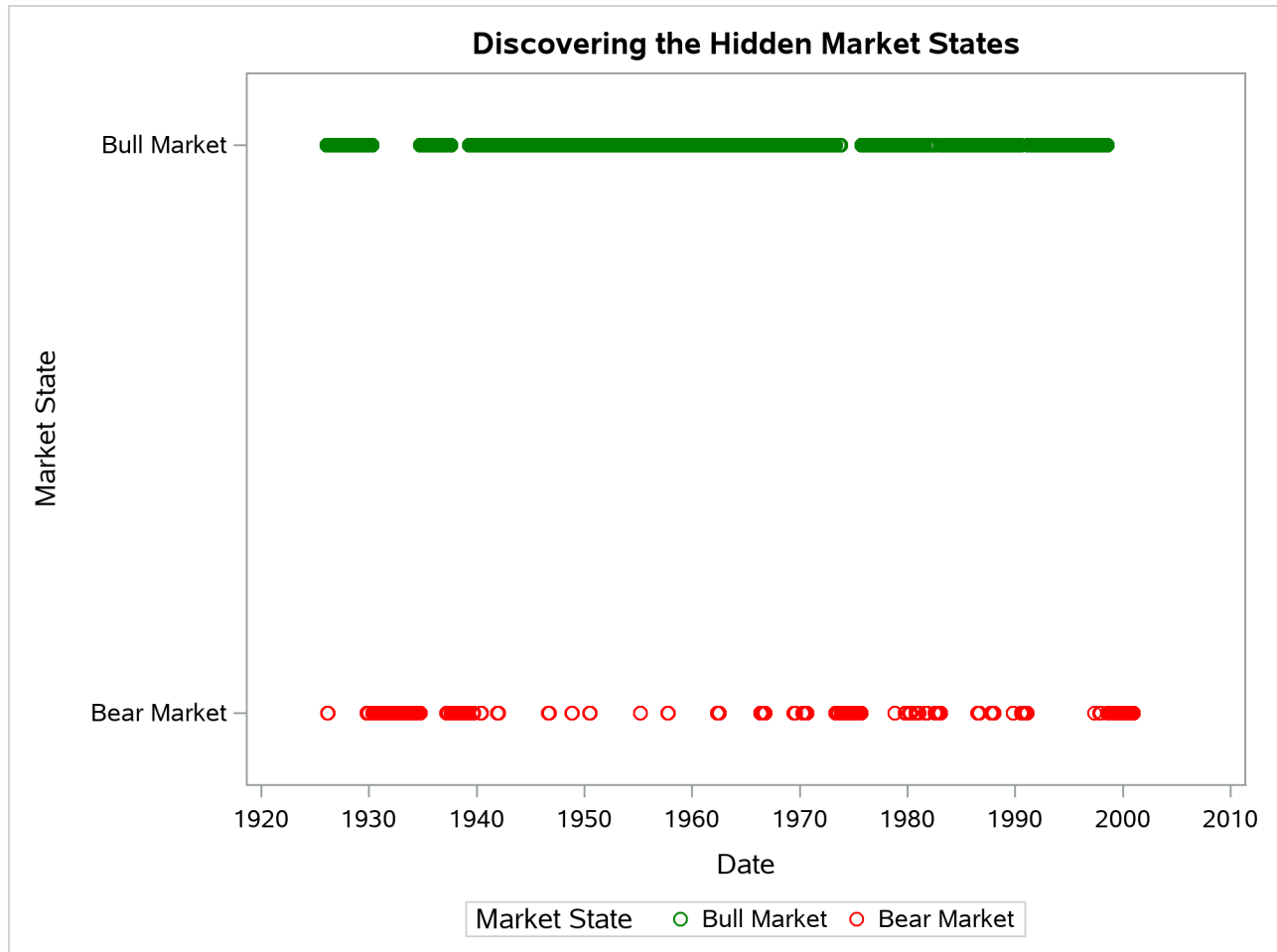
```
* plot decoded results;
data &ds.InDecode_k2to2_p0to0; set mycas.&ds.InDecode_k2to2_p0to0; run;
proc sort data=&ds.InDecode_k2to2_p0to0; by date; run;
data decodeMaketStates;
  set &ds.In;
  set &ds.InDecode_k2to2_p0to0;
  array returnws returnw1-returnw2;
  array prices price1-price2;
  if(state~=.) then do;
    if(state=1) then do;
      returnws[1] = returnw; prices[1] = price;
      marketState="Bull Market";
    end;
    if(state=2) then do;
```

```

    returnws[2] = returnw; prices[2] = price;
    marketState="Bear Market";
  end;
end;
label marketState="Market State";
run;
proc sgplot data=decodeMaketStates;
  styleattrs datacontrastcolors=(green red);
  scatter x=date y=marketState / group=marketState;
  yaxis label="Market State" offsetmin=0.1 offsetmax=0.1 reverse;
run;

```

Output 13.1.11 Decoded Market States



You can incorporate the decoded market states with other variables of interest. The following statements plot the decoded weekly returns, as shown in [Output 13.1.12](#):

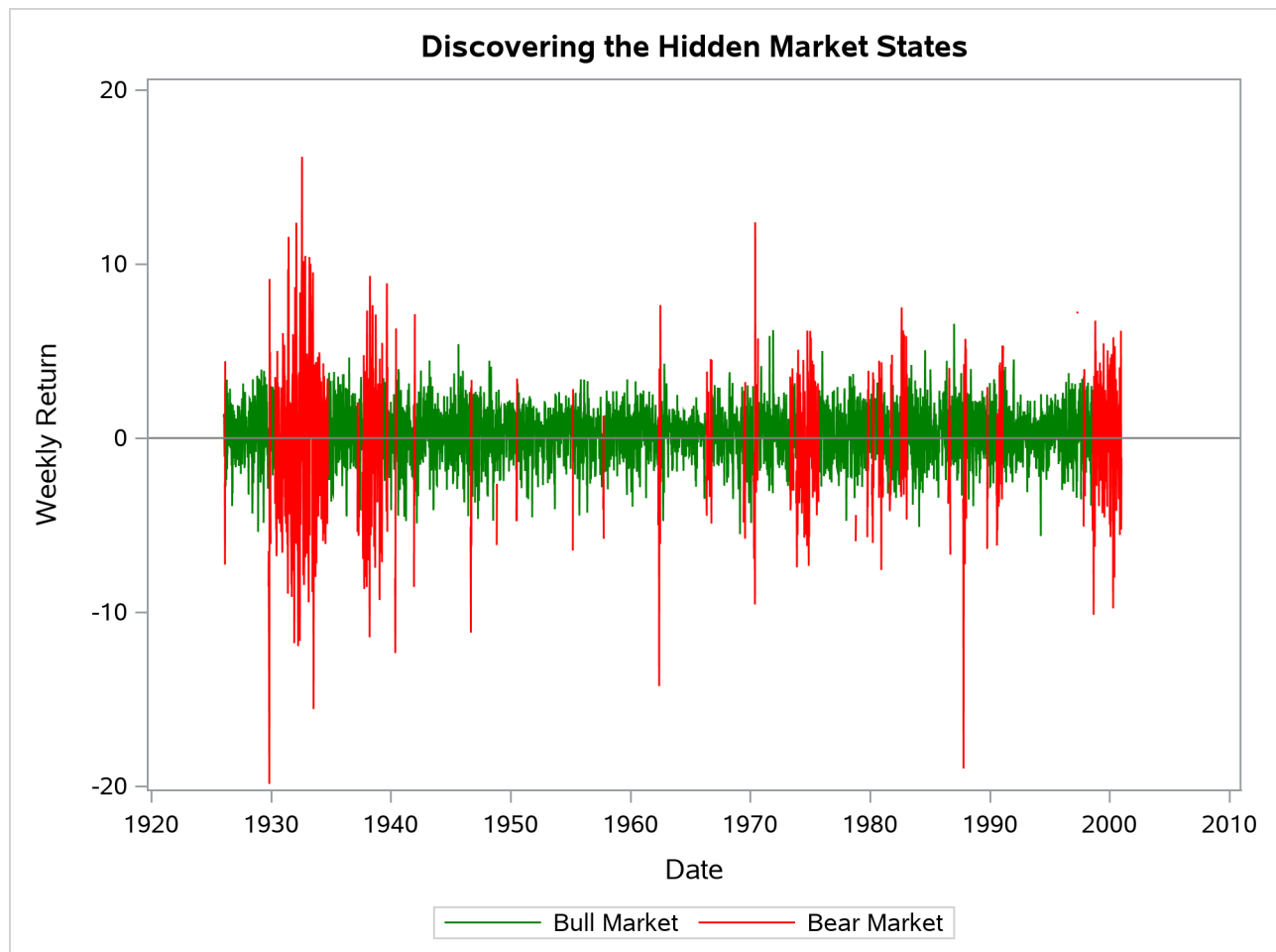
```

proc sgplot data=decodeMaketStates;
  series x=date y=returnw1 / break lineattrs=(color=green)
    legendlabel="Bull Market";
  series x=date y=returnw2 / break lineattrs=(color=red)
    legendlabel="Bear Market";
  refline 0 / axis=y lineattrs=(color=gray thickness=1);

```

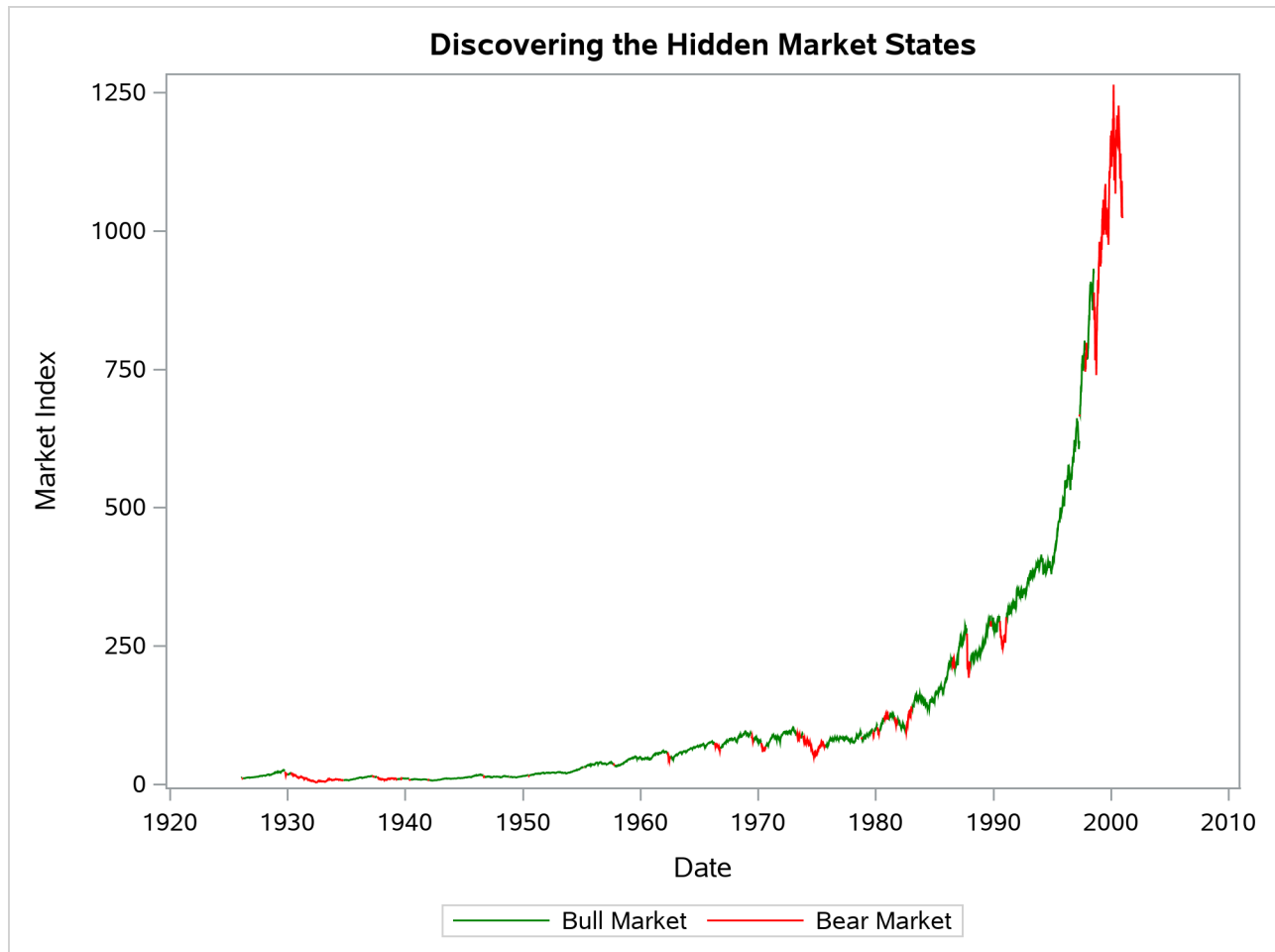
```
yaxis max=20 min=-20 label="Weekly Return";
run;
```

Output 13.1.12 Decoded Weekly Returns



The following statements plot the decoded stock prices, as shown in [Output 13.1.13](#):

```
proc sgplot data=decodeMaketStates;
  series x=date y=price1 / break lineattrs=(color=green)
        legendlabel="Bull Market";
  series x=date y=price2 / break lineattrs=(color=red)
        legendlabel="Bear Market";
  yaxis label="Market Index";
run;
```

Output 13.1.13 Decoded Stock Prices

The following statements plot the log likelihood of each observation conditional on its past; the plot is shown in [Output 13.1.14](#).

```
* plot conditional log likelihood;
%macro plotEvaluation(inds, outds, qConditional);
  data &outds.;
    set &inds.;
    format date MMDDYY10.;
run;
%if (&qConditional.=1) %then %do;
  proc sort data=&outds.;
    by date;
  run;
  data &outds.;
    set &outds.;
    if(_N_=1) then
      condLogLikelihood = logLikelihood;
    else
      condLogLikelihood = logLikelihood - lag(logLikelihood);
  run;
```

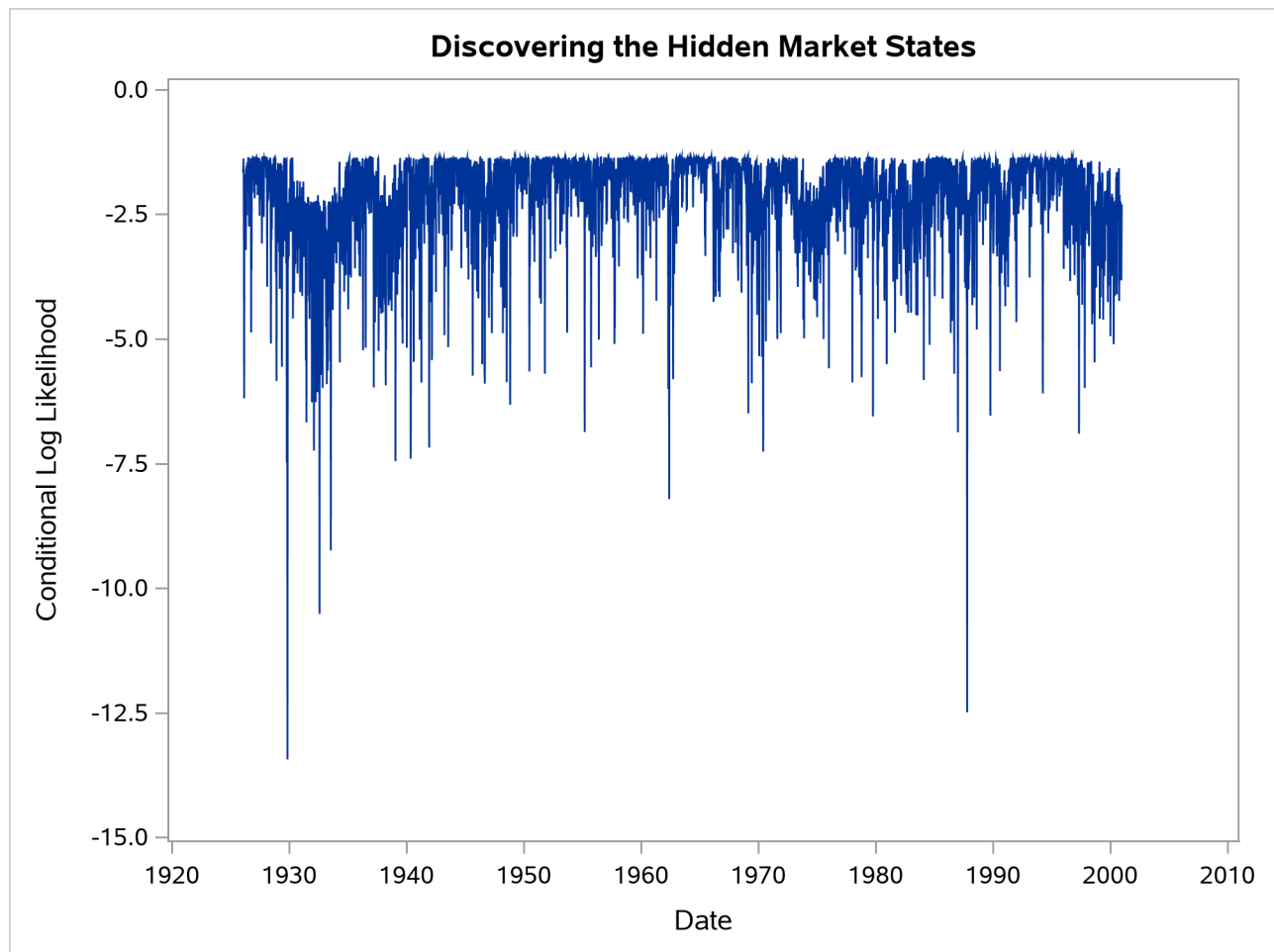
```

proc sgplot data=&outds.;
  series x=date y=condLogLikelihood;
  yaxis label='Conditional Log Likelihood' min=-15 max=0;
run;
%end;
%else %do;
  proc sgplot data=&outds.;
    series x=date y=logLikelihood;
    yaxis label='Log Likelihood';
  run;
%end;
%mend plotEvaluation;

%plotEvaluation(mycas.&ds.inEval_k2to2_p0to0, &ds.inEval_k2to2_p0to0, 1);

```

Output 13.1.14 Conditional Log Likelihood



As shown in [Output 13.1.14](#), some observations are explained well by the model, but there are still many exceptional observations whose conditional log likelihoods are much smaller than others. This means that there might be model misspecification, which can be addressed by solving the model selection problem.

Model Selection

To model the weekly returns, how many states should be considered? How many AR lags should a state include? These are common questions in the model selection process. In this example, the best model is selected using Akaike's information criterion (AIC): the smaller the AIC, the better the model.

The following macro estimates the model or models according to the arguments:

```
*****;
* MODEL SELECTION;
*****;

%macro estimateRSAR(myds, inEstDs, kStart, kEnd, pStart, pEnd, method,
                    maxiter, qMultiStart);
  proc hmm data=&myds. labelSwitch=(sort=none)
    outstat=&myds.&method.Stat_k&kStart.To&kEnd._p&pStart.To&pEnd.;
    id time=date;
    model returnw / type=ar nstate=&kStart.:&kEnd. ylag=&pStart.:&pEnd.
      method=&method.;
    optimize printLevel=3 printIterFreq=1 algorithm=interiorpoint
      maxiter=&maxiter. Multistart=&qMultiStart.;
    score outmodel=&myds.&method.Model_k&kStart.To&kEnd._p&pStart.To&pEnd.;
    learn out=&myds.&method.Learn_k&kStart.To&kEnd._p&pStart.To&pEnd.
      %if %length(&inEstDs.)>0 %then %do; in=&inEstDs. %end;
    ;
    display / excludeall;
  run;
%mend;
```

First, the following statements estimate two- to nine-state RS-AR(0) models. The MAP method is applied to bound and smooth the objective functions. The parameter estimates are saved for further use as the initial values for new estimations. For the best results, multistart mode is strongly recommended, although it might be time consuming.

```
* (1) MAP + MULTISTART for AR(0);
* be aware that the following macro might take tens of hours to finish;
* uncomment it to run;
* %estimateRSAR(myds=mycas.&ds.In, inEstDs=,
  kStart=2, kEnd=9, pStart=0, pEnd=0,
  method=MAP, maxiter=128, qMultiStart=1);
```

Then, the following statements estimate two- to nine-state RS-AR(0) to AR(5) models. The ML method is used because the AIC is based on the log likelihood. The parameter estimates of RS AR(0) models from previous MAP method estimations are used to ensure that the estimates in the ML method are reasonable. The k -state RS AR(p) models, $k = 2, \dots, 9$, $p = 1, \dots, 5$, use the corresponding k -state RS AR($p - 1$) models' parameter estimates as initial values. This is the default implementation in the HMM procedure. The multistart mode is not needed any longer because the initial values are good.

```
* (2) ML + initial values estimated from MAP's AR(0) for AR(0)
  and initial values estimated from AR(p-1) for AR(p), p>0;
* be aware that the following macro might take tens of minutes to finish;
* uncomment it to run;
* %estimateRSAR(myds=mycas.&ds.In, inEstDs=mycas.&ds.InMAPLearn_k2to9_p0to0,
  kStart=2, kEnd=9, pStart=0, pEnd=5,
```

```
method=ML, maxiter=128, qMultiStart=0);
```

Estimating all 48 of the two- to nine-state RS AR(0) to AR(5) models produces AIC values. The following statements print the AIC values, which are shown in [Output 13.1.15](#):

```
* print fit statistics in matrix form;
data &ds.InMLStat_k2to9_p0to5; set mycas.&ds.InMLStat_k2to9_p0to5; run;
proc sort data=&ds.InMLStat_k2to9_p0to5; by modelIndex; run;
%macro printFitStat(fitStat);
  data &ds.In&fitStat.;
    set &ds.InMLStat_k2to9_p0to5 end=eof;
    array &fitStat.s{8,6} _TEMPORARY_;
    array yLags{6} yLag0-yLag5;
    &fitStat.s[nState-1,yLag+1] = &fitStat.;
    if eof then do;
      do i = 1 to 8;
        nState = i+1;
        do j = 1 to 6; yLags(j) = &fitStat.s[i,j]; end;
        output;
      end;
    end;
    keep nState yLag0-yLag5;
run;
proc print data=&ds.In&fitStat. label noobs
  style(header)={textalign=center};
  var nState yLag0-yLag5;
  label nState='k' yLag0='p = 0' yLag1='p = 1' yLag2='p = 2'
        yLag3='p = 3' yLag4='p = 4' yLag5='p = 5';
run;
%mend;
%printFitStat(AIC);

* Uncomment the following macros if you want to see other fit statistics;
* %printFitStat(logLikelihood);
* %printFitStat(AICC);
* %printFitStat(BIC);
* %printFitStat(HQC);
```

[Output 13.1.15](#) indicates that the smallest AIC corresponds to the eight-state RS AR(4) model.

Output 13.1.15 AICs for 48 RS AR Models

Discovering the Hidden Market States

k	p = 0	p = 1	p = 2	p = 3	p = 4	p = 5
2	16987.55	16979.59	16981.64	16985.00	16988.80	16992.07
3	16820.74	16811.38	16816.45	16821.02	16825.88	16826.56
4	16772.85	16764.64	16754.79	16757.74	16763.77	16767.28
5	16728.05	16733.38	16738.14	16743.40	16750.67	16753.11
6	16697.18	16684.36	16684.90	16689.57	16695.59	16700.58
7	16701.64	16693.28	16682.39	16685.51	16690.77	16695.31
8	16685.88	16688.43	16687.35	16668.80	16650.11	16659.51
9	16711.81	16699.75	16698.66	16707.04	16679.98	16684.87

The Selected Model and In-Sample Inference

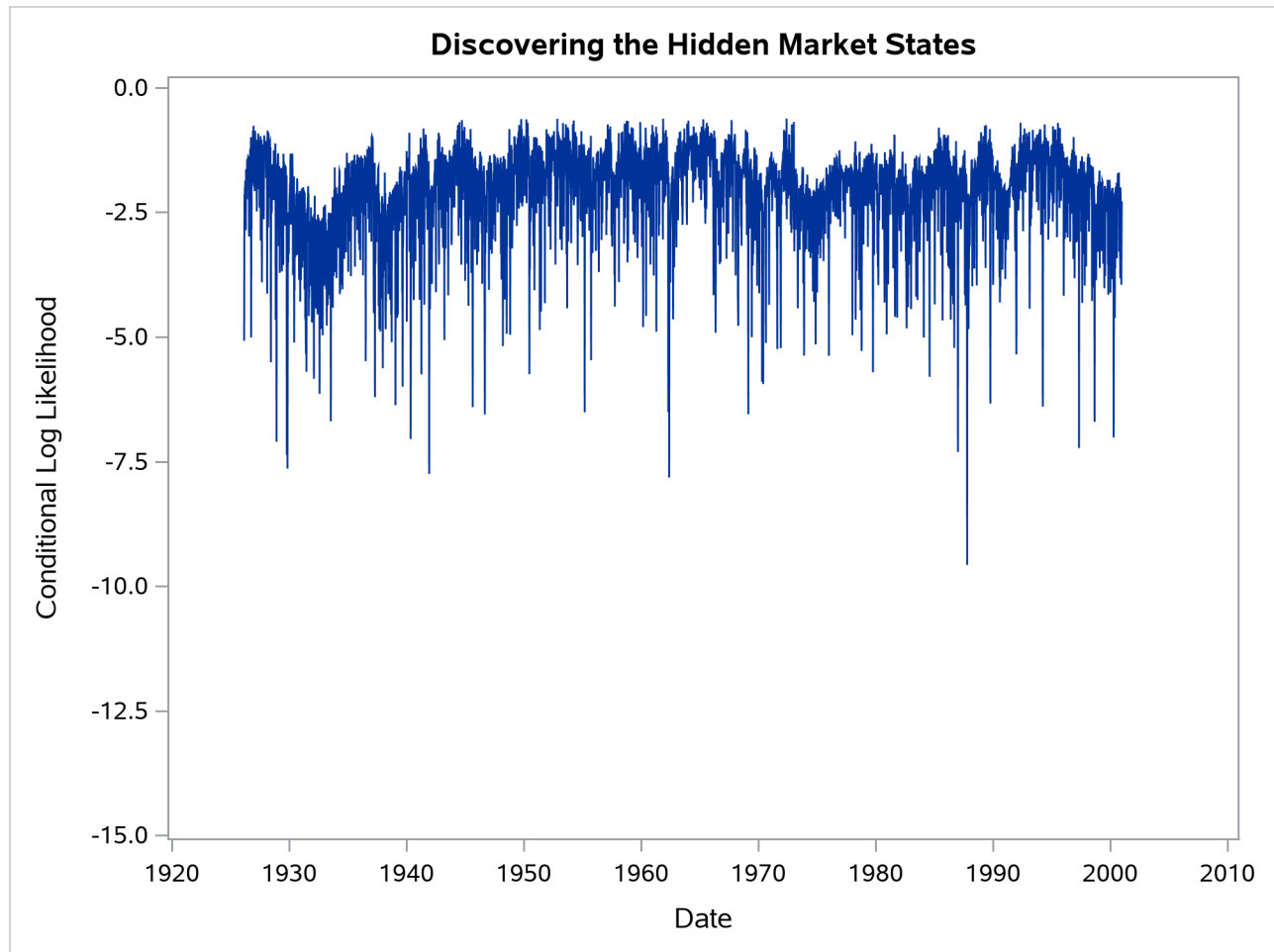
The following statements estimate the AIC-selected eight-state RS AR(4) model:

```
*****;
* THE SELECTED MODEL AND IN-SAMPLE INFERENCE ;
*****;

* select eight-state RS-AR(4) model according to AIC or AICC;
* adjust data to be the same sample size for AR(5);
proc hmm data=mycas.&ds.in(where=(date>='30JAN1926'd));
  id time=date;
  model returnw / type=ar nstate=8 ylag=4;
  optimize printLevel=3 algorithm=interiorpoint maxiter=0;
  score outmodel=mycas.&ds.inMLModel_k8To8_p4To4;
  learn out=mycas.&ds.inMLLearn_k8To8_p4To4
        in=mycas.&ds.inMLLearn_k2to9_p0to5;
  smooth out=mycas.&ds.inSmooth_k8to8_p4to4;
  decode out=mycas.&ds.inDecode_k8to8_p4to4;
  evaluate out=mycas.&ds.inEval_k8to8_p4to4;
  display tpm;
run;
```

The following statement plots the conditional log likelihoods. Compared to the bivariate RS-AR(0) model (as shown in [Output 13.1.14](#)), the eight-state RS-AR(0) model (as shown in [Output 13.1.16](#)) has much higher conditional log likelihoods, which means a much better in-sample goodness of fit.

```
%plotEvaluation(mycas.&ds.inEval_k8to8_p4to4, &ds.inEval_k8to8_p4to4, 1);
```

Output 13.1.16 Conditional Log Likelihoods

The selected model has 112 parameters. 48 of which are related to observation distributions and are often called observation parameters. The following statements print the observation parameters in [Output 13.1.17](#). The analysis of each AR(4) process for each state is beyond this example.

```
* print observation parameters;
data &ds.InObsParm;
  set mycas.&ds.InMLLearn_k8To8_p4To4(where=(type='EST' and index>=17));
  if(index=17) then parameter='Constant';
  if(index=19) then parameter='AR(1)';
  if(index=21) then parameter='AR(2)';
  if(index=23) then parameter='AR(3)';
  if(index=25) then parameter='AR(4)';
  if(index=27) then parameter='Variance';
run;
proc sort data=&ds.InObsParm; by index; run;
proc print data=&ds.InObsParm noobs label;
  var parameter state1-state8;
  label state1='State 1' state2='State 2' state3='State 3'
        state4='State 4' state5='State 5' state6='State 6'
        state7='State 7' state8='State 8';
run;
```

Output 13.1.17 Observation Parameters**Discovering the Hidden Market States**

Parameter	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8
Constant	0.93305	1.15326	-0.59159	1.68887	-0.82124	-2.68701	-0.01897	-0.5056
AR(1)	-0.23663	0.03568	-0.06096	-0.17268	-0.04365	-0.34339	-0.05648	0.1213
AR(2)	0.00568	-0.07571	0.33570	-0.08520	-0.09594	-0.11391	0.06549	0.0259
AR(3)	-0.06493	-0.07301	0.01605	-0.06019	0.04540	-0.60512	0.11289	0.0025
AR(4)	-0.02869	-0.08348	-0.01595	-0.02281	0.12060	-0.29044	-0.11970	0.0613
Variance	0.23055	0.77141	1.21104	1.44655	1.54215	3.64774	9.42764	31.8773

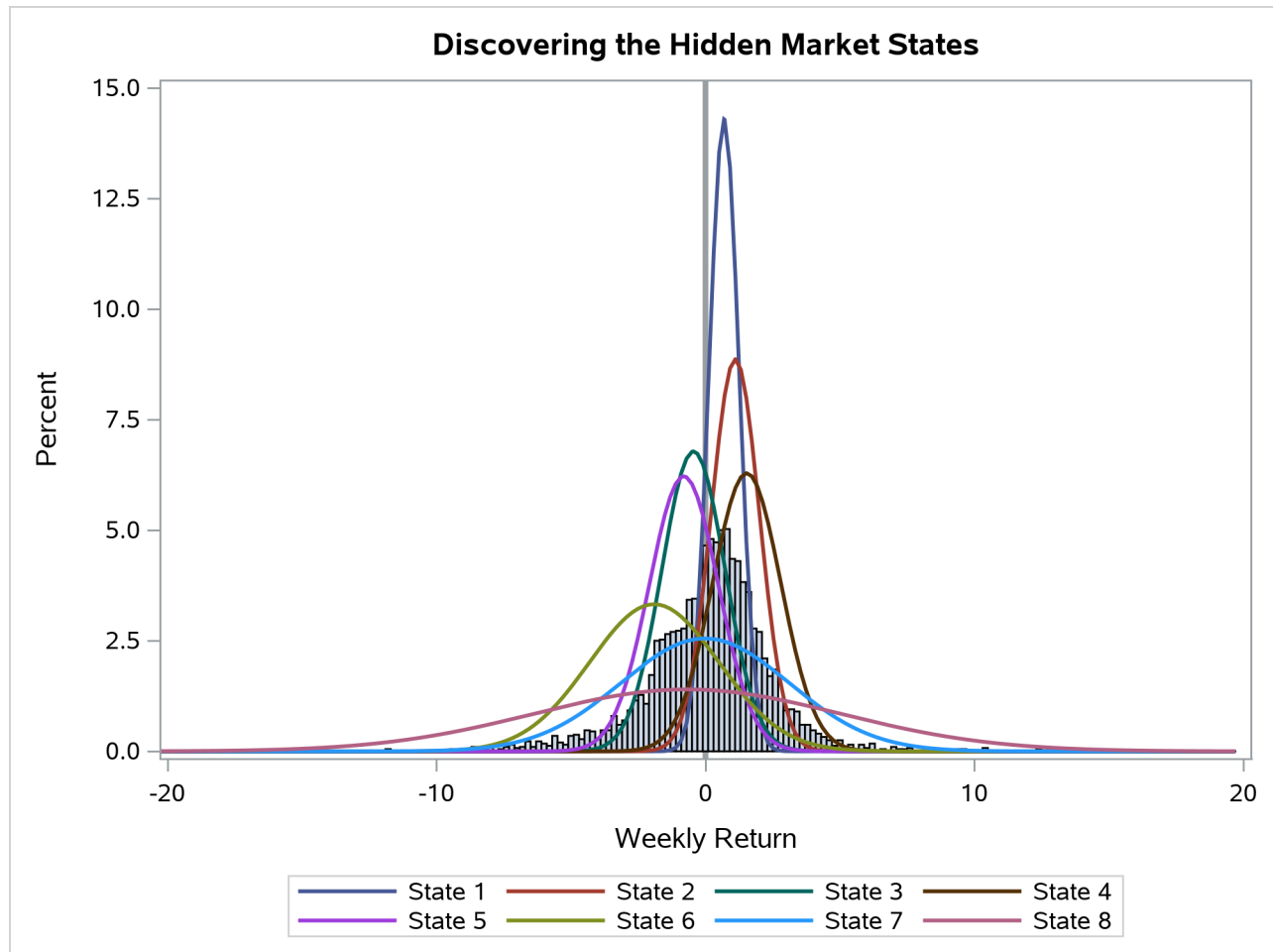
The following statements calculate and print the sample mean and standard error for each state in [Output 13.1.18](#) and then plot the corresponding Gaussian kernels in [Output 13.1.19](#):

```
data &ds.inSmooth_k8to8_p4to4;
    set mycas.&ds.inSmooth_k8to8_p4to4(where=(date>='25FEB1926'd));
run;
proc sort data=&ds.inSmooth_k8to8_p4to4; by date; run;
data &ds.inTruncated; set &ds.in(where=(date>='25FEB1926'd)); run;
data &ds.inWeighted;
    merge &ds.inSmooth_k8to8_p4to4(in=a) &ds.inTruncated(in=b);
    by date;
    if(a=b);
run;
%printMeanAndStdErrOfAState(8);

%plotState(&ds.In,returnw,1,8);
```

Output 13.1.18 Sample Mean and Standard Error of Each State**Discovering the Hidden Market States**

State	Sample Mean	Sample Standard Error
1	0.68995	0.55836
2	1.10751	0.90016
3	-0.44647	1.17678
4	1.53207	1.26888
5	-0.80688	1.28357
6	-1.91024	2.39908
7	-0.01947	3.12820
8	-0.67999	5.70094

Output 13.1.19 Gaussian Kernels for Eight States

As shown in [Output 13.1.19](#), states 1, 2, and 4 seem like a bull market; they have positive expected returns with low risk. The remaining states seem like a bear market; they have negative expected returns or very high risk (or both). However, the market states and their corresponding states are not that simple. The transition probability matrix of the eight-state AR(4) model is shown in [Output 13.1.20](#).

Output 13.1.20 Estimates of Transition Probability Matrix

Discovering the Hidden Market States

State	Transition Probabilities							
	1	2	3	4	5	6	7	8
1	0.00000	0.00000	0.98736	0.00000	0.00000	0.01264	0.00000	0.00000
2	0.57934	0.36501	0.00000	0.00000	0.02789	0.02776	0.00000	0.00000
3	0.00000	0.42100	0.57900	0.00000	0.00000	0.00000	0.00000	0.00000
4	0.00000	0.01976	0.00000	0.61521	0.36503	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.33913	0.55466	0.10621	0.00000	0.00000
6	0.05948	0.00653	0.00000	0.17954	0.05488	0.59808	0.03352	0.06797
7	0.00000	0.00000	0.00000	0.02594	0.00000	0.00000	0.97186	0.00220
8	0.00000	0.00000	0.00000	0.03679	0.00000	0.00000	0.02078	0.94243

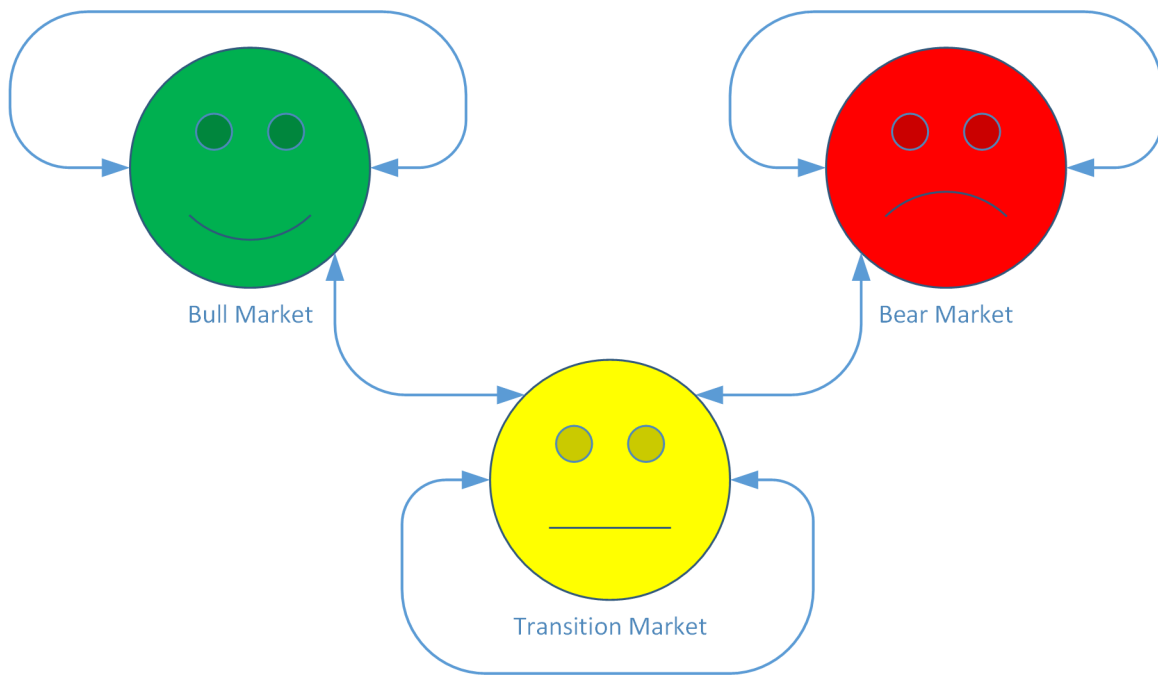
The eight states seem to be categorized into three groups:

- States 1, 2, and 3 belong to first group. Although there are very upward-moving days (state 2) and somewhat downward-moving days (state 3), the main trend is upward (state 1). The risk in this group is pretty low.
- States 4, 5, and 6 belong to second group. There are good days (state 4), bad days (state 5), and even very bad days (state 6). This group has medium risk.
- States 7 and 8 belong to the third group, where the risk is extremely high.

The transition probability matrix tells the more detailed story:

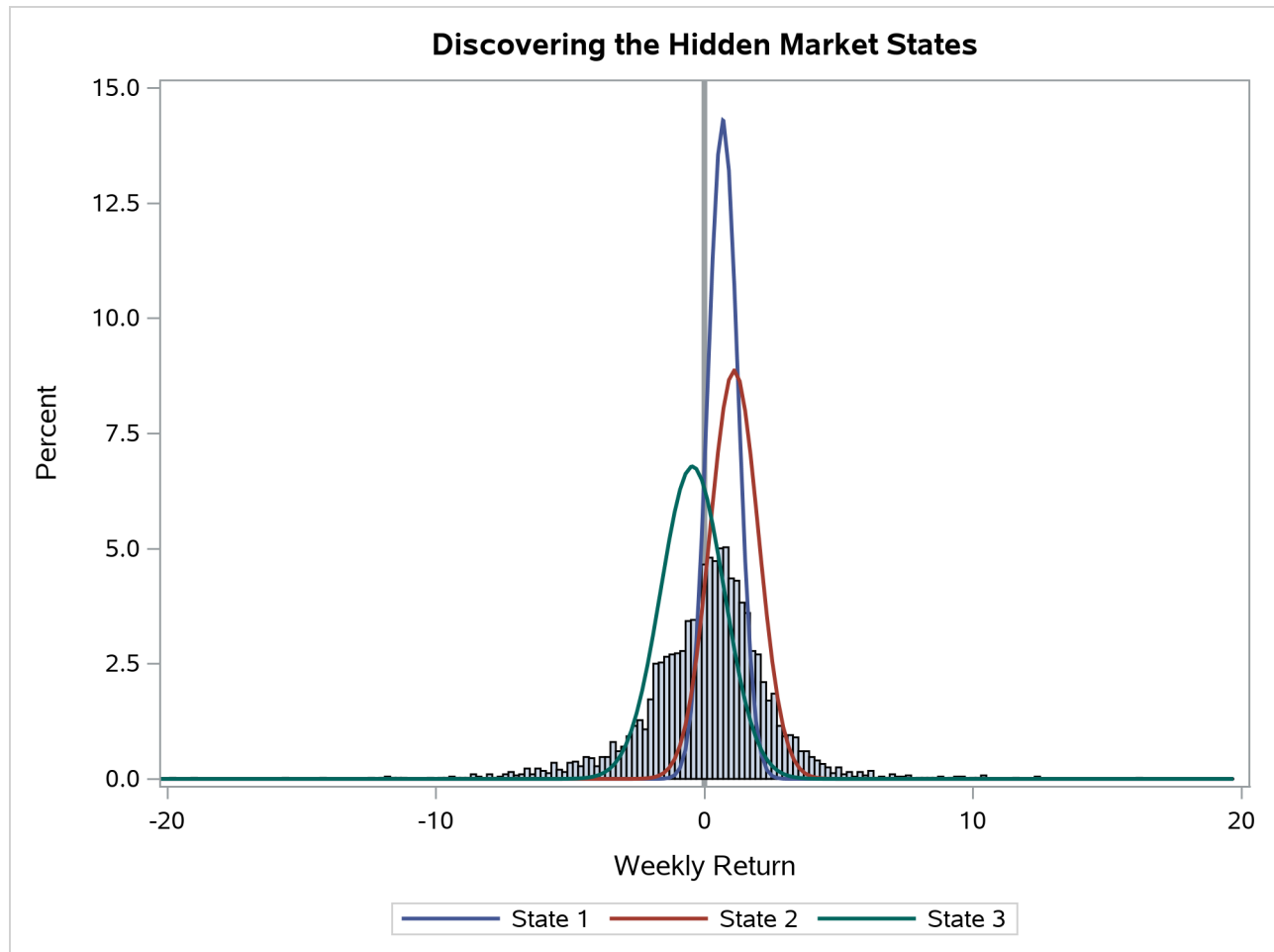
- The first group has a 98% chance of switching to itself and a 2% chance of switching to the second group.
- The third group has a 97% chance of switching to itself and a 3% chance of switching to the second group.
- The second group has a 94% chance of switching to itself and equal 3% chances of switching to the first and third groups.
- The first and third group have zero chance of switching to each other.
- When the market is in one group, it lasts for a long time before switching to the other groups.

Hence, according to the eight-state RS-AR(4) model, there is one more market state besides the well-known bear market (the first group) and bull market (the third group): the transition market state (the second group). Figure 13.68 shows the relationship between these market states.

Figure 13.68 Market States and Their Transitions

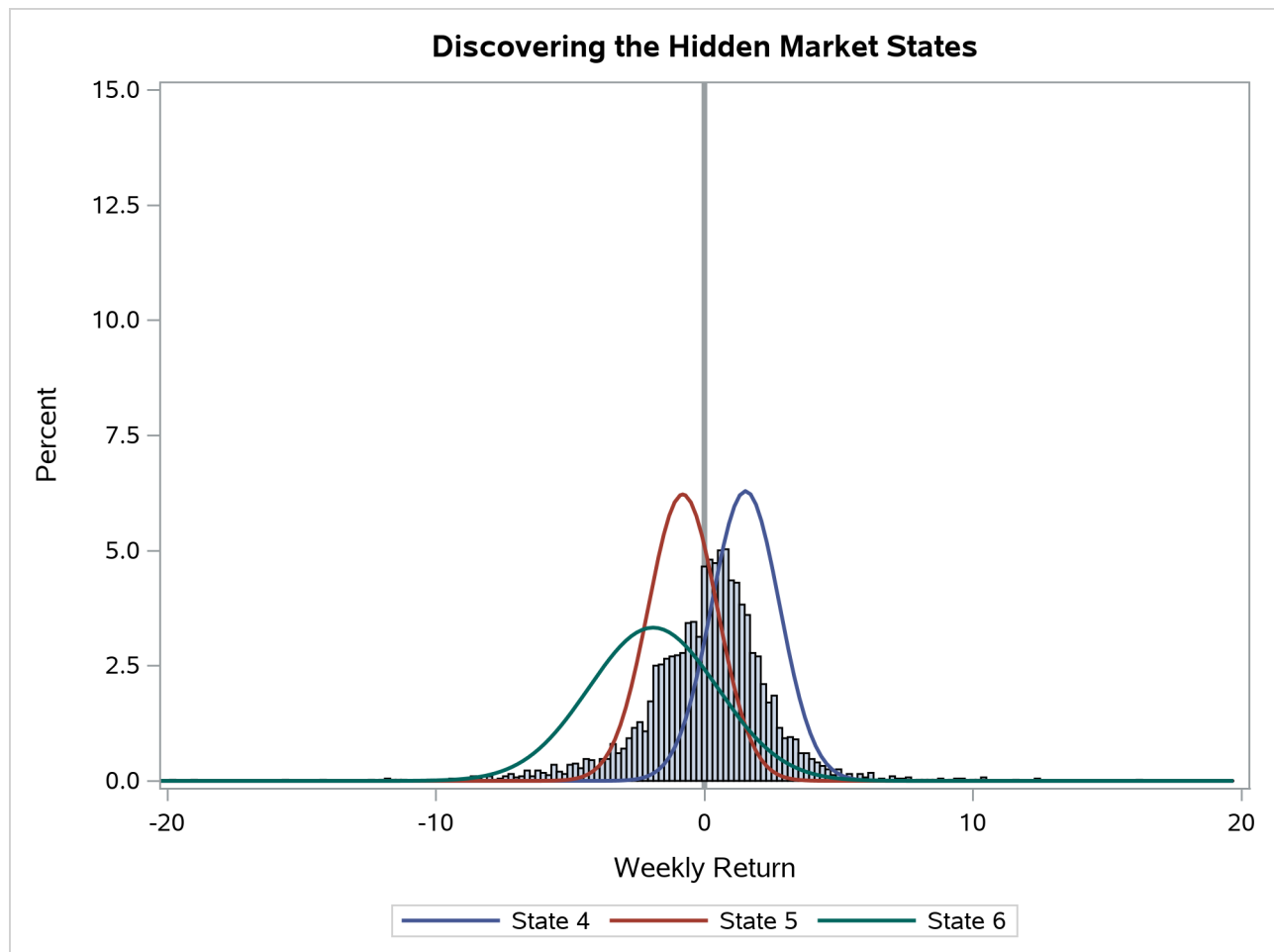
The following statement plots three states that correspond to the bull market state, which are shown in Output 13.1.21:

```
%plotState(&ds.In, returnw, 1, 3);
```


Output 13.1.21 Three Gaussian Kernels for the Bull Market State

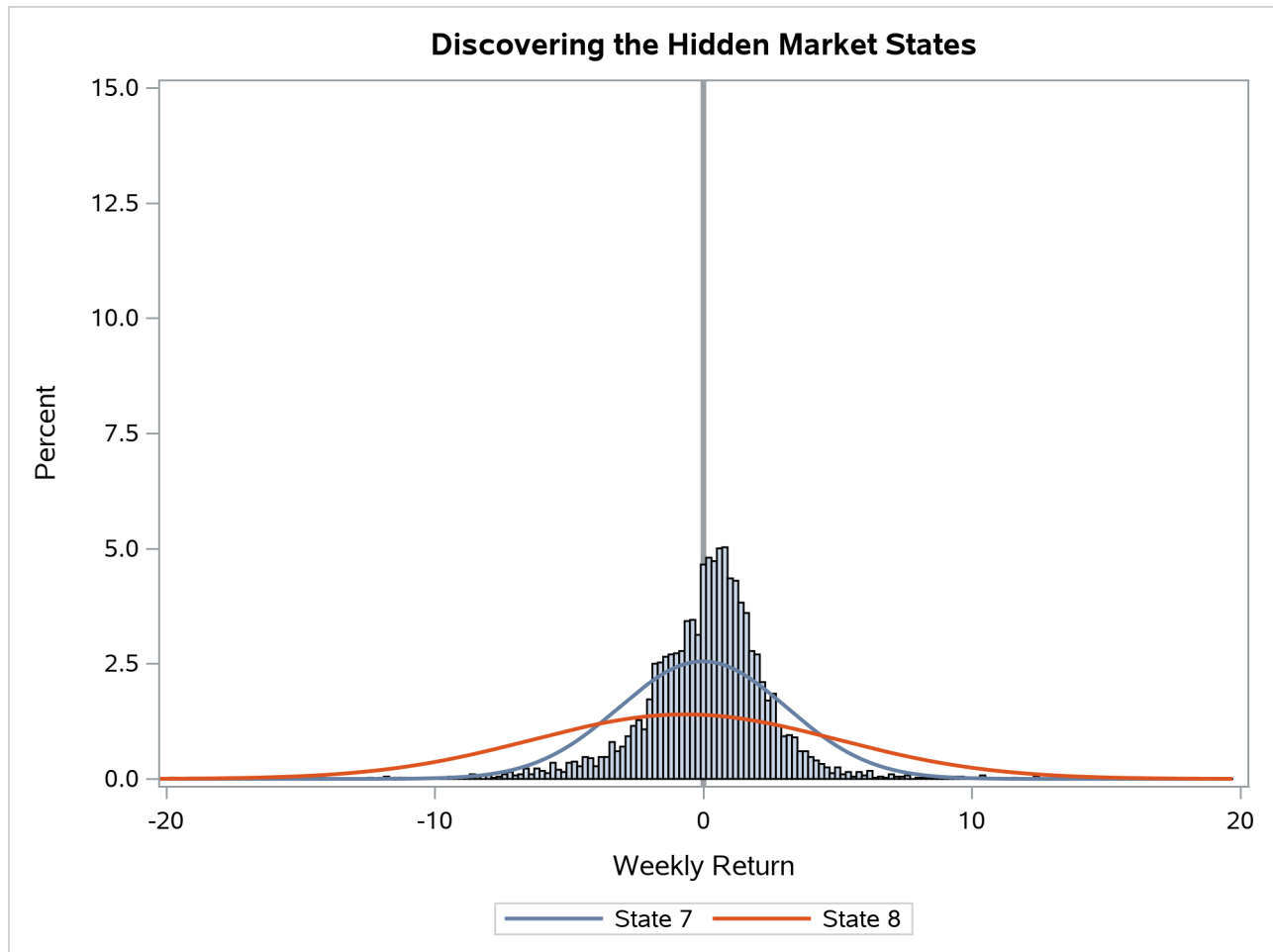
The following statement plots three states that correspond to the transition market state, which are shown in [Output 13.1.22](#):

```
%plotState(&ds.In,returnw,4,6);
```

Output 13.1.22 Three Gaussian Kernels for the Transition Market State

The following statement plots two states that correspond to the bear market state, which are shown in [Output 13.1.23](#):

```
%plotState(&ds.In,returnw,7,8);
```

Output 13.1.23 Two Gaussian Kernels for the Bear Market State

The following statements plot the decoded path of states according to the eight-state RS-AR(4) model; the result is shown in [Output 13.1.24](#).

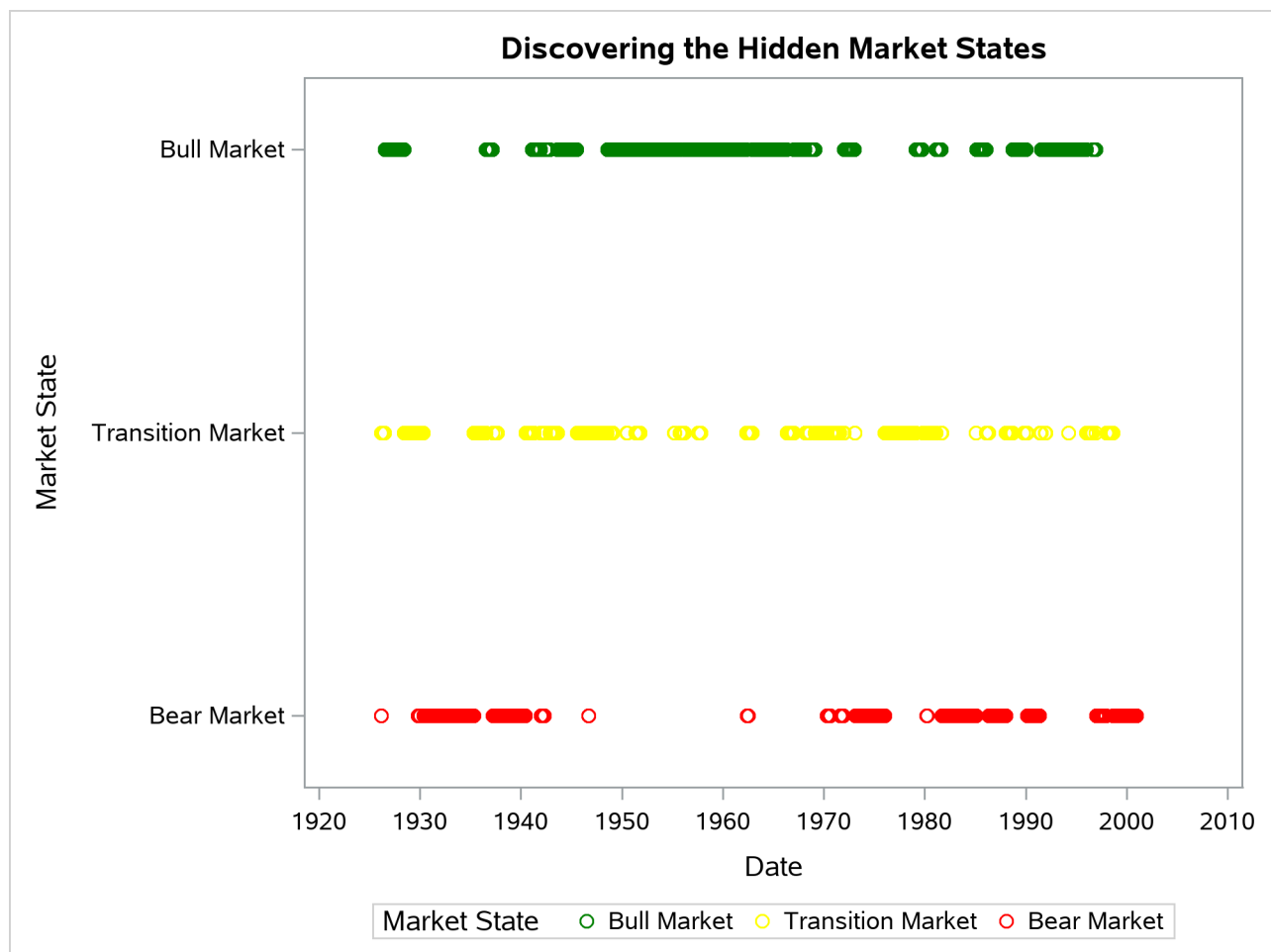
```
* plot decoded results;
data &ds.InDecode_k8to8_p4to4; set mycas.&ds.InDecode_k8to8_p4to4; run;
proc sort data=&ds.InDecode_k8to8_p4to4; by date; run;
data decodeMaketStates;
  set &ds.In(where=(date>='30JAN1926'd));
  set &ds.InDecode_k8to8_p4to4;
  array returnws returnw1-returnw3;
  array prices price1-price3;
  if(state~=.) then do;
    if(state=1 | state=2 | state=3) then do;
      returnws[1] = returnw; prices[1] = price;
      marketState="Bull Market    ";
    end;
    if(state=4 | state=5 | state=6) then do;
      returnws[2] = returnw; prices[2] = price;
      marketState="Transition Market";
    end;
  end;
```

```

    if (state=7 | state=8) then do;
        returnws[3] = returnw; prices[3] = price;
        marketState="Bear Market      ";
    end;
end;
label marketState="Market State";
run;
proc sgplot data=decodeMaketStates;
    styleattrs datacontrastcolors=(red yellow green);
    scatter x=date y=marketState / group=marketState nomissinggroup;
    yaxis label="Market State" offsetmin=0.1 offsetmax=0.1;
    keylegend / sortorder=reverseauto;
run;

```

Output 13.1.24 Decoded Market States



The following statements plot the decoded weekly returns, which are shown in [Output 13.1.25](#):

```

proc sgplot data=decodeMaketStates;
    series x=date y=returnw1 / break lineattrs=(color=green)
        legendlabel="Bull Market";
    series x=date y=returnw2 / break lineattrs=(color=yellow)
        legendlabel="Transition Market";

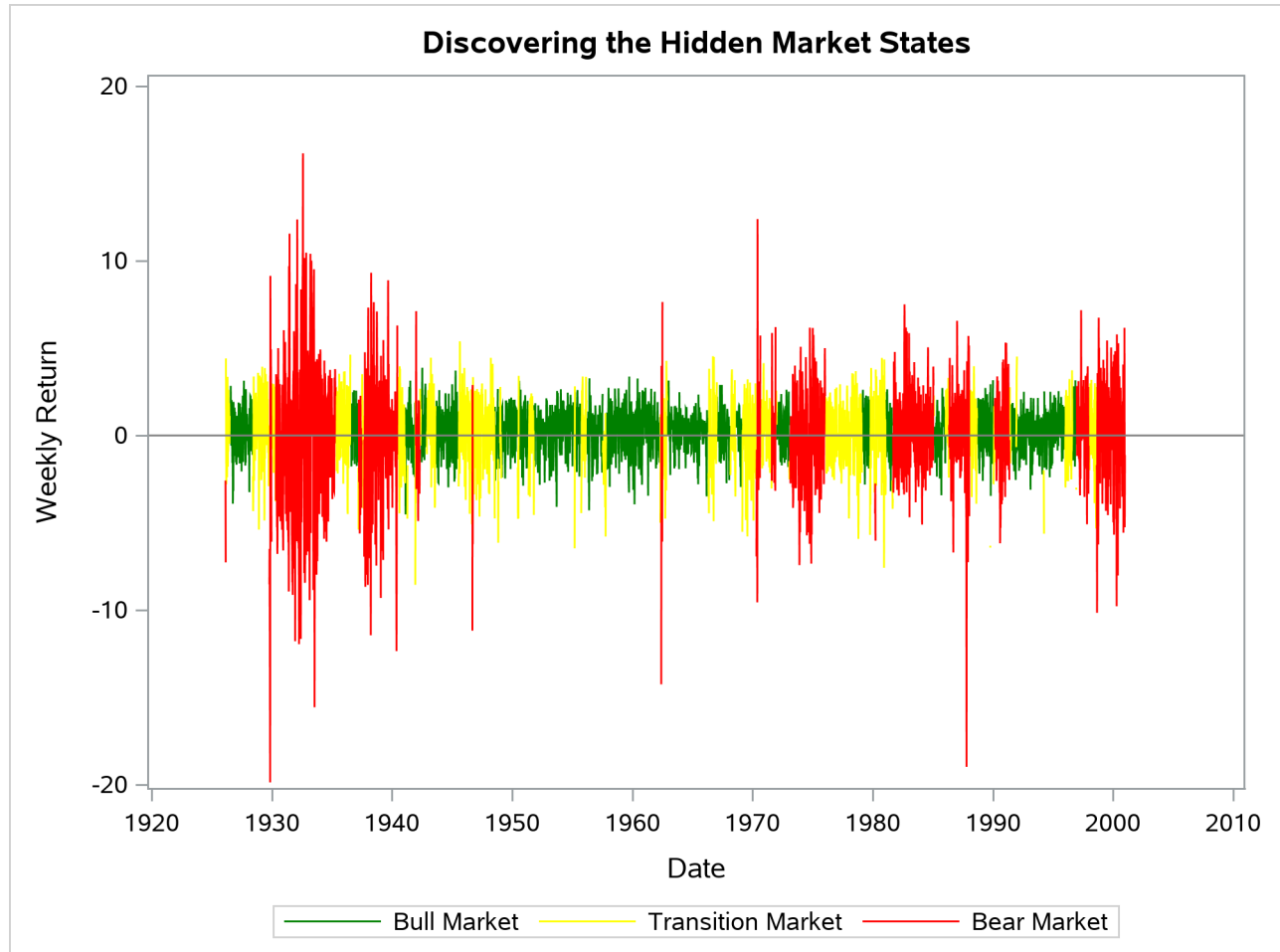
```

```

series x=date y=returnnw3 / break lineattrs=(color=red)
                        legendlabel="Bear Market";
refline 0 / axis=y lineattrs=(color=gray thickness=1);
yaxis max=20 min=-20 label="Weekly Return";
run;

```

Output 13.1.25 Decoded Weekly Returns

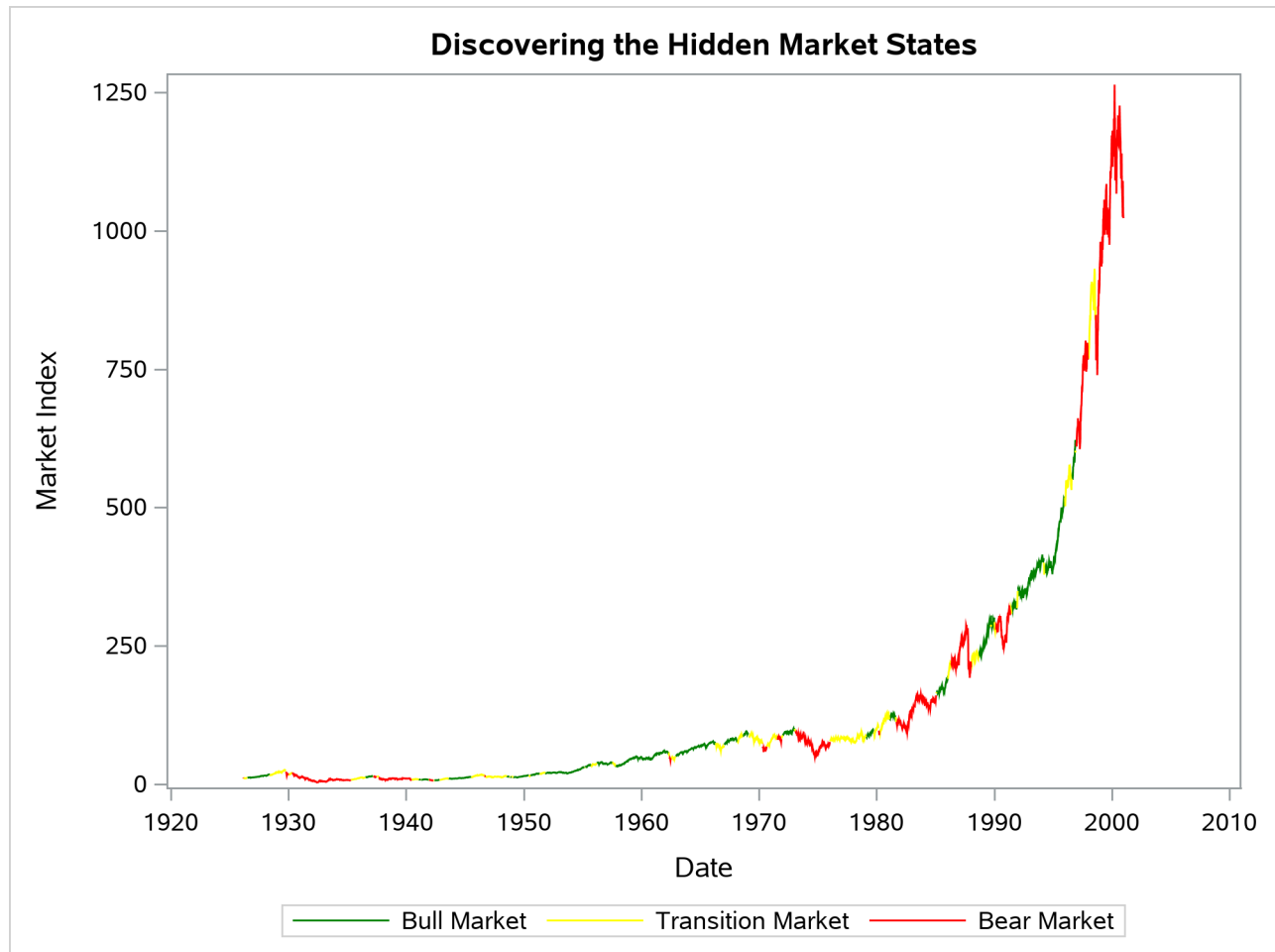


The following statements plot the decoded stock prices, which are shown in [Output 13.1.26](#):

```

proc sgplot data=decodeMaketStates;
  series x=date y=price1 / break lineattrs=(color=green)
                        legendlabel="Bull Market";
  series x=date y=price2 / break lineattrs=(color=yellow)
                        legendlabel="Transition Market";
  series x=date y=price3 / break lineattrs=(color=red)
                        legendlabel="Bear Market";
  yaxis label="Market Index";
run;

```

Output 13.1.26 Decoded Stock Prices

The Out-of-Sample Testing

In the model selection process, the AIC, but not the in-sample goodness of fit (that is, the log likelihood), is used as the criterion for selecting the best model because a bias-variance trade-off exists. In this section, three types of out-of-sample forecasts are tested to check whether the selected model has a good forecast performance.

The Value-at-Risk (VaR) Forecast

You can specify the INMODEL= option in the SCORE statement to forecast the VaRs at different levels, such as 1%, 5%, and 10%, as shown in [Output 13.1.27](#). The following macros can predict the VaRs for the out-of-sample period and evaluate the predictive performance by the likelihood ratio (LR) test for the unconditional coverage of the VaR forecast (Kuester, Mittnik, and Paolella 2006):

```
* print Predictive Performance of VaR Forecasts;
%let alpha1 = 0.80;
%let alpha2 = 0.90;
%let alpha3 = 0.98;
%macro forecastVaR(myCasLib, dsData, dsModel, timeId, varReturn,
    k, p, iStart, iEnd, oosStart, oosEnd,
```

```

dsForecastPrefix);
%do i = &iStart. %to &iEnd.;
    proc hmm data=&myCasLib..&dsData.;
        score inmodel=&myCasLib..&dsModel.;
        forecast out=&myCasLib..&dsForecastPrefix.&i. alpha=&&alpha&i. online;
    run;
    data &dsForecastPrefix.&i.;
        set &myCasLib..&dsForecastPrefix.&i.;
    run;
    proc sort data=&dsForecastPrefix.&i.; by &timeId.; run;
    data &dsForecastPrefix.&i.;
        set &dsForecastPrefix.&i. (FIRSTOBS=&oosStart. OBS=&oosEnd.
            keep=&timeId. &varReturn._Q1
            rename=(&varReturn._Q1=&varReturn._Q1_&k.&p.&i.));
        time=_N_; nState = &k; yLag = &p;
    run;
%end;
%mend forecastVaR;

%macro evaluateVaR(dsData,dsForecastPrefix,varReturn,k,p,iStart,iEnd,n,dsOut);
    data &dsData.out;
        set &dsData.;
        if (date>&cutDate.) then output;
    run;
    data forecastData;
        set &dsData.out;
        time=_N_;
    run;
    data forecastData;
        merge &dsForecastPrefix.: forecastData;
        by time;
    run;
    data &dsOut.;
        set forecastData;
        retain %do i = &iStart. %to &iEnd.; cviol&k.&p.&i. 0 %end; ;
        retain %do i = &iStart. %to &iEnd.; c&varReturn.&k.&p.&i. 0 %end; ;
        %do i = &iStart. %to &iEnd.;
            if &varReturn. le &varReturn._Q1_&k.&p.&i. then do;
                cviol&k.&p.&i.=cviol&k.&p.&i.+1;
            end;
            c&varReturn.&k.&p.&i. = c&varReturn.&k.&p.&i.
                + &varReturn._Q1_&k.&p.&i.;
        %end;
        if _N_ = &n. then do;
            %do i = &iStart. %to &iEnd.;
                Norminal = (1-&&alpha&i.)/2;
                Viol = cviol&k.&p.&i. / &n.;
                LR = 2*(cviol&k.&p.&i.*log(Viol)+(&n.-cviol&k.&p.&i.)*log(1-Viol)
                    -(cviol&k.&p.&i.*log(Norminal)+(&n.-cviol&k.&p.&i.)
                    *log(1-Norminal)));
                pValue = 1 - cdf("CHISQUARE", LR, 1);
                meanVaR = c&varReturn.&k.&p.&i. / &n.;
                output;
            %end;
        end;
    %end;

```

```

end;
label Norminal='Target Prob.' Viol='Violation Ratio' LR='LR Stat.'
    pValue='Pr > ChiSq' meanVaR='Avg. of VaR' nState='Number of States'
    yLag='AR Lag';
keep nState yLag Norminal Viol LR pValue meanVaR;
run;
proc print data=dsOut.noobs label; format Viol LR pValue meanVaR 6.4; run;
%mend evaluateVaR;

%macro VaR(k,p);
    %forecastVaR(myCasLib=mycas, dsData=ds.,
        dsModel=ds.inMLModel_k&k.To&k._p&p.To&p.,
        timeId=date, varReturn=returnw,
        k=&k., p=&p., iStart=1, iEnd=3,
        oosStart=%eval(3999-(5-&p.)), oosEnd=%eval(4854-1-(5-&p.)),
        dsForecastPrefix=ds.Forecastk&k._p&p.);
    %evaluateVaR(dsData=ds., dsForecastPrefix=ds.Forecastk&k._p&p.,
        varReturn=returnw, k=&k., p=&p., iStart=1, iEnd=3, n=855,
        dsOut=VaR_outputk&k._p&p.);
%mend VaR;

%VaR(8,4);

```

Output 13.1.27 Predictive Performance of VaR Forecasts**Discovering the Hidden Market States**

Number of AR States	AR Lag	Target Prob.	Violation Ratio	LR Stat.	Pr > ChiSq	Avg. of VaR
8	4	0.10	0.1053	0.2592	0.6107	-2.718
8	4	0.05	0.0538	0.2541	0.6142	-3.691
8	4	0.01	0.0129	0.6503	0.4200	-5.974

According to the p -values in the Pr > ChiSq column in [Output 13.1.27](#), at the 5% significance level, no tests can reject the null hypothesis that the number of violations is correct. Hence, the eight-state RS-AR(4) model has the correct unconditional coverage for the 1%, 5%, and 10% VaR forecasts.

The Out-of-Sample Log Likelihood

In fact, besides the good predictability of the tail of the distribution of weekly returns as shown in the VaR forecast analysis, the eight-state RS-AR(4) model also provides a very good prediction of the whole distribution of weekly returns, which can be seen by comparing the average weekly log likelihoods of the in-sample period and the out-of-sample period. As shown in [Output 13.1.28](#), compared to both the simplest model (the two-state RS-AR(0) model, which has the fewest parameters and the worst in-sample fit) and the most complex model (the nine-state RS-AR(5) model, which has the most parameters and the best in-sample fit), the eight-state RS-AR(4) model has the best out-of-sample forecast ability (that is, the largest average weekly log likelihood in the out-of-sample period).

```

* print in-sample and out-of-sample average weekly log likelihood;
proc hmm data=mycas.ds.
    outstat=mycas.ds.Stat_k2to9_p0to5;
    score inmodel=mycas.ds.inmlmodel_k2to9_p0to5;

```



```

    evaluate out=mycas.&ds.Eval_k2to9_p0to5;
run;
data inLL;
    set mycas.&ds.Eval_k2to9_p0to5 (where=(date='26DEC2000'd)
        rename=(logLikelihood=isLL));
run;
data fullLL;
    set mycas.&ds.Eval_k2to9_p0to5 (where=(date='22DEC2017'd)
        rename=(logLikelihood=fullLL));
run;
data oosLL;
    merge inLL(in=a) fullLL(in=b);
    by modelIndex;
    oosLL = (fullLL - isLL)/855;
    if(a=b);
    drop date isLL fullLL;
run;

data inLL;
    set mycas.&ds.InMLStat_k2to9_p0to5;
    isLL = logLikelihood / (3999-5);
    keep modelIndex nState yLag isLL;
run;
proc sort data=inLL; by modelIndex; run;

data avgLL;
    merge inLL(in=a) oosLL(in=b);
    by modelIndex;
    if(a=b and (modelIndex=1 or modelIndex=41 or modelIndex=48));
    drop modelIndex;
run;

proc print data=avgLL noobs label;
    label nState='k' yLag='p' isLL='In-Sample' oosLL='Out-Of-Sample';
run;

```

Output 13.1.28 Comparison of Average Weekly Log Likelihoods**Discovering the Hidden Market States**

k	p	In-Sample	Out-Of-Sample
2	0	-2.12513	-2.14688
8	4	-2.05835	-2.10316
9	5	-2.05494	-2.12056

Trading Strategies and Portfolio Values

As mentioned at the beginning of this example, if you knew it would be a bull market, you might choose to buy the stock; if you knew it would be a bear market, you might choose to sell the stock. In this section, the market states are forecasted and then several trading strategies are compared.

The following statements forecast and plot the probability distribution of market states in [Output 13.1.29](#). The forecast of market states is in probabilities, which is similar to a weather forecast. For example, a forecast of

60% chance of bear market, 39% chance of transition market, and 1% chance of bull market is comparable to a forecast of 60% chance of raining, 39% chance of being cloudy, and 1% chance of being sunny.

```
proc hmm data=mycas.&ds.;
    score inmodel=mycas.&ds.InMLModel_k8To8_p4To4;
    forecast out=mycas.&ds.Forecast_k8to8_p4to4 online;
    display / excludeall;
run;

%let forecastStartDate = '26DEC2000'd;
%let forecastEndDate = '22DEC2017'd;
data ForecastProb;
    set mycas.&ds.Forecast_k8to8_p4to4
        (where=(date>=&forecastStartDate. and date<=&forecastEndDate.));
run;
proc sort data=ForecastProb; by date; run;
data ForecastProb;
    set ForecastProb(keep=state1 - state8) end=last;
    set &ds.(where=(date>=&forecastStartDate. and date<=&forecastEndDate.)
        keep=date price);
    * probability distribution of market states;
    bullProb = state1+state2+state3;
    transitionProb = state4+state5+state6;
    bearProb = state7+state8;
    bearTransitionProb = bearProb + transitionProb;
run;
* plot forecasted prob. dist. of market states and trading strategies;
%macro plotForecastAndTradingStrategies(ds, bearThreshold, bullThreshold,
    oneMinusBullThreshold);

proc sgplot data=&ds.;
    band x=date lower=0 upper=bearProb /
        fillattrs=(color=red) name="bear" legendlabel="Bear Market";
    band x=date lower=bearProb upper=bearTransitionProb /
        fillattrs=(color=yellow)
        name="transition" legendlabel="Transition Market";
    band x=date lower=bearTransitionProb upper=1 /
        fillattrs=(color=green) name="bull" legendlabel="Bull Market";
    series x=date y=price / y2axis lineattrs=(color=blue)
        name="index" legendlabel="Market Index";
    %if %length(&bearThreshold.)>0 %then %do;
        refline &bearThreshold. / axis=y lineattrs=(color=red thickness=3)
            label="Bear Threshold=&bearThreshold." labelloc=inside
            labelattrs=(weight=bold);
    %end;
    %if %length(&bearThreshold.)>0 %then %do;
        refline &oneMinusBullThreshold. / axis=y
            lineattrs=(color=green thickness=3)
            label="Bull Threshold=&bullThreshold." labelloc=inside
            labelattrs=(weight=bold);
    %end;
    xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
        tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
    yaxis label="Forecasted Prob. Dist. of Market States";
    y2axis label="Market Index";
```

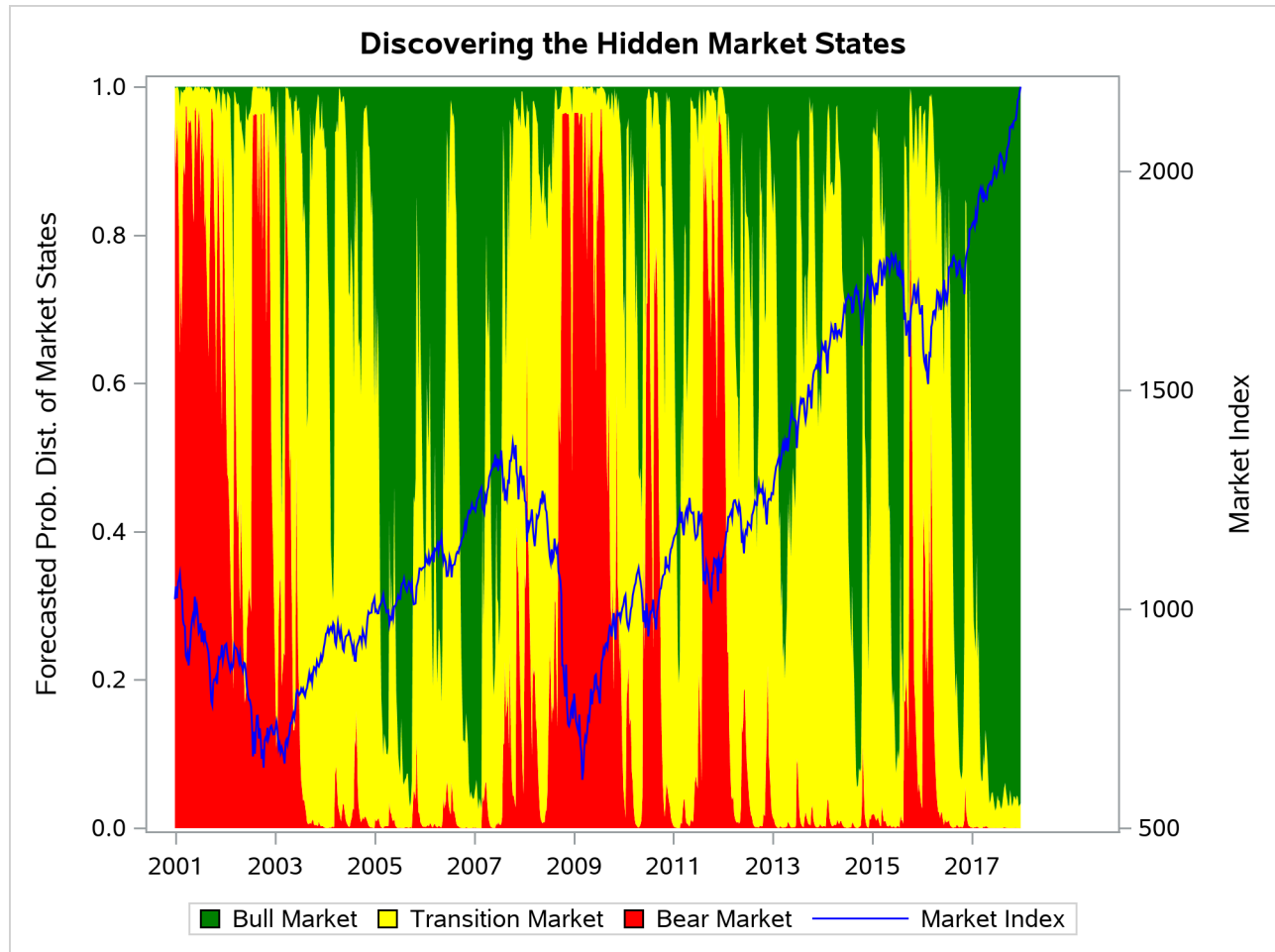
```

keylegend "bull" "transition" "bear" "index"/
location=outside position=bottom;
run;
%mend;

%plotForecastAndTradingStrategies (ForecastProb) ;

```

Output 13.1.29 Forecasted Probabilities of Market States



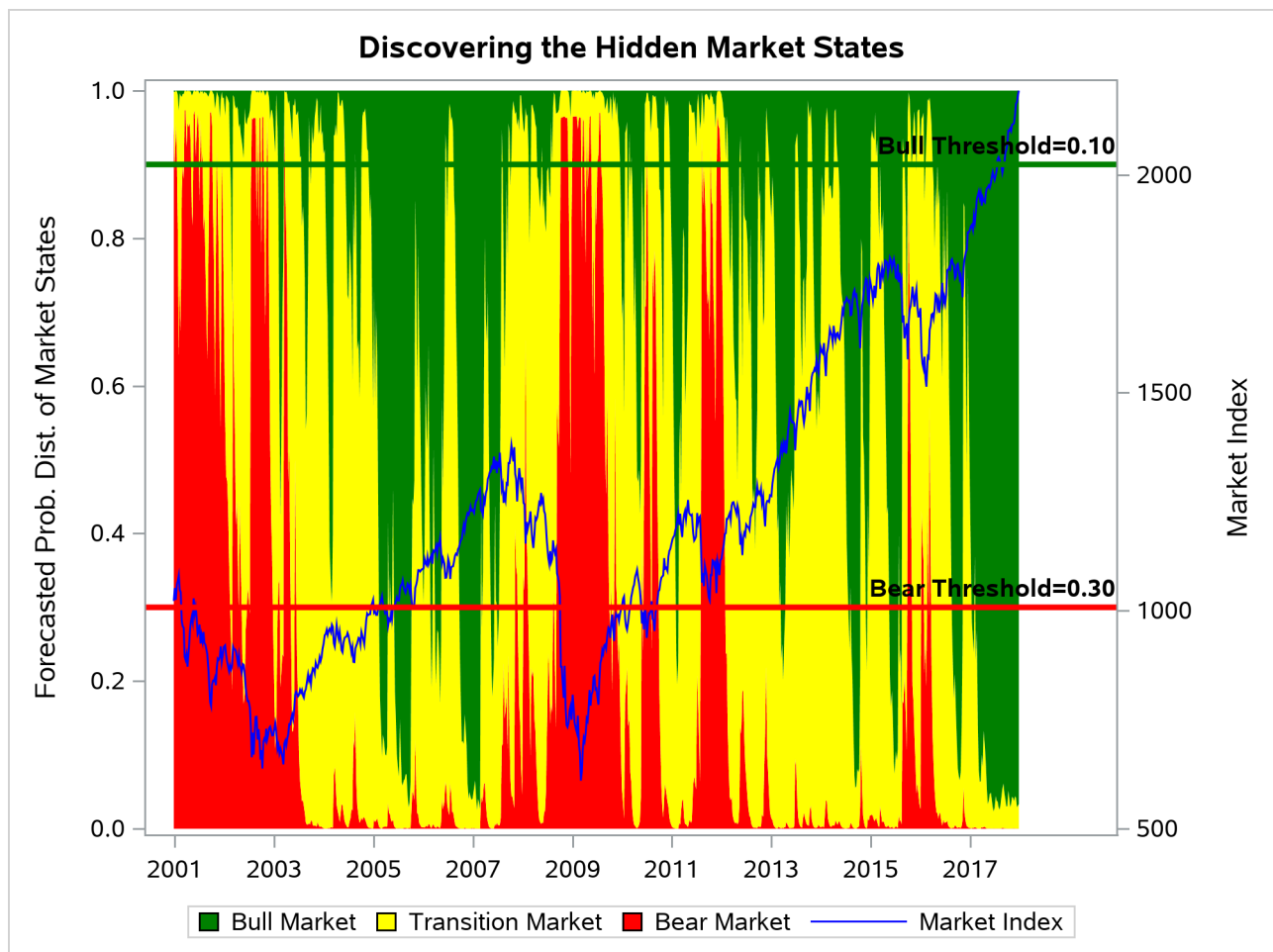
In this example, four simple trading strategies are considered:

- Risk-free strategy: hold the bond for the entire time period. This strategy does not depend on any forecast, and it is the first benchmark strategy.
- Market strategy: hold the stock for the entire time period. This strategy does not depend on any forecast, either. It reflects the market performance, and it is the second benchmark strategy.
- Bear strategy: sell the stock if the chance of a bear market in the next week is no less than 30%; otherwise, buy the stock.
- Bull strategy: buy the stock if the chance of a bull market in the next week is no less than 10%; otherwise, sell the stock.

The thresholds in bear and bull strategies, 30% and 10% respectively, are ad hoc, which reflect the investor's risk attitude. The trading strategy of switching between bond and stock can also be analogous to the decision of traveling with or without umbrella. The risk-free strategy is analogous to a decision to always travel with an umbrella; the market strategy is analogous to never traveling with umbrella; the bear strategy is analogous to traveling with an umbrella if the chance of raining tomorrow is no less than 30%; and the bull strategy is analogous to traveling without an umbrella if the chance of being sunny tomorrow is no less than 10%. The following statements plot the thresholds for the bear and bull trading strategies:

```
%let bearThreshold=0.30;
%let bullThreshold=0.10;
%let oneMinusBullThreshold=0.90;
%plotForecastAndTradingStrategies(ForecastProb, &bearThreshold.,
                                &bullThreshold., &oneMinusBullThreshold.);
```

Output 13.1.30 Thresholds for Bull and Bear Trading Strategies



The following statements calculate the portfolio values and plot the wealth curves of four trading strategies:

```
%let strategy1=RiskFree;
%let strategy2=Market;
%let strategy3=Bear;
%let strategy4=Bull;
```

```

%let strategy1Label=Risk Free;
%let strategy2Label=Market;
%let strategy3Label=Bear;
%let strategy4Label=Bull;
* assume that 1 year = 252 trading days = 50.4 weeks;
* weekly risk free rate =  $\exp(\ln(1+APY)/50.4)-1$ ;
%let riskFreeRate = 0.0002661; * APY=1.35%;
%let initialMoney=1;

%macro port_val_cal(stockPosition, stockPrice, riskFreeRate, strategy,
                    initialMoney, rowNo);
    if (&rowNo. = 1) then do;
        &strategy.CashValue = &initialMoney.-&initialMoney.*&stockPosition.;
        &strategy. = &initialMoney.;
        &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
        &strategy.CurrentCashValue = &strategy.CashValue;
    end;
    else do;
        &strategy.CurrentCashValue = &strategy.CashValue*(1+&riskFreeRate.);
        &strategy. = &strategy.CurrentCashValue
            + &strategy.StockAmount*&stockPrice.;
        &strategy.StockAmount = &strategy.*&stockPosition./&stockPrice.;
        &strategy.CashValue = &strategy.*(1-&stockPosition.);
    end;
    &strategy.ExcessReturn = &strategy./lag(&strategy.)-1-&riskFreeRate.;
    &strategy.ExcessReturn2 = &strategy.ExcessReturn**2;
%mend port_val_cal;

%macro port_val_gen(ForecastProb, bearThreshold, bullThreshold, riskFreeRate,
                    initialMoney, startDate, endDate);
    data PortValPath;
        set &ForecastProb. end=last;
        array stockPos[4] %do i = 1 %to 4; &&strategy&i..StockPos %end; ;
        retain
            %do i = 1 %to 4;
                &&strategy&i..CashValue 0 &&strategy&i..StockAmount 0
                &&strategy&i..CurrentCashValue 0 &&strategy&i..CumExcessReturn 0
                &&strategy&i..CumExcessReturn2 0
            %end; ;
        * stock position for each strategy;
        stockPos[1] = 0;
        stockPos[2] = 1;
        if(bearProb>=&bearThreshold.) then stockPos[3]=0;
        else stockPos[3]=1;
        if(bullProb>=&bullThreshold.) then stockPos[4]=1;
        else stockPos[4]=0;
        * wealth for each strategy;
        %do i = 1 %to 4;
            %port_val_cal(stockPosition=stockPos[&i.], stockPrice=price,
                riskFreeRate=&riskFreeRate., strategy=&&strategy&i..,
                initialMoney=&initialMoney., rowNo=_N_);
            &&strategy&i..CumExcessReturn+&&strategy&i..ExcessReturn;
            &&strategy&i..CumExcessReturn2+&&strategy&i..ExcessReturn2;
        %end;
    run;

```

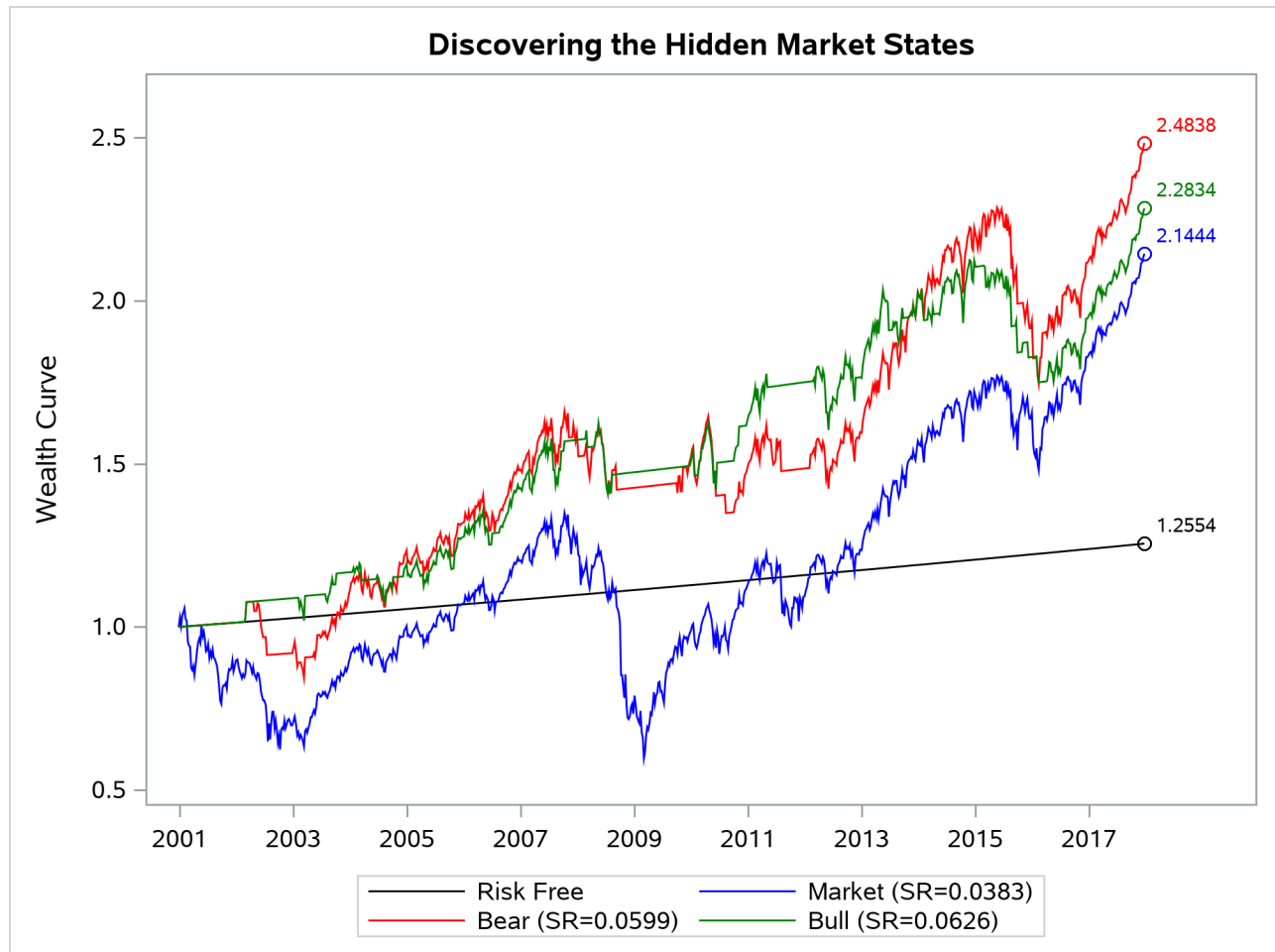
```

* Sharpe ratio and portfolio final wealth;
if last then do;
    &strategy1.LastPorVal = &strategy1.;
    %do i = 2 %to 4;
        &&strategy&i..LastPorVal = &&strategy&i..;
        &&strategy&i..ExcessReturnMean =
            &&strategy&i..CumExcessReturn/(_N_-1);
        &&strategy&i..ExcessReturnStd =
            sqrt(&&strategy&i..CumExcessReturn2/(_N_-1)
                - &&strategy&i..ExcessReturnMean**2);
        &&strategy&i..SharpeRatio = round(&&strategy&i..ExcessReturnMean
            /&&strategy&i..ExcessReturnStd, 0.0001);
        call symputx("&&strategy&i..SharpeRatio",
            &&strategy&i..SharpeRatio, 'G');
    %end;
end;
run;
%mend;

* plot portfolio values of different trading strategies;
%let color1=black;
%let color2=blue;
%let color3=red;
%let color4=green;
%macro perf_comp_plot_gen(nStrategies);
    proc sgplot data=PortValPath cycleattrs;
        %do i = 1 %to &nStrategies.;
            %let strategy=&&strategy&i..;
            series x=date y=&&strategy&i.. / name="&strategy."
                lineattrs=(color=&&color&i..)
                %if (&i.=1) %then %do;
                    legendlabel="&&strategy&i..Label.";
                %end;
                %else %do;
                    legendlabel="&&strategy&i..Label. (SR=&&&strategy.SharpeRatio.)";
                %end;
            scatter x=date y=&&strategy&i..LastPorVal / datalabelpos=topright
                markerattrs=(color=&&color&i..)
                datalabel=&&strategy&i..LastPorVal;
        %end;
        xaxis values=("01JAN2001"d to "31DEC2017"d by year) display=(nolabel)
            tickvalueformat=year4. offsetmin=0.03 offsetmax=0.15;
        yaxis max=2.6 min=0.5 label="Wealth Curve";
        keylegend %do i = 1 %to &nStrategies.; "&&strategy&i.." %end; /
            location=outside position=bottom;
    run;
%mend perf_comp_plot_gen;

%port_val_gen(ForecastProb=ForecastProb,
    bearThreshold=&bearThreshold., bullThreshold=&bullThreshold.,
    riskFreeRate=&riskFreeRate., initialMoney=&initialMoney.,
    startDate=&forecastStartDate., endDate=&forecastEndDate.);
%perf_comp_plot_gen(nStrategies=4);

```

Output 13.1.31 Wealth Curves of Four Trading Strategies

As shown in [Output 13.1.31](#), thanks to the forecast of market states by the eight-state RS-AR(4) model, both bear and bull strategies beat the market: compared to the market strategy, they increase the Sharpe ratio more than 50% and get higher portfolio values at the end of out-of-sample period.

Example 13.2: Clustering Time Series by Using the Gaussian Hidden Markov Model

Clustering time series is a very common method of analyzing cross-sectional time series data. For more information, see Liao (2005) and Rani and Sikka (2012) and references therein. Especially for methods that use hidden Markov models (HMMs), see Smyth (1997), Oates, Firoiu, and Cohen (1999), and Ghassempour, Giosi, and Maeder (2014).

This example illustrates an algorithm for quickly clustering time series. With prior knowledge of the number of clusters and the number of states for each cluster, the algorithm creates the initial clusters according to the distribution of the likelihoods on an arbitrary HMM. Then, one HMM on each cluster is estimated and the time series are moved between clusters according to their likelihoods on those HMMs. The process is repeated until the algorithm converges or the maximum number of iterations is reached (Liao 2005).

In this example, the data generating process (DGP) is the same one that is described in Smyth (1997). There are two bivariate Gaussian HMMs. For the first Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

For the second Gaussian HMM, the parameters are

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 0.4 & 0.6 \\ 0.6 & 0.4 \end{pmatrix}, \mathbf{B} = \left\{ \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

The only difference between these two Gaussian HMMs is that they have different transition probability matrices, so the time series that are generated from the first HMM have slower dynamics than those from the second HMM. Hence, the clustering time series algorithm should assign the time series that are generated from the first HMM to a cluster, and assign those from the second HMM to another cluster, if there is prior knowledge that the number of clusters is two and the number of states for each cluster is two.

The following macro generates the cross-sectional time series data table for analysis:

```

title "Clustering Time Series";

* create a data table dgpTable
consisting of nSections sections and T observations for each section;
* then copy the data table dgpTable to the data table trainingTable
and the CAS table casTrainingTable;
%macro createClustersTable(nSections,T,
                           dgpTable,trainingTable,casTrainingTable);

    %let seed = 12345;
    %let nClustersDGP = 2;
    %let clportion = 0.5;

    %let clpi1 = 0.5;
    %let cla11 = 0.6;
    %let cla22 = 0.6;
    %let clmu1 = 0;
    %let clsigma1 = 1;
    %let clmu2 = 3;
    %let clsigma2 = 1;

    %let c2pi1 = 0.5;
    %let c2a11 = 0.4;
    %let c2a22 = 0.4;
    %let c2mu1 = 0;
    %let c2sigma1 = 1;
    %let c2mu2 = 3;
    %let c2sigma2 = 1;
    data &dgpTable.;

```



```

do sec = 1 to &nSections.;
  * which cluster;
  * (1) randomly assign clusters;
  * u = uniform(&seed.);
  * if(u<=&clportion.) then cluster = 1;
  * else cluster = 2;
  * (2) all time series for one cluster are in one block;
  if(sec/&nSections.<=&clportion.) then cluster = 1;
  else cluster = 2;
  %do i = 1 %to &nClustersDGP.;
    if(cluster=&i.) then do;
      do t = 1 to &T.;
        if(t=1) then p = &c&i.p11;
        else do;
          if(lagState=1) then p = &c&i.a11;
          else p = 1-&c&i.a22;
        end;
        u = uniform(&seed.);
        if(u<=p) then state = 1;
        else state = 2;
        if(state=1) then
          y = &c&i.mu1. + sqrt(&c&i.sigma1.)*normal(&seed.);
        else
          y = &c&i.mu2. + sqrt(&c&i.sigma2.)*normal(&seed.);
        output;
        lagState = state;
      end;
    end;
  %end;
end;
run;

* trainingTable is sorted by sec and t;
data &trainingTable.;
  set &dgpTable.;
  keep sec t y;
run;

data &casTrainingTable.;
  set &trainingTable.;
run;
%mend;

```

The following macro initializes the clusters:

```

%macro estSection(casTrainingTable,casSectionTable,selectedSec,i,T,
                  evalTable,casModelTable);
  data &casSectionTable.&selectedSec.;
    set &casTrainingTable.;
    if(sec=&selectedSec.);
  run;

  proc hmm data=&casSectionTable.&selectedSec.;
    id section=sec time=t;

```

```

        model y / method=ml type=gaussian nstate=&k.;
        optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
        score outmodel=&casModelTable.&i.;
run;

proc hmm data=&casTrainingTable.;
    evaluate out=mycas.tempEval;
    score inmodel=&casModelTable.&i.;
run;

data &evalTable.&i.;
    set mycas.tempEval;
    logLikelihood&i. = logLikelihood;
    if(t=&T.) then output;
    keep sec logLikelihood&i.;
run;

proc sort data=&evalTable.&i.; by sec; run;
%mend;

%macro assignSeriesToClusters(evalTable,nClusters);
    data &evalTable.;
        merge &evalTable.1 - &evalTable.&nClusters.;
        by sec;
        retain sumLogLikelihood 0;
        array arr[*] logLikelihood1 - logLikelihood&nClusters.;
        sumLogLikelihood = sumLogLikelihood + max(of arr[*]);
        pCluster = whichn(max(of arr[*]), of arr[*]);
        keep sec pCluster sumLogLikelihood;
    run;
%mend;

* initialize clusters;
* each section has T observations;
%macro ctsInitClusters(casTrainingTable,nClusters,nSections,T,
                      evalTable,casModelTable);
    * as a start, use first section or randomly select a section;
    %let selectedSec = 1;
    %let i = 0;
    %estSection(&casTrainingTable.,mycas.tempSection,
                &selectedSec.,&i.,&T.,
                &evalTable.,&casModelTable.);
proc sort data=&evalTable.&i.; by descending logLikelihood&i.; run;

%do i = 1 %to &nClusters.;
    data _null_;
        set &evalTable.0;
        pos = floor(1+(&nSections.-1)*(&i.-1)/(&nClusters.-1)+0.5);
        if(_N_=pos) then
            call symput('selectedSec',trim(left(put(sec,12.))));
    run;
    %estSection(&casTrainingTable.,mycas.tempSection,
                &selectedSec.,&i.,&T.,
                &evalTable.,&casModelTable.);

```

```

%end;

%assignSeriesToClusters(&evalTable., &nClusters.);
%mend;

```

The following macro repeatedly moves the time series between clusters until no movement is necessary:

```

* estimate HMM for cluster i;
* each section has T observations;
%macro estCluster(casClusteredTable, casTrainingTable, i, T,
                  evalTable, casModelTable);
    data &casClusteredTable.&i.;
        set &casClusteredTable.;
        if(pCluster=&i.);
    run;

    proc hmm data=&casClusteredTable.&i.;
        id section=sec time=t;
        model y / method=ml type=gaussian nstate=&k.;
        optimize ALGORITHM=interiorpoint printlevel=3 printIterFreq=1;
        score outmodel=&casModelTable.&i.;
    run;

    proc hmm data=&casTrainingTable.;
        evaluate out=mycas.tempEval;
        score inmodel=&casModelTable.&i.;
    run;

    data &evalTable.&i.;
        set mycas.tempEval;
        logLikelihood&i. = logLikelihood;
        if(t=&T.) then output;
        keep sec logLikelihood&i.;
    run;

    proc sort data=&evalTable.&i.; by sec; run;
%mend;

* moves time series between clusters based on their likelihoods;
* each section has T observations;
* (1) &casModelTable.&i. has the model estimates for cluster i,
* and (2) &evalTable. has the clustering result;
%macro ctsClustering(trainingTable, casTrainingTable, nClusters, T,
                    casClusteredTable, evalTable, casModelTable);
    * save last clustering result;
    data prev&evalTable.; set &evalTable.; run;

    * clustering data;
    data &casClusteredTable.;
        merge &trainingTable. (in=a) &evalTable. (in=b);
        by sec;
    run;

    %do i = 1 %to &nClusters.;

```

```

        %estCluster(&casClusteredTable., &casTrainingTable., &i., &T.,
                    &evalTable., &casModelTable.);
    %end;

    %assignSeriesToClusters(&evalTable., &nClusters.);

    * compare current and previous clustering results;
    proc compare data=&evalTable. compare=prev&evalTable.; run;
    %let converged=%eval(&sysinfo.=0);
    %let nIter=%eval(&nIter.+1);
    %let done=%eval(&converged. or &nIter.>=&maxIter.);

    %put nIter      = &nIter.;
    %put converged = &converged.;
    %put done      = &done.;
%mend;

* cluster time series;
* each section has T observations;
%macro ctsLoopClustering(trainingTable, casTrainingTable, nClusters, T,
                        casClusteredTable, evalTable, casModelTable);
    %let done = 0;      * terminate clustering;
    %do %while(&done.=0);
        %ctsClustering(&trainingTable., &casTrainingTable., &nClusters., &T.,
                        &casClusteredTable., &evalTable., &casModelTable.);
    %end;
%mend;

```

Because the DGP is known, you can use the following macro to evaluate the performance of the clustering algorithm:

```

* compare the clustering results with the true clusters in the DGP;
* it is applicable only when the DGP is available;
%macro evaluateClusteringResults(dgpTable, evalTable, nSections, T);
    data true&evalTable.;
        set &dgpTable.;
        if(t=&T.);
            keep sec cluster;
        run;

    data ctsCheck;
        merge true&evalTable. (in=a) &evalTable. (in=b);
        by sec;
        if(cluster=pCluster) then do; correct=1; end;
        else do; correct=0; end;
        if(a=b);
    run;

    data ctsAccuracy;
        set ctsCheck;
        retain count 0 correctCount 0;
        count = count + 1;
        correctCount = correctCount + correct;
        if(count>&nSections.-0.5) then do;

```

```

        accuracy = correctCount / count;
        if (accuracy < 0.5) then accuracy = 1 - accuracy;
        nIterations = &nIter.;
        converged = &converged.;
        output;
    end;
    keep nIterations converged accuracy;
run;

proc print data = ctsAccuracy noobs;
    var nIterations converged accuracy;
    format accuracy 6.4;
run;
%mend;

```

As data become easy to access, the number of sections might become large. In this example, the number of sections is set to 10,000. As a start, the sample size for each time series is set to 200, which is the same as the sample size in Smyth (1997) and Oates, Firoiu, and Cohen (1999). In fact, the time series might have different lengths; if so, you should replace the log likelihood with the log likelihood per observation when you compare the time series.

The following macro calls simulate 10,000-section time series, apply the clustering algorithm, and then evaluate its performance:

```

%let N = 10000;
%let T = 200;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mycas.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mycas.ctsTrain, &nClusters., &N., &T.,
    evalCluster, mycas.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;     * number of iterations;
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain, mycas.ctsTrain, &nClusters., &T.,
    mycas.ctsTrainClustered, evalCluster, mycas.modelCluster);
%evaluateClusteringResults(ctsDGP, evalCluster, &N., &T.);

```

As shown in [Output 13.2.1](#), the accuracy of the clustering algorithm is very high.

Output 13.2.1 Clustering Accuracy ($N = 10,000$, $T = 200$)

Clustering Time Series

nIterations	converged	accuracy
5	1	0.9868

The following statements display the parameter estimates for cluster 1 and cluster 2 in [Output 13.2.2](#) and [Output 13.2.3](#), respectively:

```

* print parameter estimates for cluster 1;
proc hmm data=mycas.ctsTrain;
    score inmodel=mycas.modelCluster1;
run;

```

```

* print parameter estimates for cluster 2;
proc hmm data=mycas.ctsTrain;
  score inmodel=mycas.modelCluster2;
run;

```

As shown in [Output 13.2.2](#) and [Output 13.2.3](#), the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.

Output 13.2.2 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.598877	0.000977	612.98	<.0001
TPM1_2	0.401123	0.000977	410.57	<.0001
TPM2_1	0.399884	0.000979	408.35	<.0001
TPM2_2	0.600116	0.000979	612.82	<.0001
MU1_1	-0.000368	0.002626	-0.14	0.8885
MU2_1	2.997355	0.002636	1137.11	<.0001
SIGMA1_1_1	0.994514	0.003457	287.69	<.0001
SIGMA2_1_1	1.002381	0.003487	287.43	<.0001

Output 13.2.3 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 200$)

Clustering Time Series

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.399589	0.001164	343.20	<.0001
TPM1_2	0.600411	0.001164	515.69	<.0001
TPM2_1	0.600027	0.001161	516.61	<.0001
TPM2_2	0.399973	0.001161	344.37	<.0001
MU1_1	-0.000618	0.002597	-0.24	0.8119
MU2_1	2.999476	0.002603	1152.48	<.0001
SIGMA1_1_1	0.995491	0.003433	290.00	<.0001
SIGMA2_1_1	0.999145	0.003447	289.87	<.0001

In the second scenario, the sample size of one time series is increased to 400:

```

%let N = 10000;
%let T = 400;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mycas.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;          * number of states in the HMM for each cluster;
%ctsInitClusters(mycas.ctsTrain, &nClusters., &N., &T.,
  evalCluster, mycas.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;    * number of iterations;

```

```
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain, mycas.ctsTrain, &nClusters., &T.,
    mycas.ctsTrainClustered, evalCluster, mycas.modelCluster);
%evaluateClusteringResults(ctsDGP, evalCluster, &N., &T.);
```

As the sample size increases, more information becomes available for each time series. As expected, the accuracy increases (to almost 100% in this case). [Output 13.2.4](#) shows the accuracy.

Output 13.2.4 Clustering Accuracy ($N = 10,000$, $T = 400$)

Clustering Time Series

nIterations	converged	accuracy
4	1	0.9989

The following statements print the parameter estimates for both clusters, which also become closer to the true parameter values, as shown in [Output 13.2.5](#) and [Output 13.2.6](#):

```
* print parameter estimates for cluster 1;
proc hmm data=mycas.ctsTrain;
    score inmodel=mycas.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mycas.ctsTrain;
    score inmodel=mycas.modelCluster2;
run;
```

Output 13.2.5 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 400$)

Clustering Time Series

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.599184	0.000689	869.96	<.0001
TPM1_2	0.400816	0.000689	581.95	<.0001
TPM2_1	0.400637	0.000690	580.78	<.0001
TPM2_2	0.599363	0.000690	868.85	<.0001
MU1_1	-0.000181	0.001851	-0.10	0.9219
MU2_1	2.999973	0.001855	1617.37	<.0001
SIGMA1_1_1	0.996500	0.002443	407.90	<.0001
SIGMA2_1_1	0.999350	0.002452	407.56	<.0001

Output 13.2.6 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 400$)**Clustering Time Series**

Parameter	Parameter Estimates			
	Estimate	Standard Error	t Value	Pr > t
TPM1_1	0.399302	0.000827	482.79	<.0001
TPM1_2	0.600698	0.000827	726.30	<.0001
TPM2_1	0.600445	0.000827	725.85	<.0001
TPM2_2	0.399555	0.000827	483.00	<.0001
MU1_1	2.998405	0.001852	1619.15	<.0001
MU2_1	-0.000477	0.001852	-0.26	0.7966
SIGMA1_1_1	1.001306	0.002449	408.85	<.0001
SIGMA2_1_1	1.001528	0.002449	408.92	<.0001

In the last scenario, the sample size for each time series decreases to 20 (see the following statements), which is a very small number. This case is common when the sampling frequency cannot be high (for example, yearly tax return data from millions of families). The information in one time series is very limited; this might even hinder the estimation of the hidden Markov models. However, as shown in this example, by clustering the time series, you condense the common information in time series, and then the true DGP is recovered.

```
%let N = 10000;
%let T = 20;
%createClustersTable(&N., &T., ctsDGP, ctsTrain, mycas.ctsTrain);
%let nClusters = 2; * number of clusters;
%let k=2;           * number of states in the HMM for each cluster;
%ctsInitClusters(mycas.ctsTrain, &nClusters., &N., &T.,
    evalCluster, mycas.modelCluster);
%let maxIter = 100; * maximum number of iterations;
%let nIter = 0;    * number of iterations;
%let converged = 0; * converge status;
%ctsLoopClustering(ctsTrain, mycas.ctsTrain, &nClusters., &T.,
    mycas.ctsTrainClustered, evalCluster, mycas.modelCluster);
%evaluateClusteringResults(ctsDGP, evalCluster, &N., &T.);
```

As expected and as shown in [Output 13.2.7](#), the algorithm needs many more iterations to converge, and the accuracy of the clustering algorithm is not high.

Output 13.2.7 Clustering Accuracy ($N = 10,000$, $T = 20$)**Clustering Time Series**

nIterations	converged	accuracy
26	1	0.7552

The following statements print the parameter estimates for both clusters, which are shown in [Output 13.2.8](#) and [Output 13.2.9](#). Although each time series might contain little information, the parameter estimates of the Gaussian HMMs for the two clusters are very close to the true parameters in the DGP.


```

* print parameter estimates for cluster 1;
proc hmm data=mycas.ctsTrain;
  score inmodel=mycas.modelCluster1;
run;

* print parameter estimates for cluster 2;
proc hmm data=mycas.ctsTrain;
  score inmodel=mycas.modelCluster2;
run;

```

Output 13.2.8 Parameter Estimates for Cluster 1 ($N = 10,000$, $T = 20$)**Clustering Time Series**

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.357167	0.003560	100.32	<.0001
TPM1_2	0.642833	0.003560	180.56	<.0001
TPM2_1	0.638448	0.003510	181.91	<.0001
TPM2_2	0.361552	0.003510	103.02	<.0001
MU1_1	-0.002996	0.007409	-0.40	0.6860
MU2_1	2.989537	0.007619	392.36	<.0001
SIGMA1_1_1	0.965620	0.009808	98.45	<.0001
SIGMA2_1_1	1.011633	0.010366	97.59	<.0001

Output 13.2.9 Parameter Estimates for Cluster 2 ($N = 10,000$, $T = 20$)**Clustering Time Series**

Parameter Estimates				
Parameter	Estimate	Standard	t Value	Pr > t
		Error		
TPM1_1	0.638344	0.002875	222.00	<.0001
TPM1_2	0.361656	0.002875	125.78	<.0001
TPM2_1	0.365361	0.002894	126.23	<.0001
TPM2_2	0.634639	0.002894	219.26	<.0001
MU1_1	3.010552	0.007647	393.69	<.0001
MU2_1	0.011987	0.007950	1.51	0.1316
SIGMA1_1_1	0.981346	0.010152	96.66	<.0001
SIGMA2_1_1	1.029845	0.010807	95.30	<.0001

Example 13.3: Analysis of the Business Cycle by Using the Regime-Switching Mean-Adjusted Autoregression Model

This example revisits the model in Hamilton (1989) to illustrate how the regime-switching mean-adjusted autoregression model is used to estimate the business cycle with quarterly US real gross national product (GNP) data from 1951 to 1984.

The following statements import the data and calculate the GNP growth rate (in percentage) to which the model applies:

```

title 'Analysis of the Business Cycle';
data gnpHamilton;
    input date: anydtdte. gnp;
    format date yyq6.;
    dgnp = 100*(log(gnp)-log(lag(gnp)));
    if cmiss(of _all_) then delete;
    label date="Date" gnp="GNP" dgnp="Growth Rate of GNP";
datalines;
1951q1  1286.6
1951q2  1320.4
1951q3  1349.8

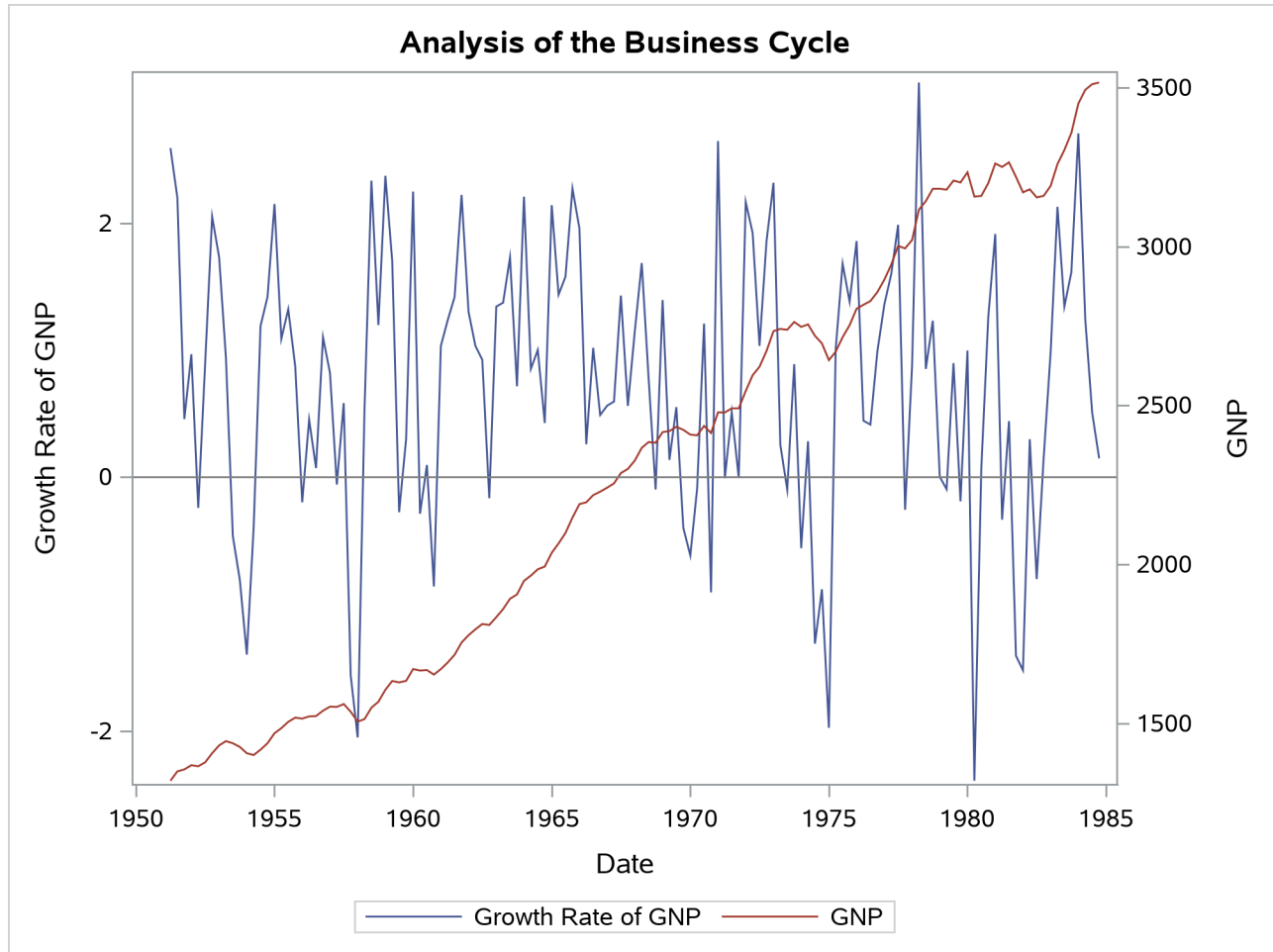
    ... more lines ...

1982q4  3159.3
1983q1  3190.6
1983q2  3259.3
1983q3  3303.4
1983q4  3357.2
1984q1  3449.4
1984q2  3492.6
1984q3  3510.4
1984q4  3515.6
;
```

The following statements plot the data in [Output 13.3.1](#):

```

proc sgplot data=gnpHamilton;
    series x=date y=dgnp;
    series x=date y=gnp / y2axis;
    refline 0 / axis=y lineattrs=(color=gray thickness=1);
run;
```

Output 13.3.1 GNP and Growth Rate of GNP

Hamilton's model is a bivariate regime-switching mean-adjusted AR(4) model which can be described as

$$y_t = c^{(s_t)} + \sum_{\tau=1}^4 \alpha_{\tau}^{(s_t)} (y_{t-\tau} - c^{(s_{t-\tau})}) + \varepsilon_t, \quad \varepsilon_t \sim N(0, \Sigma^{(s_t)})$$

where $s_t \in \{1, 2\}$ is the unobserved state.

In Hamilton (1989), the autoregressive parameters α 's and variance parameters Σ 's are assumed to be state-independent, which means $\alpha_{\tau}^{(1)} = \alpha_{\tau}^{(2)}$, $\tau = 1, \dots, 4$, and $\Sigma^{(1)} = \Sigma^{(2)}$. The following statements re-create Hamilton's result by setting the initial parameter values as the parameter estimates in Table 1 in Hamilton (1989) and also setting the MAXITER= option in the OPTIMIZE statement to zero. The FILTER statement conducts the probabilities of state on the basis of data available up to the current period. The DECODE statement outputs the best possible path of states on the basis of all observations. The SMOOTH statement finds the probabilities of states on the basis of all observations.

```

data mycas.gnpHamilton; set gnpHamilton; run;
proc hmm data=mycas.gnpHamilton labels switch=(sort=desc(const));
  id time=date;
  model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
              stateIndependent=(ar cov);
  optimize maxiter=0;
  initial tpm={0.9049 0.0951, 0.2450 0.7550},
             ar={0.014 -0.058 -0.247 -0.213, 0.014 -0.058 -0.247 -0.213},
             const={1.1643, -0.3577},
             cov={0.5914, 0.5914};
  filter out=mycas.filter;
  decode out=mycas.decode;
  smooth out=mycas.smooth;
run;

```

The number of observations and model information are shown in [Output 13.3.2](#).

Output 13.3.2 Number of Observations and Model Information
Analysis of the Business Cycle

Observation Information	
Number of Observations	135
Number of Missing Observations	0

Model Information	
Type of Model	Regime-Switching Mean-adjusted Autoregression Model
Stationary	Yes
Number of States	2
Number of Dependent Variables	1
Number of Exogenous Variables	0

The fit statistics are shown in [Output 13.3.3](#).

Output 13.3.3 Model Information

Fit Statistics	
Log Likelihood	-183.1751626
AIC	384.35032521
AICC	385.79032521
BIC	410.49779822
HQC	394.97592522

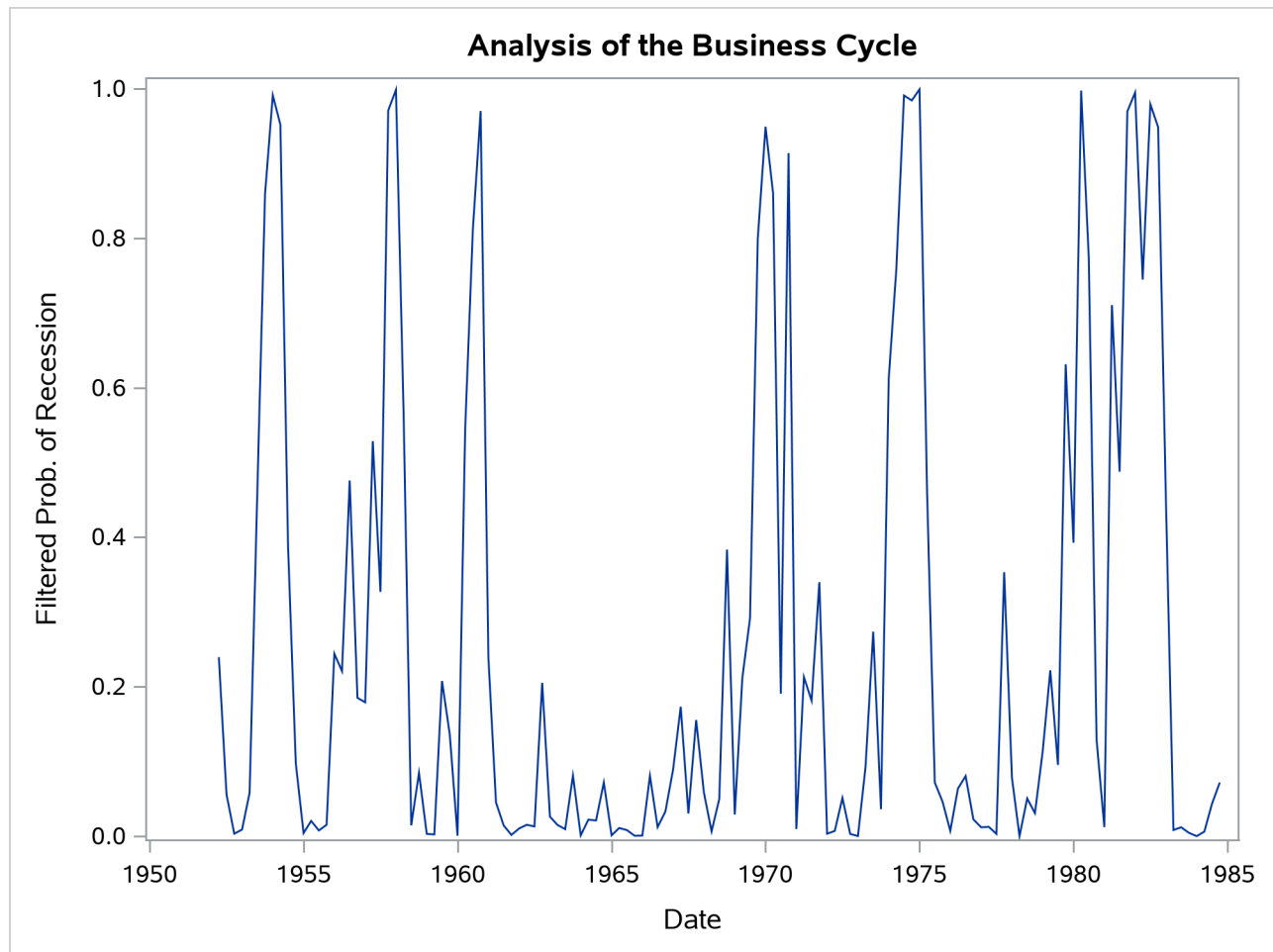
The following statements plot the filtered probabilities of state 2 in [Output 13.3.4](#), which is comparable to Figure 1 in Hamilton (1989). State 2 corresponds to recession because the intercept (and then the expected mean) in state 2 is negative.

```

data filter; set mycas.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
  series x=date y=state2;
  yaxis label="Filtered Prob. of Recession";
run;

```

Output 13.3.4 Filter Plot of Economic Recession



In order to show how states correspond to the signs of GNP growth rate, the following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak. As shown in [Output 13.3.5](#), the recession includes most periods that have a negative GNP growth rate.

```

data plot2state;
  merge mycas.decode mycas.gnpHamilton;
  if state=2 then do; recession=1; boom=0; end;
  else if state=1 then do; recession=0; boom=1; end;
  rec = recession*(-3);
  boo = boom*3;
  by date;
  if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;

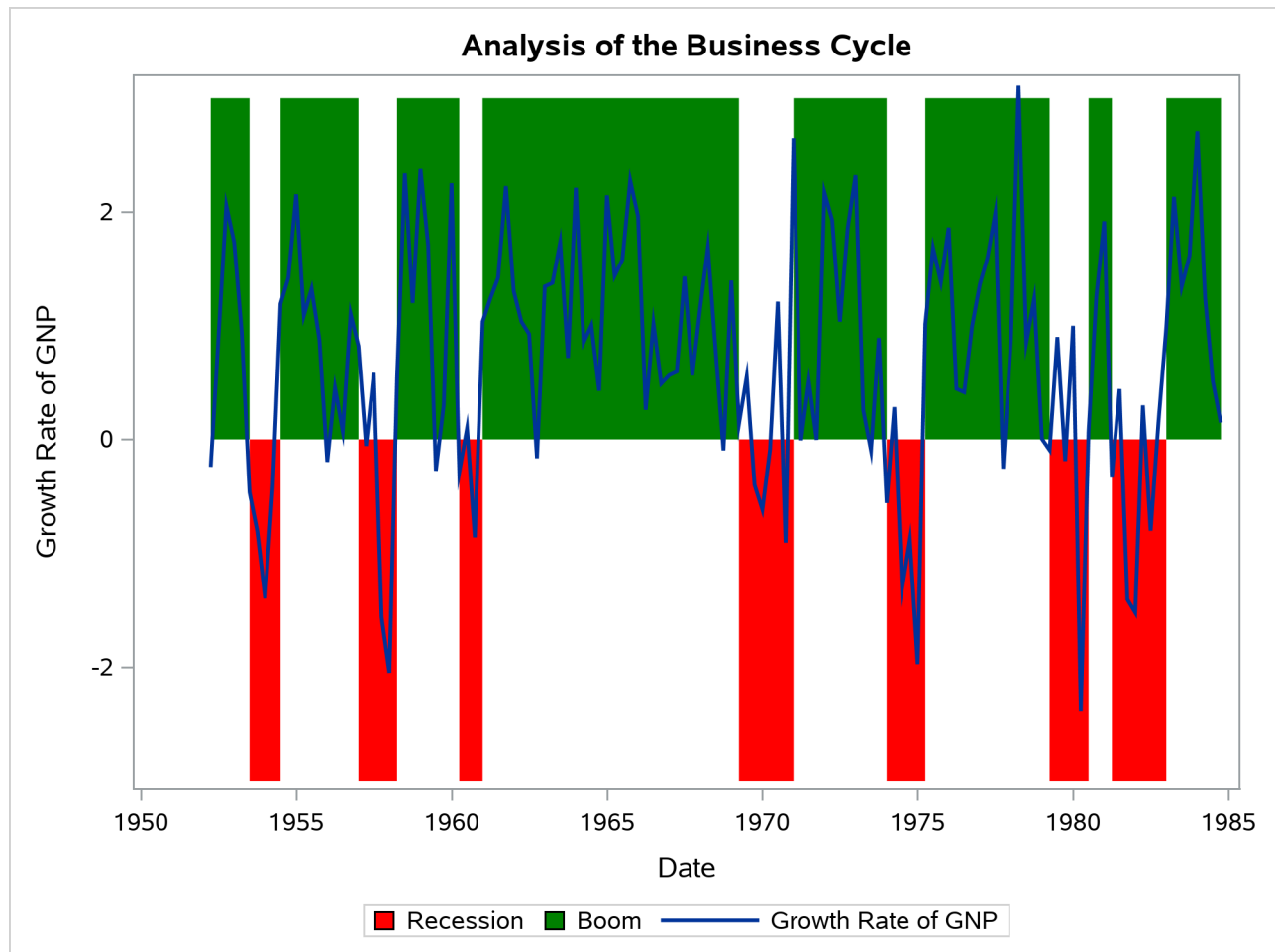
```

```

band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                fillattrs=(color=red);
band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                fillattrs=(color=green);
series x=date y=dgnp / lineattrs=(thickness=2);
run;

```

Output 13.3.5 Decoded Plot of Business Cycle



Hamilton (1989) also compares his choice of business-cycle dates with the historical business-cycle dates that are reported from the National Bureau of Economic Research (NBER). The following data step inputs the business-cycle dates from NBER:

```

data mycas.nber;
  input date: anydtdte. recessionNBER;
  format date yyq6.;
  label date="Date" recessionNBER="Recession Reported from NBER";
datalines;
1951q1  0
1951q2  0
1951q3  0
1951q4  0
1952q1  0

```

```

1952q2  0

... more lines ...

1982q4  1
1983q1  0
1983q2  0
1983q3  0
1983q4  0
1984q1  0
1984q2  0
1984q3  0
1984q4  0
;

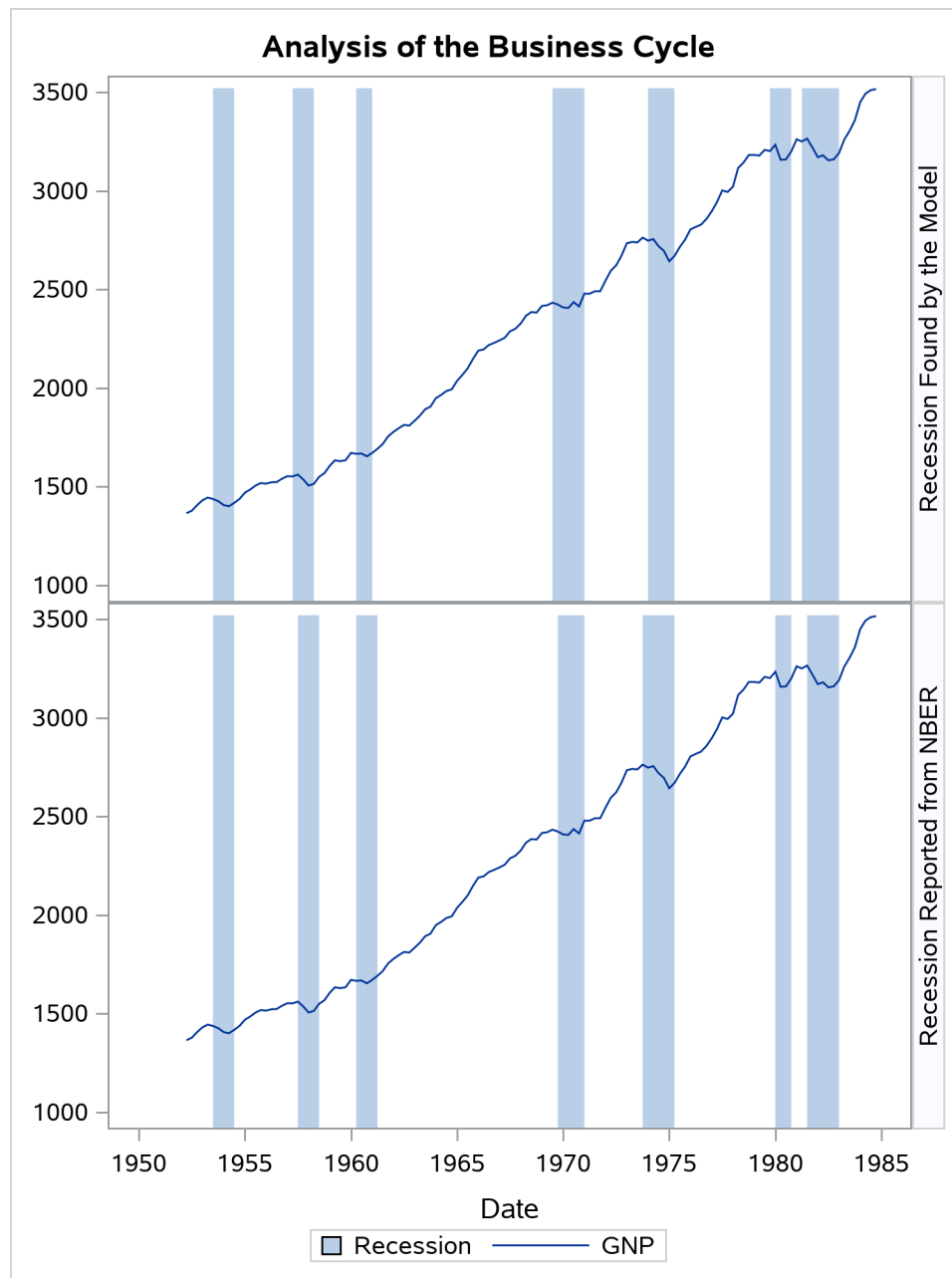
```

The following statements plot together the recession dates that are found by the model and those that are reported from NBER. As shown in [Output 13.3.6](#), the recession dates that are found by Hamilton's model (the upper part) is extremely close to the dates that are reported from NBER (the lower part).

```

data gnpsmooth;
  merge mycas.smooth mycas.nber mycas.gnpHamilton;
  if state2>=0.5 then recession=1; else recession=0;
  by date;
  gnp2=gnp;
  if cmiss(of _all_) then delete;
  label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth out=gnpsmooth_t;
  by date;
  var recessionNBER recession;
run;
proc transpose data=gnpsmooth out=gnpsmooth_t2;
  by date;
  var gnp gnp2;
run;
data gnpplot;
  merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 coll=recession))
        gnpsmooth_t2(rename=(_name_=name2 _label_=label2 coll=gnp));
  by date;
  rec=recession*3520;
run;
proc sgpanel data=gnpplot;
  panelby label1 / layout=rowlattice uniscale=column novarname;
  band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
  series x=date y=gnp / legendlabel='GNP';
  rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;

```

Output 13.3.6 Smoothed Plot of Economic Recession

The following statements use PROC HMM to estimate Hamilton's model while using the STATEINDEPENDENT=(AR COV) option in the MODEL statement so that the autoregression parameter and the variance are fixed across the states:

```
proc hmm data=mycas.gnpHamilton labels switch=(sort=desc(const));
  id time=date;
  model dgnp / armean=adjusted type=ar ylag=4 method=ml nstate=2
    stateIndependent=(ar cov);
  optimize printIterFreq=1 printlevel=3 ;
  filter out=mycas.filter;
```



```

decode out=mycas.decode;
smooth out=mycas.smooth;
run;

```

The parameter estimates are shown in [Output 13.3.7](#), and the fit statistics are shown in [Output 13.3.8](#). The log-likelihood value and the information criterion in [Output 13.3.8](#) show that this solution is better than the one in [Output 13.3.3](#) when MAXITER=0.

Output 13.3.7 Parameter Estimates

Analysis of the Business Cycle

Parameter Estimates				
Parameter	Standard		t Value	Pr > t
	Estimate	Error		
TPM1_1	0.904414	0.037629	24.03	<.0001
TPM1_2	0.095586	0.037629	2.54	0.0123
TPM2_1	0.254962	0.099402	2.56	0.0115
TPM2_2	0.745038	0.099402	7.50	<.0001
CONST1_1	1.160773	0.077552	14.97	<.0001
CONST2_1	-0.392035	0.275951	-1.42	0.1580
AR1_1_1_1	0.016338	0.121529	0.13	0.8933
AR1_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR1_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR1_4_1_1	-0.184025	0.111810	-1.65	0.1024
AR2_1_1_1	0.016338	0.121529	0.13	0.8933
AR2_2_1_1	-0.088765	0.152015	-0.58	0.5604
AR2_3_1_1	-0.219363	0.114233	-1.92	0.0572
AR2_4_1_1	-0.184025	0.111810	-1.65	0.1024
COV1_1_1	0.609788	0.108633	5.61	<.0001
COV2_1_1	0.609788	0.108633	5.61	<.0001

Output 13.3.8 Fit Statistics

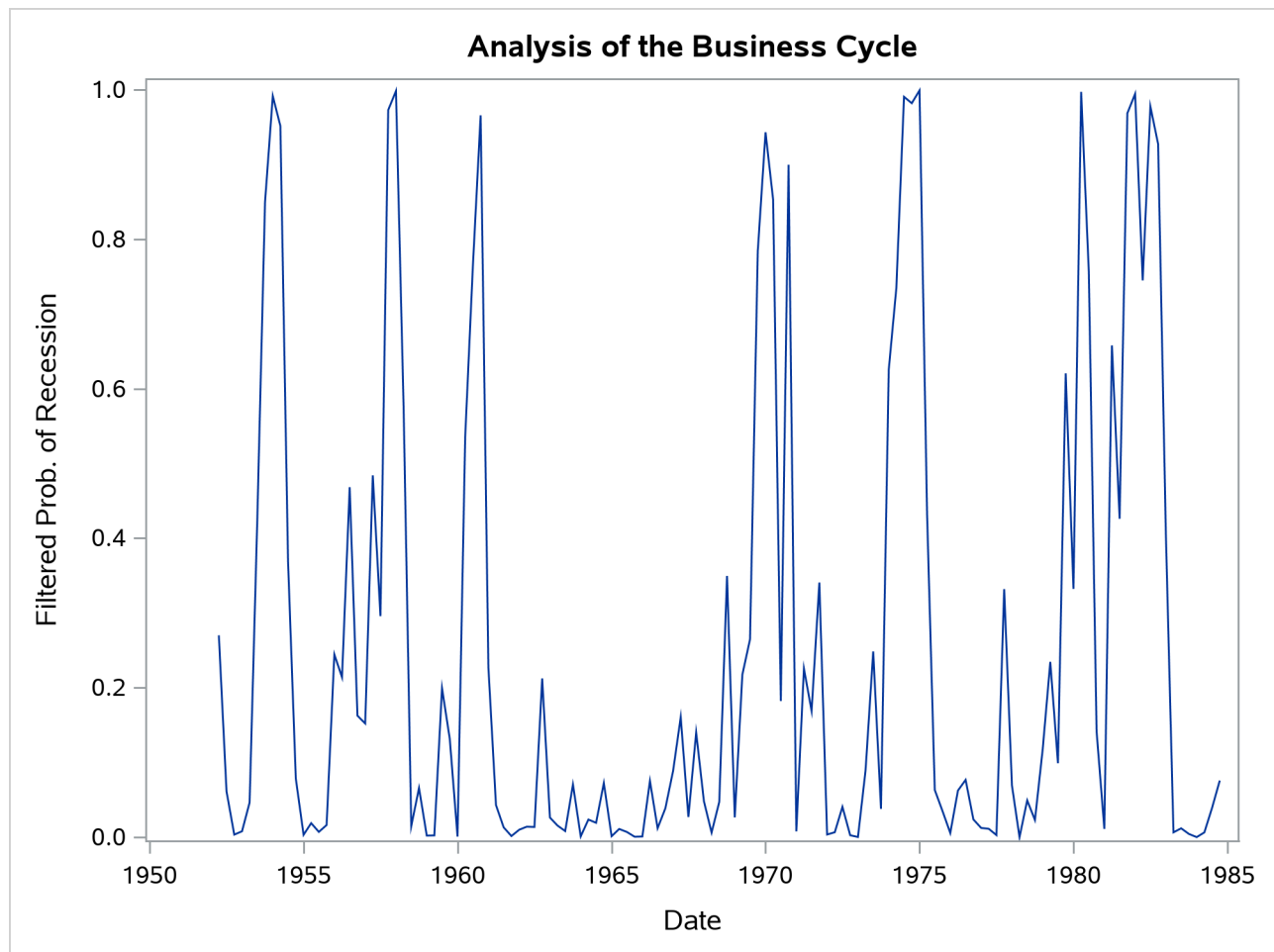
Fit Statistics	
Log Likelihood	-183.0222956
AIC	384.04459127
AICC	385.48459127
BIC	410.19206427
HQC	394.67019127

The following statements plot the filtered probabilities of state 2 (recession). [Output 13.3.9](#) shows the results.

```

data filter; set mycas.filter; run;
proc sort data=filter; by date; run;
proc sgplot data=filter;
  series x=date y=state2;
  yaxis label="Filtered Prob. of Recession";
run;

```

Output 13.3.9 Filter Plot of Economic Recession

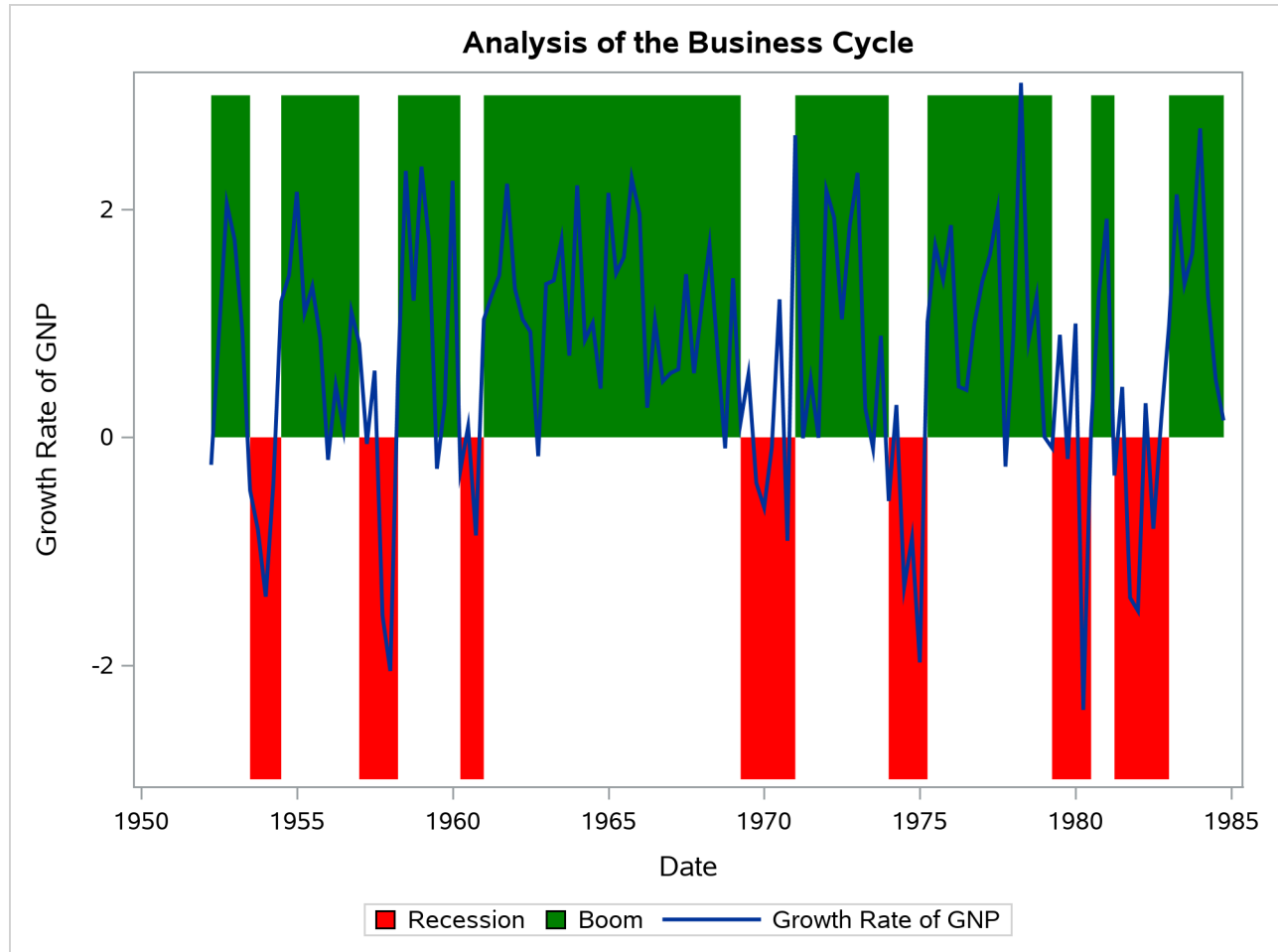
The following statements overlay the decoded states on the GNP growth rate. Recession is from peak to trough, and boom is from trough to peak.

```
data plot2state;
  merge mycas.decode mycas.gnpHamilton;
  if state=2 then do; recession=1; boom=0; end;
  else if state=1 then do; recession=0; boom=1; end;
  rec = recession*(-3);
  boo = boom*3;
  by date;
  if cmiss(of _all_) then delete;
run;

proc sgplot data=plot2state;
  band x=date upper=0 lower=rec / legendlabel='Recession' type=step
                                     fillattrs=(color=red);
  band x=date upper=boo lower=0 / legendlabel='Boom' type=step
                                     fillattrs=(color=green);
  series x=date y=dgnp / lineattrs=(thickness=2);
run;
```

Output 13.3.10 shows that more states switch than are shown in Hamilton's decoded plot. The reason is that the estimates indicate less persistence of each state.

Output 13.3.10 Decoded Plot of Business Cycle



The following statements plot the recession dates that are found by the model and those that are reported from NBER:

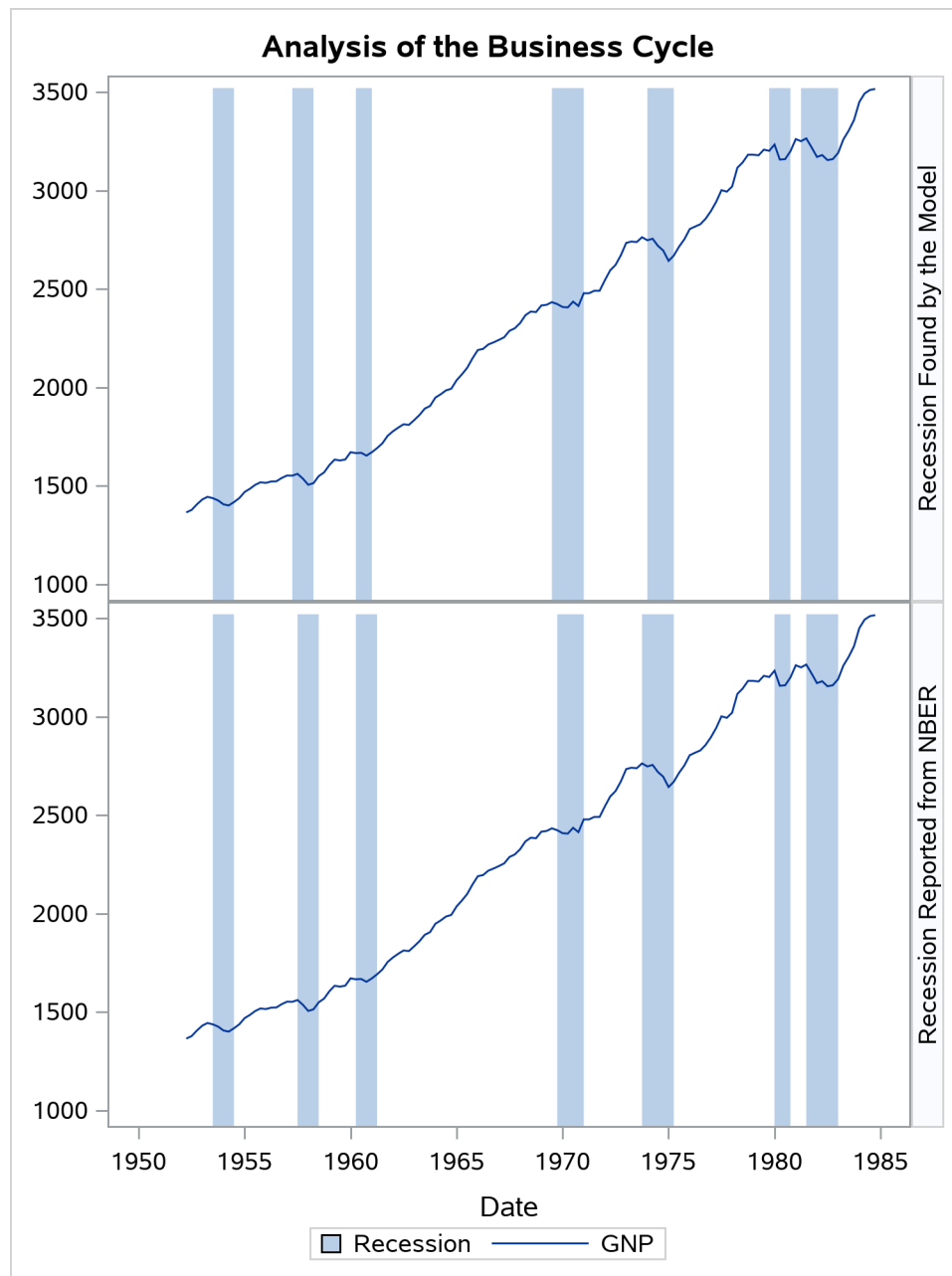
```
data gnpsmooth;
  merge mycas.smooth mycas.nber mycas.gnpHamilton;
  if state2>=0.5 then recession=1; else recession=0;
  by date;
  gnp2=gnp;
  if cmiss(of _all_) then delete;
  label recession="Recession Found by the Model" gnp2="GNP";
run;
proc transpose data=gnpsmooth out=gnpsmooth_t;
  by date;
  var recessionNBER recession;
run;
proc transpose data=gnpsmooth out=gnpsmooth_t2;
  by date;
```

```

    var gnp gnp2;
run;
data gnpplot;
    merge gnpsmooth_t(rename=(_name_=name1 _label_=label1 coll=recession))
          gnpsmooth_t2(rename=(_name_=name2 _label_=label2 coll=gnp));
    by date;
    rec=recession*3520;
run;
proc sgpanel data=gnpplot;
    panelby label1 / layout=rowlattice uniscale=column novarname;
    band x=date upper=rec lower=0 / legendlabel='Recession' type=step;
    series x=date y=gnp / legendlabel='GNP';
    rowaxis display=(nolabel) values=(1000 to 3520 by 500);
run;

```

As shown in [Output 13.3.11](#), the recession dates that are found by the model (the upper part) are extremely close to the dates that are reported from NBER (the lower part).

Output 13.3.11 Smoothed Plot of Economic Recession

References

- Cappé, O., Moulines, E., and Rydén, T. (2010). *Inference in Hidden Markov Models*. New York: Springer.
- Center for Research in Security Prices (2018). "CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index." CRSP, University of Chicago Booth School of Business.
- Frühwirth-Schnatter, S. (2006). *Finite Mixture and Markov Switching Models*. New York: Springer.

- Ghassempour, S., Girosi, F., and Maeder, A. (2014). “Clustering Multivariate Time Series Using Hidden Markov Models.” *International Journal of Environmental Research and Public Health* 11:2741–2763. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3968966/>.
- Hamilton, J. D. (1989). “A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle.” *Econometrica* 57:357–384.
- Krolzig, H.-M. (1997). *Markov-Switching Vector Autoregressions: Modelling, Statistical Inference, and Application to Business Cycle Analysis*. Berlin: Springer.
- Kuester, K., Mittnik, S., and Paolella, M. S. (2006). “Value-at-Risk Prediction: A Comparison of Alternative Strategies.” *Journal of Financial Econometrics* 4:53–89.
- Liao, T. W. (2005). “Clustering of Time Series Data—a Survey.” *Pattern Recognition* 38:1857–1874.
- Oates, T., Firoiu, L., and Cohen, P. R. (1999). “Clustering Time Series with Hidden Markov Models and Dynamic Time Warping.” In *Proceedings of the IJCAI-99 Workshop on Neural, Symbolic, and Reinforcement Learning Methods for Sequence Learning*, 17–21.
- Rabiner, L. R. (1989). “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.” *Proceedings of the IEEE* 77:257–286.
- Rani, S., and Sikka, G. (2012). “Recent Techniques of Clustering of Time Series Data: A Survey.” *International Journal of Computer Applications* 52:1–9.
- Smyth, P. (1997). “Clustering Sequences with Hidden Markov Models.” In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, edited by M. C. Mozer, T. Petsche, and M. I. Jordan, 648–654. Cambridge, MA: MIT Press.
- Sung, H. G. (2004). “Gaussian Mixture Regression and Classification.” Ph.D. diss., Rice University.

Chapter 14

The SASEMOOD Interface Engine

Contents

Overview: SASEMOOD Interface Engine	782
Using CAS Sessions and CAS Engine Librefs	782
Getting Started: SASEMOOD Interface Engine	783
Syntax: SASEMOOD Interface Engine	785
The LIBNAME <i>libref</i> SASEMOOD Statement	785
Details: SASEMOOD Interface Engine	791
SAS Output Data Set	791
Examples: SASEMOOD Interface Engine	792
Example 14.1: Retrieving Consumer Price Index Data from Moody’s Analytics Data Buffet	792
Example 14.2: Retrieving Quarterly Gross Domestic Product Data by Industry from the Bureau of Economic Analysis (BEA)	793
Example 14.3: Retrieving Weekly Unemployment Insurance Claim Statistics for the United States	794
Example 14.4: Retrieving Daily Data from the S&P 500	795
Example 14.5: Retrieving GDP by Industry: Chain-Type Price Index for Value Added for the United States	797
Example 14.6: Retrieving Unemployment Rate Data for the United States	798
Example 14.7: Retrieving Employment Data: Total, Nonfarm, for the United States	800
Example 14.8: Retrieving Unemployment Data and Household Debt to Personal Disposable Income Ratio for the United States	802
Example 14.9: Retrieving National Income and Product Account Data	804
References	805

Overview: SASEMOOD Interface Engine

The SASEMOOD interface engine enables SAS users to retrieve time series data from the Moody's Analytics Data Buffet (Economy.com) website. This website offers access to over 600 sources of global historical statistical data at the regional, national, and subnational level, in addition to 40 forecast database offerings. Over 220 million time series are updated promptly after release, on topics such as housing, labor, demographics, finance, industry, housing prices, and consumer credit performance forecasts. Time series are offered in yearly, semiannual, quarterly, monthly, semimonthly, biweekly, weekly, workday (business), daily, and indexed frequencies.

The SASEMOOD interface engine uses the LIBNAME statement to enable you to download Moody's online economic data from the website at the following root URL:

<https://api.economy.com/data/v1/>

It also enables you to specify which time series you want to retrieve, by using the mnemonic code that corresponds to the data set that you select. You choose the time range of the retrieved data by specifying a start date and an end date. You can then use the SAS DATA step to perform further subsetting, retrieve the data, and store the resulting time series in a SAS data set and in a CAS table (if desired).

You can view the catalog of available Moody's databases on the web page at the following URL:

<https://www.economy.com/databuffet/>

The SASEMOOD interface engine supports Linux X64 (LAX) and Windows.

Although the SASEMOOD engine uses Moody's Analytics Data Buffet online API, it is not endorsed or certified by Moody's Analytics. By using the SASEMOOD interface engine, you are agreeing to comply with the terms of use, which are described on the web page at the following URL:

<https://www.economy.com/about/terms-of-use>

For more information see the Moody's Analytics (2019) Data Buffet API User Guide.

To get started using the SASEMOOD engine, follow the steps in the section “[Getting Started: SASEMOOD Interface Engine](#)” on page 783, which enable you to view the GDP\$ (Gross Domestic Product in the National Income and Product Accounts) database to retrieve the GDP\$ time series. The sample SAS code for accessing the Moody's GDP\$ time series appears at the end of the section.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:


```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

For more information about how to store your Moody’s data in a CAS table, see [Example 14.9](#) in this chapter.

Getting Started: SASEMOOD Interface Engine

You need a subscription to use Moody’s Analytics Data Buffet web service, and you must have the proper credentials to access this service. For more information, see “Using Data Buffet: Data Buffet API v1.0” (Thorne 2019b). After you log on, you will see your access key and encryption key in your account information at the following URL:

<https://www.economy.com/myeconomy/api-key-info>

For more information about getting started using Data Buffet, see “Using Data Buffet: Which Series Can You Download?” (Thorne 2019f).

The web page at the following URL describes how to use the Moody’s Analytics Data Buffet API (Moody’s Analytics 2019):

<https://github.com/moodyanalytics/databuffet.api.codesamples/blob/master/databuffet-api-userguide.pdf>

The statements at the end of this section enable you to access the mnemonic GDP\$.IUSA, which is described as “Gross domestic product, (Mil. Ch 2012 USD, SAAR) for United States.”

The MNEMONIC= option specifies the time series mnemonic code(s) that you need in order to retrieve the data from Moody’s Analytics Data Buffet. GDP\$.IUSA is the time series mnemonic code in the sample code shown at the end of this section.

The OUT= option specifies both the name of the resulting JSON file(s) and the name of the SAS data set.

The range of the retrieved data is determined by the START= and END= options. You specify the start and end dates in ‘YYYY-MM-DD’ format.

The JSON data that the Moody’s Analytics Data Buffet website returns are placed in a file that is named by the OUT= option—in this case, *GDP1.json*. **NOTE:** The SASEMOOD engine appends a numeral to the JSON file name, and the file extension (.json) is excluded from the file name that appears in the OUT= option. When the SET statement is executed, the JSON data are read (and merged) into a SAS data set named GDP.sas7bdat, which resides in the location that is specified inside the string enclosed in double quotation marks in the LIBNAME statement.

The result is named in the DATA step in the SET statement and is shown in [Output 14.1](#).

For another example that uses more LIBNAME statement options, see the section “[Examples: SASEMOOD Interface Engine](#)” on page 792.

```
options validvarname=any;

title 'Retrieve Gross Domestic Product Data for United States';

libname mylib "/sasusr/mood/";

title1 "National Income and Product Accounts Databank from Moody's Analytics Data Buffet";
title2 "Request US Gross Domestic Product";
libname moody sasemood "%sysget(MOOD_DATA)"
    mnemonic="GDP$.IUSA"          /* steps 2 and 3 */
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /* access key */
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' /* encryption key */
    freq=quarterly
    conv=2
    outjson=GDP
    start='2017-01-01'
    end='2017-12-01'
    ;

data mylib.myGDP;
    set moody.GDP;                /* GDP is specified in the OUT= option */
run;

title3 "The mylib.myGDP Data Set";
proc print data=mylib.myGDP; run;
```

Figure 14.1 US Gross Domestic Product

**National Income and Product Accounts Databank from Moody's Analytics Data Buffet
Request US Gross Domestic Product
The mylib.myGDP Data Set**

Obs	date	GDP\$.IUSA
1	2017-03-31	17925256
2	2017-06-30	18021048
3	2017-09-30	18163558
4	2017-12-31	18322464

Syntax: SASEMOOD Interface Engine

The SASEMOOD interface engine uses standard engine syntax to read the observations or data values for one or more time series from Moody's Analytics Data Buffet.

Table 14.1 summarizes the options that the SASEMOOD engine uses.

Table 14.1 Summary of SASEMOOD Interface Engine Options

Option	Description
CASLIB=	Specifies the name of the CASLIB where the in-memory CAS table containing the MOODY's data is stored
CASOUT=	Specifies the name of the in-memory CAS table, which contains the data that the SASEMOOD interface engine returns
CONV=	Specifies the method to aggregate (or disaggregate) the frequency to
DEBUG=	Specifies whether to include diagnostic message logging in the SAS log window
END=	Specifies the end date of the retrieved data
FREQ=	Specifies the frequency to convert the time series to (if a conversion is necessary)
GOCAS=	Specifies whether to generate an in-memory CAS table for the Moody's data
MNEMONIC=	Specifies the required Moody's data set mnemonic code (concept code and geography code) that enables you to specify which data to select from Moody's Analytics Data Buffet
OUT=	Specifies the name of the SAS data set and the JSON file, which contains the data that the SASEMOOD interface engine returns
PASS=	Specifies the required Moody's encryption key that enables you to access Moody's Analytics Data Buffet
START=	Specifies the start date of the retrieved data
TRANS=	Specifies the transformation function to be applied to the selected data sets
USER=	Specifies the required Moody's access key to authenticate the request for the retrieved data

The LIBNAME libref SASEMOOD Statement

LIBNAME libref SASEMOOD *'physical-name' options* ;

The LIBNAME statement assigns a SAS library reference (libref) to the physical path of the directory where the SASEMOOD interface engine stores the Moody's data. The required *physical-name* argument specifies the final location of the folder where your Moody's data reside in a SAS data set named by the OUT= option. The *physical-name* should end with a backslash if you are in a Windows environment or a forward slash if you are in a UNIX environment.

You can specify the following *options*:

CASLIB=*mood_caslib_name*

specifies the name of the CAS library (CASLIB) where the in-memory CAS table is stored when the JSON data from Moody's Analytics Data Buffet are read into SAS. You can use the CASLIB= option to name your CASLIB where the in-memory CAS table is stored. The default for the CASLIB= option is the active CASLIB. The GOCAS= ON option is required when the CASLIB= option is specified.

CASOUT=*mood_castable_name***OUTCAS=***mood_castable_name*

specifies the name of the in-memory CAS table that is created when the JSON data from Moody's Analytics Data Buffet are read into SAS. You can use the CASOUT= option to name your in-memory CAS table. The default for the CASOUT= option is the same name that you specify in the OUT= option. The in-memory table is created only when the GOCAS=ON option is also specified.

CONV=CONSTANT | CUBIC | DISCRETE | LINEAR**CONVERSION=CONSTANT | CUBIC | DISCRETE | LINEAR**

specifies the technique to use to convert the native frequency to the desired frequency. You can specify the following values:

CONSTANT (or **0**) converts the native frequency to a cubic frequency.

CUBIC (or **2**) converts the native frequency to a constant frequency.

DISCRETE (or **3**) converts the native frequency to a discrete frequency.

LINEAR (or **1**) converts the native frequency to a linear frequency.

By default, CONV=CUBIC. For more about native frequency and frequency conversion, see “Using Data Buffet: The Convert Function” (Thorne 2019d).

For more information about frequency conversion, see “Using Data Buffet: How Frequency Conversion Works” (Zandi 2019) on the web page at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=FE8BEE0D-53E5-4F6A-AF55-8A9E765DCE68>

DEBUG=ON | OFF

specifies whether or not to include diagnostic message logging in the SAS log window. This information can be useful for troubleshooting a problem.

END=*mood_endDate***ENDDATE=***mood_endDate***ENDATE=***mood_endDate***END_TIME=***mood_endDate***ENDTIME=***mood_endDate***OBSERVATION_END=***mood_endDate*

specifies the end date of the Moody's data that you request. Specify the *mood_endDate* in 'YYYY-MM-DD' format. The valid data range of available data varies with each Moody's data set.

FREQ=*mood_frequency***FREQUENCY**=*mood_frequency***INTERVAL**=*mood_frequency***PERIOD**=*mood_frequency*

specifies the desired frequency for the time series that are listed in the MNEMONIC= option. You can specify the *mood_frequency* as annual, quarterly, monthly, weekly, daily, and so on. Native is the default frequency when the FREQ= option is not specified. For more information about frequency conversion, see “Using Data Buffet: Caveats When Converting Low Frequencies” (Thorne 2019a). For a complete list of frequencies and their frequency codes, see [Table 14.2](#).

Table 14.2 Frequency Codes for Moody’s Analytics Data and the FREQ= Option

Interval or Frequency	Moody’s Frequency Code	Description
Yearly, year, annual, ann, anndec	204	Annual, year ending in December
Annnov	203	Annual, year ending in November
Annoct	202	Annual, year ending in October
Annsep	201	Annual, year ending in September
Annaug	200	Annual, year ending in August
Annjul	199	Annual, year ending in July
Annjun	198	Annual, year ending in June
Annmay	197	Annual, year ending in May
Annapr	196	Annual, year ending in April
Annmar	195	Annual, year ending in March
Annfeb	194	Annual, year ending in February
Annjan	193	Annual, year ending in January
Semiannual	188	Semiannual, year ending in December
Semnov	187	Semiannual, year ending in November
Semoct	186	Semiannual, year ending in October
Semsep	185	Semiannual, year ending in September
Semaug	184	Semiannual, year ending in August
Semjul	183	Semiannual, year ending in July
Quarterly, Qtr, Qtrdec, Quarter	172	Quarterly, year ending in December
Qtrnov	171	Quarterly, year ending in November
Qtroct	170	Quarterly, year ending in October
Bimdec	156	Bimonthly, year ending in December
Bimnov	155	Bimonthly, year ending in November
Monthly, mon, month	128	Monthly
Semmon	112	Semimonthly, 15th and end of month
Bwsat2	110	Biweekly, ending on alternating Saturdays, starting Jan. 26, 1850, Feb. 9, 1850, ..., Dec. 30, 2017, Jan. 13, 2018, Jan. 27, 2018
Bwfri2	109	Biweekly, ending on alternating Fridays, starting Jan. 25, 1850, Feb. 8, 1850, ..., Dec. 29, 2017, Jan. 12, 2018, Jan. 26, 2018

Table 14.2 *continued*

Interval or Frequency	Moody's Frequency Code	Description
Bwthu2	108	Biweekly, ending on alternating Thursdays, starting Jan. 24, 1850, Feb. 7, 1850, ..., Dec. 28, 2017, Jan. 11, 2018, Jan. 25, 2018
Bwwed2	107	Biweekly, ending on alternating Wednesdays, starting Jan. 23, 1850, Feb. 6, 1850, ..., Dec. 27, 2017, Jan. 10, 2018, Jan. 24, 2018
Bwtue2	106	Biweekly, ending on alternating Tuesdays, starting Jan. 22, 1850, Feb. 5, 1850, ..., Dec. 26, 2017, Jan. 9, 2018, Jan. 23, 2018
Bwmon2	105	Biweekly, ending on alternating Mondays, starting Jan. 21, 1850, Feb. 4, 1850, ..., Dec. 25, 2017, Jan. 8, 2018, Jan. 22, 2018
Bwsun2	104	Biweekly, ending on alternating Sundays, starting Jan. 20, 1850, Feb. 3, 1850, ..., Dec. 24, 2017, Jan. 7, 2018, Jan. 21, 2018
Bwsat1	103	Biweekly, ending on alternating Saturdays, starting Jan. 19, 1850, Feb. 2, 1850, ..., Dec. 23, 2017, Jan. 6, 2018, Jan. 20, 2018
Bwfri1	102	Biweekly, ending on alternating Fridays, starting Jan. 18, 1850, Feb. 1, 1850, ..., Dec. 22, 2017, Jan. 5, 2018, Jan. 19, 2018
Bwthu1	101	Biweekly, ending on alternating Thursdays, starting Jan. 17, 1850, Jan. 31, 1850, ..., Dec. 21, 2017, Jan. 4, 2018, Jan. 18, 2018
Bwwed1	100	Biweekly, ending on alternating Wednesdays, starting Jan. 16, 1850, Jan. 30, 1850, ..., Dec. 20, 2017, Jan. 3, 2018, Jan. 17, 2018
Bwtue1	99	Biweekly, ending on alternating Tuesdays, starting Jan. 15, 1850, Jan. 29, 1850, ..., Dec. 19, 2017, Jan. 2, 2018, Jan. 16, 2018
Bwmon1	98	Biweekly, ending on alternating Mondays, starting Jan. 14, 1850, Jan. 28, 1850, ..., Dec. 18, 2017, Jan. 1, 2018, Jan. 15, 2018
Bwsun1	97	Biweekly, ending on alternating Sundays, starting Jan. 27, 1850, Feb. 10, 1850, ..., Dec. 31, 2017, Jan. 14, 2018, Jan. 28, 2018
Tenday	80	Three times a month: 10th, 20th, and end of month
Weekly, week, wsaturday	71	Weekly, ending Saturday
Wfriday, wfri	70	Weekly, ending Friday
Wthursday, wthu	69	Weekly, ending Thursday
Wwednesday, wwed	68	Weekly, ending Wednesday

Table 14.2 *continued*

Interval or Frequency	Moody's Frequency Code	Description
Wtuesday, wtue	67	Weekly, ending Tuesday
WMonday, wmon	66	Weekly, ending Monday
WSunday, wsun	65	Weekly, ending Sunday
Business, busin, bus	50	Business daily, Monday through Friday
Daily, day	49	Daily
Index	16	INDEX
Default	0	Native

GOCAS=ON | OFF

specifies whether or not to create an in-memory CAS table of the Moody's data (in addition to the SAS data set that is created). When GOCAS=ON, the SASEMOOD engine assumes that there is an active CAS session running before the LIBNAME statement is used. By default, GOCAS=OFF.

MNEMONIC='mood_mnemonic_list'

ID='mood_mnemonic_list'

IDS='mood_mnemonic_list'

specifies the Moody's mnemonic code (or codes) that enables you to access the data set that corresponds to that code. Each mnemonic code consists of a concept code and a geography code, separated by a period (.). When you want more than one mnemonic code, specify the mnemonic codes in a list, separated by semicolons. There are too many codes to list here; you can view the catalog of the available list by visiting the web page at the following URL:

<https://www.economy.com/databuffet/>

OUT=mood_jsonfile

OUTJSON=mood_jsonfile

JSON=mood_jsonfile

specifies the name of both the JSON file (downloaded) and the SAS data set that is created when the JSON data are read into SAS. You can use the OUT= option to name your JSON data file. It is stored in the SAS Work library. The SAS data set that is created (when the JSON data are read into SAS) is stored in the folder specified by *physical-name*, and you can refer to it by using the myLib libref in your LIBNAME statement.

PASS='mood_password'

ENCRYPTKEY='mood_password'

specifies the unique PASS (password) encryption key that Moody's Analytics assigns to the user (subscriber) to provide security for accessing the Moody's Analytics Data Buffet online data.

USER='mood_user'

ACCESSKEYID='mood_user'

specifies the unique USER access key that Moody's Analytics assigns to the user (subscriber) to allow access to the Moody's Analytics Data Buffet API service.

START=mood_start

STARTDATE=mood_start

STDATE=mood_start

START_TIME=mood_start

STARTTIME=mood_start

BEGIN=mood_start

OBSERVATION_START=mood_start

specifies the start date for the Moody's data that you request. Specify the data in 'YYYY-MM-DD' format. The valid date range for available data varies with each Moody's data set.

TRANS=0 | 1 | 2 | 3 | 4 | 8

TRANSFORMATION=0 | 1 | 2 | 3 | 4 | 8

specifies the transformation function to be performed on the selected series. You can specify the following values:

0 indicates none.

1 indicates year-over-year percentage change.

2 indicates simple difference.

3 indicates annualized growth.

4 indicates percentage change.

8 indicates year-over-year difference.

By default, TRANS=0 (no transformation). For more information about Moody's transformations, see "Using Data Buffet: Transformations" (Thorne 2019e) at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=783658E9-93DA-4F85-A206-9D14FC44904F>

Details: SASEMOOD Interface Engine

The SASEMOOD interface engine enables SAS users to access time series data that are stored in Moody's Analytics Data Buffet data sets. Each Moody's time series is identified by a unique mnemonic code. The mnemonic code consists of a concept code and a geography code. For more information about mnemonic codes, see "Using Data Buffet: Mnemonic, concept code and geo code" (Thorne 2019c) on the web page at the following URL:

<https://www.economy.com/support/blog/buffet.aspx?did=FC5702E3-32B7-4068-AC3B-5D18D3400416>

SAS Output Data Set

You can use a SAS DATA step to write the selected Moody's time series data to a SAS data set. This enables you to use SAS software to easily analyze the data. If you specify the name of the output data set in the DATA statement, the SAS engine supervisor creates a SAS data set that has the specified name in the location specified by the SASEMOOD engine libref's *physical-name*.

The contents of the SAS data set include the date of each observation and the name of each time series that is read from the Moody's website.

You can use the PRINT and CONTENTS procedures to print your output data set and its contents. Alternatively, you can view your SAS output observations by opening the desired output data set in a SAS Explorer window. You can also use the SQL procedure with your SASEMOOD engine libref to create a custom view of your data.

Because each SASEMOOD engine libref results in retrieving the requested data from the Moody's Analytics Data Buffet website, it is best to use a DATA step to store the data. You should avoid the inefficient use of the SASEMOOD engine libref that follows:

```
proc print data=moody.GDP; run;
```

Here the member name, GDP, is used in the PROC PRINT statement that invokes the Moody libref to run the SASEMOOD engine. It is more efficient to refer to the SAS data set myGDP repeatedly than to invoke the interface engine repeatedly. This use of the member name, GDP, corresponds to specifying the OUT=GDP option. Although using this statement might seem easier, it is not as efficient, because every time you use the SASEMOOD engine libref, the SASEMOOD engine reads the entire JSON file into SAS again.

Examples: SASEMOOD Interface Engine

Example 14.1: Retrieving Consumer Price Index Data from Moody's Analytics Data Buffet

This example requests Consumer Price Index data (all items, total for United States) from the OECD Main Economic Indicators database. The MNEMONIC= option specifies the Moody's data to retrieve, which in this example is OCMCPALTT01IXOBAM.IUSA. The FREQ= option requests monthly data. The range is specified using the START= and END= options. The OUTJSON= option names the JSON file and the SAS output data set that contains the time series data from the downloaded JSON file. The CONV= option specifies cubic conversion to the desired frequency. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set.

The SAS code is shown after the next paragraph, and it is followed by the output, shown in [Output 14.1.1](#), for the selected date range.

The SET statement reads observations from the input data set Moody.monCPI, and the DATA step stores them in a SAS data set named myMONCPI. When all the data are retrieved, they are placed in a SAS data set that is named by the OUT= option and that is located in the folder specified by the *physical-name* in the LIBNAME statement.

```
options validvarname=any;

title 'Request Consumer Price Index (Total - All Items - Index 2015-100)';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
    mnemonic="OCMCPALTT01IXOBAM.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=monthly
    conv=2
    outjson=moncpi
    start='2017-01-01'
    end='2018-05-01'
    format=json;

data myMONCPI;
    set moody.moncpi;
run;

proc print data=myMONCPI; run;
```

Output 14.1.1 US Monthly CPI Data in OECD Main Economic Indicators Database**Request Consumer Price Index (Total - All Items - Index 2015=100)**

Obs	date	OCMCPALTT01IXOBAM.IUSA
1	2017-01-31	102.853
2	2017-02-28	102.930
3	2017-03-31	102.840
4	2017-04-30	102.968
5	2017-05-31	102.930
6	2017-06-30	103.023
7	2017-07-31	103.111
8	2017-08-31	103.493
9	2017-09-30	103.966
10	2017-10-31	104.038
11	2017-11-30	104.352
12	2017-12-31	104.592
13	2018-01-31	105.007
14	2018-02-28	105.211
15	2018-03-31	105.266
16	2018-04-30	105.459
17	2018-05-31	105.750

Example 14.2: Retrieving Quarterly Gross Domestic Product Data by Industry from the Bureau of Economic Analysis (BEA)

This example shows how to retrieve gross domestic product data by industry, starting in the second quarter of 2016 and ending in the third quarter of 2018. The output, which is shown in [Output 14.2.1](#), contains two variables, Date and PDPGPOD1Q.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character ‘.’ in the mnemonic code; the same mnemonic code is used to name the SAS variable’s series name in the SAS output data set.

The MNEMONIC= option specifies the Moody’s time series data to retrieve, which in this example is PDPGPOD1Q.IUSA. The FREQ= option selects the quarterly frequency, and the START= and END= options define the date range of the retrieved data.

```
options validvarname=any;

title 'Request Gross Domestic Product by Industry, Quarterly Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="PDPGPOD1Q.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=quarterly
  conv=cubic
  outjson=GDPI
  start='2016-01-01'
  end='2018-06-01'
```

```

format=json
;

data myIGDPQ;
  set moody.GDPI;
run;

proc print data=myIGDPQ; run;

```

Output 14.2.1 GDP by Industry: Chain-Type Price Index for Value Added – All Industries
Request Gross Domestic Product by Industry, Quarterly Data

Obs	date	PDPGPOD1Q.IUSA
1	2016-03-31	105.043
2	2016-06-30	105.738
3	2016-09-30	106.110
4	2016-12-31	106.703
5	2017-03-31	107.233
6	2017-06-30	107.553
7	2017-09-30	108.134
8	2017-12-31	108.807
9	2018-03-31	109.348
10	2018-06-30	110.172

Example 14.3: Retrieving Weekly Unemployment Insurance Claim Statistics for the United States

This example shows how to retrieve weekly unemployment insurance claim statistics data for the United States, starting January 1, 2018, and ending May 1, 2018. The output is shown in [Output 14.3.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve; its mnemonic code is UNIW.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range of the selected data.

```

options validvarname=any;

title 'Request Unemployment Insurance Claim Statistics, Weekly Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  mnemonic="UNIW.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=weekly
  conv=cubic
  outjson=uniweek
  start='2018-01-01'
  end='2018-05-01'
  format=json;

data myUNIWK;
  set moody.uniweek;
run;

```

```
proc contents data=myUNIWK; run;
proc print data=myUNIWK; run;
```

Output 14.3.1 US Unemployment Insurance Claim Weekly Statistics

Request Unemployment Insurance Claim Statistics, Weekly Data

Obs	date	UNIWK.IUSA
1	2018-01-06	250000
2	2018-01-13	230000
3	2018-01-20	238000
4	2018-01-27	225000
5	2018-02-03	219000
6	2018-02-10	226000
7	2018-02-17	216000
8	2018-02-24	216000
9	2018-03-03	223000
10	2018-03-10	222000
11	2018-03-17	221000
12	2018-03-24	217000
13	2018-03-31	231000
14	2018-04-07	227000
15	2018-04-14	227000
16	2018-04-21	210000
17	2018-04-28	213000
18	2018-05-05	211000

Example 14.4: Retrieving Daily Data from the S&P 500

This example shows how to retrieve Standard & Poor's Dow Jones Indices data, starting in January 2018 and ending in February 2018. The output is shown in [Output 14.4.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is SP500D.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range.

```
options validvarname=any;

title 'Request S&P 500 Data, Daily Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
    mnemonic="SP500D.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=daily
    conv=cubic
    outjson=sp500
    start='2018-01-01'
    end='2018-02-01'
```

```

format=json
;

data mySP500;
  set moody.sp500;
run;

```

```
proc print data=work.mySP500; run;
```

Output 14.4.1 Stock Price Index: S&P 500 (Index 1941 to 1943=10, NSA)

Request S&P 500 Data, Daily Data

Obs	date	SP500D.IUSA
1	2018-01-01	2688.01
2	2018-01-02	2695.81
3	2018-01-03	2713.06
4	2018-01-04	2723.99
5	2018-01-05	2743.15
6	2018-01-06	2746.41
7	2018-01-07	2747.02
8	2018-01-08	2749.70
9	2018-01-09	2751.29
10	2018-01-10	2748.23
11	2018-01-11	2767.56
12	2018-01-12	2786.24
13	2018-01-13	2784.92
14	2018-01-14	2780.31
15	2018-01-15	2775.69
16	2018-01-16	2776.42
17	2018-01-17	2802.56
18	2018-01-18	2798.03
19	2018-01-19	2810.30
20	2018-01-20	2824.21
21	2018-01-21	2836.14
22	2018-01-22	2838.57
23	2018-01-23	2839.13
24	2018-01-24	2837.54
25	2018-01-25	2839.25
26	2018-01-26	2872.87
27	2018-01-27	2868.00
28	2018-01-28	2854.91
29	2018-01-29	2837.68
30	2018-01-30	2822.43
31	2018-01-31	2823.81
32	2018-02-01	2821.98

Example 14.5: Retrieving GDP by Industry: Chain-Type Price Index for Value Added for the United States

This example shows how to retrieve multiple time series for GDP by industry—utilities, manufacturing, finance, and all industries—starting January 1, 2016, and ending June 1, 2018. The MNEMONIC= option specifies the Moody's data to retrieve. This example lists four mnemonic codes, separated by semicolons. The FREQ= option specifies quarterly data intervals, and the START= and END= options define the date range of the data. The output is shown in [Output 14.5.1](#), which contains five variables: Date, PDPGPOD1Q.IUSA, PDPGPO22D1Q.IUSA, PDPGPOMFD1Q.IUSA, and PDPGPOFID1Q.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic name list; this list is used to name the SAS variables (time series) in the SAS output data set.

```
options validvarname=any;

title 'Request Gross Domestic Product by Industry for Multiple Time Series';
LIBNAME moody sasemood "%sysget(MOOD_DATA)"
  mnemonic="PDPGPOD1Q.IUSA;PDPGPO22D1Q.IUSA;PDPGPOMFD1Q.IUSA;PDPGPOFID1Q.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  mnemonic="PDPGPOD1Q.IUSA;PDPGPO22D1Q.IUSA;PDPGPOMFD1Q.IUSA;PDPGPOFID1Q.IUSA"
  freq=quarterly
  conv=cubic
  outjson=incGDP
  start='2016-01-01'
  end='2018-06-01'
  format=json;

data myIGDPQ;
  set moody.incGDP;
run;

proc print data=work.myIGDPQ; run;
```

Output 14.5.1 Four Time Series for GDP by Industry: Utilities, Manufacturing, Insurance, and All Industries

Request Gross Domestic Product by Industry for Multiple Time Series

Obs	date	PDPGPOD1Q.IUSA	PDPGPO22D1Q.IUSA	PDPGPOMFD1Q.IUSA	PDPGPOFID1Q.IUSA
1	2016-03-31	105.043	108.154	103.894	112.317
2	2016-06-30	105.738	108.623	104.473	113.485
3	2016-09-30	106.110	110.468	104.659	114.749
4	2016-12-31	106.703	110.421	105.453	115.845
5	2017-03-31	107.233	111.260	105.281	116.159
6	2017-06-30	107.553	112.215	105.881	116.984
7	2017-09-30	108.134	112.827	107.583	118.053
8	2017-12-31	108.807	112.891	108.266	119.637
9	2018-03-31	109.348	115.827	107.490	121.038
10	2018-06-30	110.172	112.499	109.575	121.965

Example 14.6: Retrieving Unemployment Rate Data for the United States

This example shows how to retrieve unemployment rate data for United States (total), starting in January 1973 and ending in January 2019. The output is shown in [Output 14.6.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is LBR.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range. Although the native frequency of this series is monthly, the request is for an annual frequency, using the cubic conversion in the CONV= option.

```
options validvarname=any;

title 'Request Unemployment Data, Annual Frequency Using Cubic Conversion';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
    mnemonic="LBR.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=annual      /* Convert from monthly to annual frequency */
    conv=cubic
    outjson=labor
    start='1973-01-01'
    end='2019-01-01'
    format=json
    ;

data myLabor;
    set moody.labor;
run;

proc print data=work.myLabor; run;
```


Output 14.6.1 US Unemployment Rate: Total (% , SA)**Request Unemployment Data, Annual Frequency Using Cubic Conversion**

Obs	date	LBR.IUSA
1	1973-12-31	4.85833
2	1974-12-31	5.64167
3	1975-12-31	8.47500
4	1976-12-31	7.70000
5	1977-12-31	7.05000
6	1978-12-31	6.06667
7	1979-12-31	5.85000
8	1980-12-31	7.17500
9	1981-12-31	7.61667
10	1982-12-31	9.70833
11	1983-12-31	9.60000
12	1984-12-31	7.50833
13	1985-12-31	7.19167
14	1986-12-31	7.00000
15	1987-12-31	6.17500
16	1988-12-31	5.49167
17	1989-12-31	5.25833
18	1990-12-31	5.61667
19	1991-12-31	6.85000
20	1992-12-31	7.49167
21	1993-12-31	6.90833
22	1994-12-31	6.10000
23	1995-12-31	5.59167
24	1996-12-31	5.40833
25	1997-12-31	4.94167
26	1998-12-31	4.50000
27	1999-12-31	4.21667
28	2000-12-31	3.96667
29	2001-12-31	4.74167
30	2002-12-31	5.78333
31	2003-12-31	5.99167
32	2004-12-31	5.54167
33	2005-12-31	5.08333
34	2006-12-31	4.60833
35	2007-12-31	4.61667
36	2008-12-31	5.80000
37	2009-12-31	9.28333
38	2010-12-31	9.60833
39	2011-12-31	8.93333
40	2012-12-31	8.07500
41	2013-12-31	7.35833
42	2014-12-31	6.15833
43	2015-12-31	5.27500
44	2016-12-31	4.87500
45	2017-12-31	4.35000
46	2018-12-31	3.89167
47	2019-12-31	3.75000

Example 14.7: Retrieving Employment Data: Total, Nonfarm, for the United States

This example shows how to retrieve employment data for the United States (total, nonfarm), starting in June 2015 and ending in January 2019. The TRANS= option specifies the year-over-year percentage change transformation on the time series.

The output is shown in [Output 14.7.1](#). The MNEMONIC= option specifies the Moody's data set to retrieve the data from; its mnemonic code is LBR.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The START= and END= options define the date range.

```
options validvarname=any;

title 'Request Employment Data: Total Nonfarm Using Transformation';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
    mnemonic="ET.IUSA"
    user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
    freq=monthly
    conv=2
    trans=1          /* year-over-year percentage change*/
    outjson=etrans
    start='2015-06-01'
    end='2019-01-01'
    format=json
    ;

data myETrans;
    set moody.etrans;
run;

proc print data=work.myETrans; run;
```

Output 14.7.1 US Employment: Total, Nonfarm (Year-over-Year % Transform)**Request Employment Data: Total Nonfarm Using Transformation**

Obs	date	ET.IUSA
1	2015-06-30	2.06399
2	2015-07-31	2.10808
3	2015-08-31	2.05784
4	2015-09-30	1.92574
5	2015-10-31	1.98011
6	2015-11-30	1.93967
7	2015-12-31	1.94379
8	2016-01-31	1.85337
9	2016-02-29	1.83874
10	2016-03-31	1.94914
11	2016-04-30	1.88198
12	2016-05-31	1.66298
13	2016-06-30	1.74001
14	2016-07-31	1.76670
15	2016-08-31	1.77433
16	2016-09-30	1.86897
17	2016-10-31	1.71657
18	2016-11-30	1.66824
19	2016-12-31	1.61956
20	2017-01-31	1.73166
21	2017-02-28	1.66542
22	2017-03-31	1.58824
23	2017-04-30	1.58730
24	2017-05-31	1.66566
25	2017-06-30	1.62564
26	2017-07-31	1.53053
27	2017-08-31	1.56505
28	2017-09-30	1.38826
29	2017-10-31	1.47803
30	2017-11-30	1.51073
31	2017-12-31	1.48030
32	2018-01-31	1.42215
33	2018-02-28	1.55037
34	2018-03-31	1.58670
35	2018-04-30	1.57276
36	2018-05-31	1.66844
37	2018-06-30	1.68836
38	2018-07-31	1.66829
39	2018-08-31	1.73083
40	2018-09-30	1.79186
41	2018-10-31	1.80025
42	2018-11-30	1.78128
43	2018-12-31	1.81509
44	2019-01-31	1.90841

Example 14.8: Retrieving Unemployment Data and Household Debt to Personal Disposable Income Ratio for the United States

This example shows how to retrieve unemployment data and household debt data for the United States, starting in January 2009 and ending in January 2019. The `FREQ=` option specifies quarterly frequency conversion, using the cubic conversion in the `CONV=` option.

The output is shown in [Output 14.8.1](#). The `MNEMONIC=` option specifies the Moody's data sets to retrieve the data from; their mnemonic codes are `LBR.IUSA` and `DM%CHHDPIAQ.IUSA`. The SAS system option `VALIDVARNAME=ANY` allows for the special character '.' in the mnemonic code; the same mnemonic code is used to name the SAS variable's series name in the SAS output data set. The `START=` and `END=` options define the date range.

```
options validvarname=any;

title 'Request Unemployment Data and Household Debt to Personal Disposable Income Ratio';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  mnemonic="LBR.IUSA;DM%CHHDPIAQ.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=quarterly
  conv=2
  outjson=labdebt
  start='2009-01-01'
  end='2019-01-01'
  format=json
  ;

data myLabD;
  set moody.labdebt;
run;

proc print data=work.myLabD; run;
```

Output 14.8.1 US Unemployment Data and Household Debt to Personal Disposable Income Ratio**Request Unemployment Data and Household Debt to Personal Disposable Income Ratio**

Obs	date	LBR.IUSA	DM%CHHDPIA.Q.IUSA
1	2009-03-31	8.26667	130.204
2	2009-06-30	9.30000	128.010
3	2009-09-30	9.63333	128.343
4	2009-12-31	9.93333	127.017
5	2010-03-31	9.83333	125.155
6	2010-06-30	9.63333	122.541
7	2010-09-30	9.46667	120.804
8	2010-12-31	9.50000	119.317
9	2011-03-31	9.03333	117.183
10	2011-06-30	9.06667	115.857
11	2011-09-30	9.00000	114.054
12	2011-12-31	8.63333	113.148
13	2012-03-31	8.26667	110.396
14	2012-06-30	8.20000	108.820
15	2012-09-30	8.03333	109.323
16	2012-12-31	7.80000	105.984
17	2013-03-31	7.73333	110.214
18	2013-06-30	7.53333	109.379
19	2013-09-30	7.23333	108.786
20	2013-12-31	6.93333	108.428
21	2014-03-31	6.66667	106.470
22	2014-06-30	6.20000	105.553
23	2014-09-30	6.06667	104.550
24	2014-12-31	5.70000	103.773
25	2015-03-31	5.53333	103.274
26	2015-06-30	5.43333	102.861
27	2015-09-30	5.10000	101.807
28	2015-12-31	5.03333	101.643
29	2016-03-31	4.93333	101.437
30	2016-06-30	4.90000	101.910
31	2016-09-30	4.90000	102.117
32	2016-12-31	4.76667	101.525
33	2017-03-31	4.60000	100.814
34	2017-06-30	4.36667	101.078
35	2017-09-30	4.30000	100.757
36	2017-12-31	4.13333	100.742
37	2018-03-31	4.06667	99.822
38	2018-06-30	3.90000	99.618
39	2018-09-30	3.80000	99.460
40	2018-12-31	3.80000	98.976
41	2019-03-31	3.86667	98.887

Example 14.9: Retrieving National Income and Product Account Data

This example shows how to retrieve the National Income and Product Account (NIPA) data for two time series—exports of goods data and NIPA exports of goods and services data—starting January 1, 2017, and ending December 1, 2017. The data are quarterly, because each mnemonic code ends in “Q.IUSA”.

The output is shown in [Output 14.9.1](#). The MNEMONIC= option specifies the Moody’s data sets to retrieve the data from; their mnemonic codes are NCU18_B020RCQ.IUSA and NCU18_A253RCQ.IUSA. The SAS system option VALIDVARNAME=ANY allows for the special character ‘.’ in the mnemonic code; the same mnemonic code is used to name the SAS variable’s series name in the SAS output data set. The START= and END= options define the date range. The GOCAS=ON option specifies the data to be stored in a CAS in-memory table, which is named M2MNIPA in the CASOUT= option. The CASLIB= option specifies the CAS library name where the in-memory table, M2MNIPA, is stored.

```
options validvarname=any;

title 'Request NIPA Data';
LIBNAME moody sasemood "%sysget(MOOD_DATA) "
  gocas=on
  caslib=casuserhdfs /* CAS library for MOODY's data */
  mnemonic="NCU18_B020RCQ.IUSA;NCU18_A253RCQ.IUSA"
  user='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  pass='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
  freq=0
  conv=2
  outjson=YNIPA /* SAS data set name for the incoming NIPA data */
  casout=M2MNIPA /* in-memory CAS table name for requested NIPA data */
  start='2017-01-01'
  end='2017-12-01'
  format=json
;

data myYNIPA;
  set moody.ynipa;
run;

proc print data=work.myYNIPA; run;
```

Output 14.9.1 US Exports of Goods and Exports of Goods and Services

Request NIPA Data

Obs	date	NCU18_B020RCQ.IUSA	NCU18_A253RCQ.IUSA
1	2017-03-31	2303282	1503009
2	2017-06-30	2313183	1508669
3	2017-09-30	2360093	1535383
4	2017-12-31	2450348	1606363

In the preceding code, you can reference the in-memory CAS table, M2MNIPA, later in the same CAS session to save the data in a SASHDAT file, but do not specify the SASHDAT extension in the file name:

```
proc casutil;
    save casdata="M2MNIPA" casout="<physical path and filename>" replace;
quit;
```

To list the tables in the active CASLIB, specify the following:

```
proc casutil; list; quit;
```

References

- Moody's Analytics (2019). "Moody's Analytics Data Buffet API User Guide." Accessed July 17, 2019. <https://github.com/moodysanalytics/databuffet.api.codesamples/blob/master/dataBuffet-api-userguide.md>.
- Thorne, P. (2019a). "Using Data Buffet: Caveats When Converting Low Frequencies." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=D6427BAA-1803-4C88-A7C5-D1F6F791642A>.
- Thorne, P. (2019b). "Using Data Buffet: Data Buffet API v1.0." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=0978F3F5-60FF-41A3-B984-1DA75CD4BBE2>.
- Thorne, P. (2019c). "Using Data Buffet: Mnemonic, Concept Code and Geo Code." Accessed August 16, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=FC5702E3-32B7-4068-AC3B-5D18D3400416>.
- Thorne, P. (2019d). "Using Data Buffet: The CONVERT Function." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=A35042C7-BB22-4647-9570-4B634E9A952C>.
- Thorne, P. (2019e). "Using Data Buffet: Transformations." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=783658E9-93DA-4F85-A206-9D14FC44904F>.
- Thorne, P. (2019f). "Using Data Buffet: Which Series Can You Download?" Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=A00FDE45-EC0B-4140-8378-0FA3474D85FA>.
- Zandi, K. (2019). "Using Data Buffet: How Frequency Conversion Works." Accessed July 17, 2019. <https://www.economy.com/support/blog/buffet.aspx?did=FE8BEE0D-53E5-4F6A-AF55-8A9E765DCE68>.

Chapter 15

The SEVSELECT Procedure

Contents

Overview: SEVSELECT Procedure	808
Using CAS Sessions and CAS Engine Librefs	810
Getting Started: SEVSELECT Procedure	811
A Simple Example of Fitting Predefined Distributions	811
An Example with Left-Truncation and Right-Censoring	816
An Example of Modeling Regression Effects	821
Syntax: SEVSELECT Procedure	825
Functional Summary	826
PROC SEVSELECT Statement	829
BY Statement	842
CLASS Statement	842
DISPLAY Statement	843
DISPLAYOUT Statement	844
DIST Statement	845
EFFECT Statement	847
LOSS Statement	849
NLOPTIONS Statement	852
OUTPUT Statement	852
OUTSCORELIB Statement	856
SCALEMODEL Statement	858
SELECTION Statement	860
WEIGHT Statement	862
Programming Statements	863
Details: SEVSELECT Procedure	863
Predefined Distributions	863
Censoring and Truncation	873
Parameter Estimation Method	875
Parameter Initialization	877
Estimating Regression Effects	878
Empirical Distribution Function Estimation Methods	883
Statistics of Fit	890
Defining a Severity Distribution Model with the FCMP Procedure	895
Predefined Utility Functions	908
Scoring Functions	913
Custom Objective Functions	920
Input Data Tables	923

Output Data Tables	924
Displayed Output	929
ODS Graphics	931
Examples: SEVSELECT Procedure	936
Example 15.1: Defining a Model for the Gaussian Distribution with a Scale Parameter	936
Example 15.2: Defining a Model for Mixed-Tail Distributions	940
Example 15.3: Fitting a Scaled Tweedie Model with Regressors	947
Example 15.4: Fitting Distributions to Interval-Censored Data	952
Example 15.5: Defining a Finite Mixture Model That Has a Scale Parameter	954
Example 15.6: Scale Regression with Rich Regression Effects	960
Example 15.7: Scale Regression Model Selection	965
Example 15.8: Estimating Parameters Using the Cramér–von Mises Estimator	976
Example 15.9: Predicting Mean and Value-at-Risk by Using Scoring Functions	977
References	983

Overview: SEVSELECT Procedure

The SEVSELECT procedure estimates parameters of any arbitrary continuous probability distribution that is used to model the magnitude (severity) of a continuous-valued event of interest. Examples of such events include loss amounts paid by an insurance company and demand of a product as depicted by its sales. PROC SEVSELECT is especially useful when the severity of an event does not follow typical distributions (such as the normal distribution) that are often assumed by standard statistical methods.

PROC SEVSELECT provides a default set of probability distribution models that includes the Burr, exponential, gamma, generalized Pareto, inverse Gaussian (Wald), lognormal, Pareto, Tweedie, and Weibull distributions. In the simplest form, you can estimate the parameters of any of these distributions by using a list of severity values that are recorded in a data table. You can optionally group the values by a set of BY variables. PROC SEVSELECT computes the estimates of the model parameters, their standard errors, and their covariance structure by using the maximum likelihood method for each of the BY groups.

PROC SEVSELECT can fit multiple distributions at the same time and choose the best distribution according to a selection criterion that you specify. You can use seven different statistics of fit as selection criteria. They are log likelihood, Akaike's information criterion (AIC), corrected Akaike's information criterion (AICC), Schwarz Bayesian information criterion (SBC), Kolmogorov-Smirnov statistic (KS), Anderson-Darling statistic (AD), and Cramér–von Mises statistic (CvM).

You can request that the procedure output different types of diagnostic and inferential results, including the summary statistics of analysis variables, progress and status of the nonlinear estimation process, parameter estimates and their standard errors, estimated covariance structure of the parameters, and statistics of fit.

The following key features make PROC SEVSELECT unique among SAS procedures that can estimate continuous probability distributions:

- It enables you to fit a distribution model when the severity values are truncated, censored, or both. You can specify any combination of the following types of censoring and truncation effects: left-censoring,

right-censoring, left-truncation, or right-truncation. This is especially useful in applications with an insurance-type model, where a severity (loss) is reported and recorded only if it is greater than the deductible amount (left-truncation) and where a severity value greater than or equal to the policy limit is recorded at the limit (right-censoring). Another useful application is that of interval-censored data, where you know both the lower limit (right-censoring) and upper limit (left-censoring) on the severity, but you do not know the exact value.

PROC SEVSELECT also enables you to specify a *probability of observability* for the left-truncated data, which is a probability of observing values greater than the left-truncation threshold. This additional information can be useful in certain applications to more correctly model the distribution of the severity of events.

- It uses an appropriate estimator of the empirical distribution function (EDF). EDF is required to compute the KS, AD, and CvM statistics of fit. The procedure also provides the EDF estimates to your custom parameter initialization method. When you specify truncation or censoring, the EDF is estimated by using either Kaplan-Meier's product-limit estimator or Turnbull's estimator. The former is used by default when you specify only one form of censoring effect (right-censoring or left-censoring), and the latter is used by default when you specify both left-censoring and right-censoring effects.
- It enables you to define any arbitrary continuous parametric distribution model and to estimate its parameters. You just need to define the key components of the distribution, such as its probability density function (PDF) and cumulative distribution function (CDF), as a set of functions and subroutines written with the FCMP procedure, which is part of Base SAS software. As long as the functions and subroutines follow certain rules, the SEVSELECT procedure can fit the distribution model defined by them.
- It can model the influence of exogenous or regressor variables on a probability distribution, as long as the distribution has a scale parameter. A linear combination of regression effects is assumed to affect the scale parameter via an exponential link function. This type of model is referred to as the scale regression model.

If a distribution does not have a scale parameter, then either it needs to have another parameter that can be derived from a scale parameter by using a supported transformation or it needs to be reparameterized to have a scale parameter. If neither of these is possible, then regression effects cannot be modeled.

You can easily specify many types of regression effects by using various operators on a set of classification and continuous variables. You can specify classification variables in the CLASS statement. You can also construct the following special effects by using the EFFECT statement: collection effects, multimember effects, polynomial effects, and spline effects.

If you specify a large number of regression effects, then you can use the SELECTION statement to tell PROC SEVSELECT to perform scale regression model selection.

- It enables you to specify your own objective function to be optimized for estimating the parameters of a model. You can write SAS programming statements to specify the contribution of each observation to the objective function. You can use keyword functions such as `_PDF_` and `_CDF_` to generalize the objective function to any distribution. If you do not specify your own objective function, then PROC SEVSELECT estimates the parameters of a model by maximizing the likelihood function of the data.
- It enables you to create scoring functions that offer a convenient way to evaluate any distribution function, such as PDF, CDF, QUANTILE, or your custom distribution function, for a fitted model on new observations.

PROC SEVSELECT is the next-generation version of PROC HPSEVERITY. It requires SAS Cloud Analytic Services (CAS) in order to run. Because PROC SEVSELECT is a next-generation high-performance analytical procedure, it also does the following:

- enables you to run on a cluster of machines that distribute the data and the computations
- exploits all the available cores and concurrent threads

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: SEVSELECT Procedure

This section outlines the use of the SEVSELECT procedure to fit continuous probability distribution models. Three examples illustrate different features of the procedure.

A Simple Example of Fitting Predefined Distributions

The simplest way to use PROC SEVSELECT is to fit all the predefined distributions to a set of values and let the procedure identify the best-fitting distribution.

Consider a lognormal distribution, whose probability density function (PDF) f and cumulative distribution function (CDF) F are as follows, respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2} \quad \text{and} \quad F(x; \mu, \sigma) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$$

The following DATA step statements simulate a sample from a lognormal distribution with population parameters $\mu = 1.5$ and $\sigma = 0.25$, and store the sample in the variable Y of a data set Work.Test_sev1:

```
/*----- Simple Lognormal Example -----*/
data test_sev1(keep=y label='Simple Lognormal Sample');
  call streaminit(45678);
  label y='Response Variable';
  Mu = 1.5;
  Sigma = 0.25;
  do n = 1 to 100;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
  end;
run;
```

NOTE: Input data must be in a CAS table that is accessible in your CAS session. You must refer to this table by using a two-level name. The first level must be a CAS engine libref, and the second level must be the table name. For more information, see the sections “[Using CAS Sessions and CAS Engine Librefs](#)” on page 12 and “[Loading a SAS Data Set onto a CAS Server](#)” on page 13 in Chapter 3, “[Shared Concepts](#).”

You can load the data set Work.Test_sev1 into a data table in your CAS session by using your CAS engine libref with the following DATA step:

```
data mycas.test_sev1;
  set test_sev1;
run;
```

These statements assume that your CAS engine libref is named mycas, as in the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810, but you can substitute any appropriately defined CAS engine libref.

The following statements fit all the predefined distribution models to the values of Y and identify the best distribution according to the corrected Akaike’s information criterion (AICC):

```
proc sevselect data=mycas.test_sev1 crit=aicc
              plots(histogram kernel)=all; /*(cdf pdf);*/
  loss y;
  dist _predefined_;
run;
```

The PROC SEVSELECT statement specifies the input data table along with the model selection criterion. The PLOTS= option displays comparative plots of the probability density function (PDF) and the cumulative distribution function (CDF) for all candidate distributions. The HISTOGRAM and KERNEL options request that the PDF plot contain histogram and kernel density estimates, respectively, which are nonparametric estimates of the density function. The LOSS statement specifies the variable to be modeled, and the DIST statement with the _PREDEFINED_ keyword specifies that all the predefined distribution models be fitted.

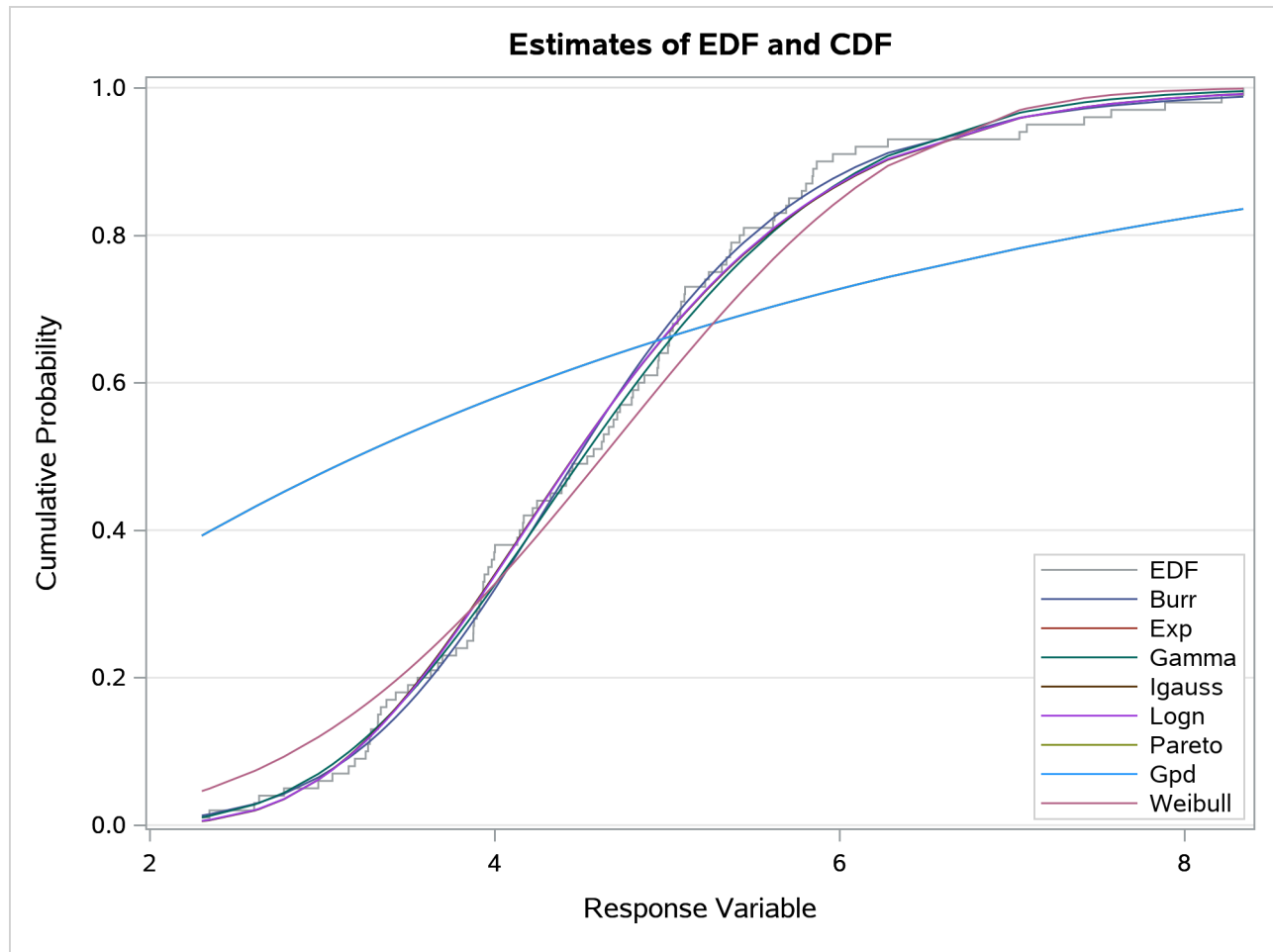
Some of the default output that is displayed by this step is shown in Figure 15.1 through Figure 15.5. First, information about the input data table is displayed followed by the “Model Selection” table, as shown in Figure 15.1. The model selection table displays the convergence status, the value of the selection criterion, and the selection status for each of the candidate models. The Converged column indicates whether the estimation process for a particular distribution model has converged, might have converged, or failed. The Selected column indicates whether a particular distribution has the best fit for the data according to the selection criterion. For this example, the lognormal distribution model is selected, because it has the lowest value for the selection criterion.

Figure 15.1 Data Table Information and Model Selection Table

Input Data Table			
Name	TEST_SEV1		
Caslib	<< active caslib >>		

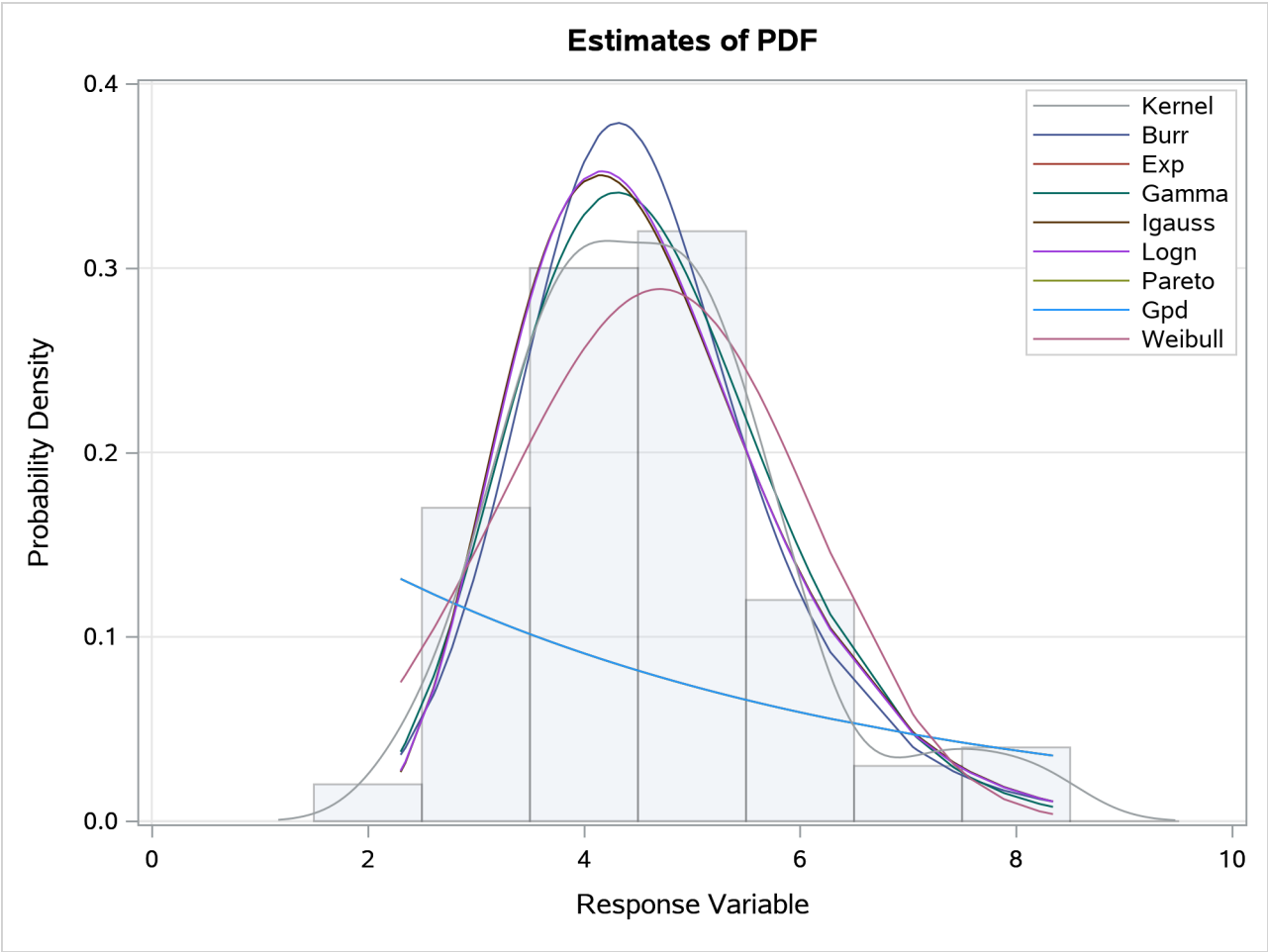
Model Selection			
Distribution	Converged	AICC	Selected
Burr	Yes	322.50845	No
Exp	Yes	508.12287	No
Gamma	Yes	320.50264	No
Igauss	Yes	319.61652	No
Logn	Yes	319.56579	Yes
Pareto	Yes	510.28172	No
Gpd	Yes	510.20576	No
Weibull	Yes	334.82373	No

Next, two comparative plots are prepared. These plots enable you to visually verify how the models differ from each other and from the nonparametric estimates. The plot in Figure 15.2 displays the CDF estimates of all the models and the estimates of the empirical distribution function (EDF). The CDF plot indicates that the Exp (exponential), Pareto, and Gpd (generalized Pareto) distributions are a poor fit as compared to the EDF estimate. The Weibull distribution is also a poor fit, although not as poor as exponential, Pareto, and generalized Pareto. The other four distributions seem to be quite close to each other and to the EDF estimate.

Figure 15.2 Comparison of EDF and CDF Estimates of the Fitted Models

The plot in [Figure 15.3](#) displays the PDF estimates of all the models and the nonparametric kernel and histogram estimates. The PDF plot enables better visual comparison between the Burr, gamma, Igauss (inverse Gaussian), and Logn (lognormal) models. The Burr and gamma models differ significantly from the Igauss and Logn models in the central portion of the range of Y values, whereas the latter two models fit the data almost identically. This provides a visual confirmation of the information in the “Model Selection” table in [Figure 15.1](#), which indicates that the AICC values of the Igauss and Logn models are very close.

Figure 15.3 Comparison of PDF Estimates of the Fitted Models



NOTE: When you request plots in a PROC SEVSELECT step that uses more than one worker node, the plots are created by merging empirical quantiles that each worker node computes by using the sample that it uses to compute the EDF-based fit statistics. So the plots are approximate. For more accurate visualization, it is recommended that you request plots only when you run PROC SEVSELECT in a CAS session that uses no more than one worker node.

The comparative plots are followed by the estimation information for each of the candidate models. The information for the lognormal model, which is the best-fitting model, is shown in Figure 15.4. The first table displays a summary of the distribution. The second table displays the convergence status. This is followed by a summary of the optimization process that indicates the technique used, the number of iterations, the number of times the objective function was evaluated, and the log likelihood that is obtained at the end of the optimization. Because the model with lognormal distribution has converged, PROC SEVSELECT displays its statistics of fit and parameter estimates. The estimates of $\mu=1.49605$ and $\sigma=0.26243$ are quite close to the population parameters of $\mu=1.5$ and $\sigma=0.25$ from which the sample was generated. The p -value for each estimate indicates the rejection of the null hypothesis that the estimate is 0, implying that both the estimates are significantly different from 0.

Figure 15.4 Estimation Details for the Lognormal Model**The SEVSELECT Procedure****Logn Distribution**

Model Information					
Distribution					Logn
Description					Lognormal Distribution
Distribution Parameters					2
Convergence Status					
Convergence criterion (GCONV=1E-8) satisfied.					
Optimization Summary					
Optimization Technique					Trust Region
Iterations					2
Function Calls					8
Log Likelihood					-157.72104
Fit Statistics					
-2 Log Likelihood					315.44208
Akaike's Information Criterion					319.44208
Corrected Akaike's Information Criterion					319.56579
Schwarz's Bayesian Information Criterion					324.65242
Kolmogorov-Smirnov Statistic					0.50641
Anderson-Darling Statistic					0.31240
Cramer-von Mises Statistic					0.04353
Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.49605	0.02651	56.43	<.0001
Sigma	1	0.26243	0.01874	14.00	<.0001

The parameter estimates of the Burr distribution are shown in [Figure 15.5](#). These estimates are used in the next example.

Figure 15.5 Parameter Estimates for the Burr Model

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.62348	0.46181	10.01	<.0001
Alpha	1	1.15706	0.47493	2.44	0.0167
Gamma	1	6.41227	0.99039	6.47	<.0001

An Example with Left-Truncation and Right-Censoring

PROC SEVSELECT enables you to specify that the response variable values are left-truncated or right-censored. The following DATA step expands the data table of the previous example to simulate a scenario that is typically encountered by an automobile insurance company. The values of the variable Y represent the loss values on claims that are reported to an auto insurance company. The variable THRESHOLD records the deductible on the insurance policy. If the actual value of Y is less than or equal to the deductible, then it is unobservable and does not get recorded. In other words, THRESHOLD specifies the left-truncation of Y. LIMIT records the policy limit. If the value of Y is equal to or greater than the recorded value, then the observation is right-censored.

```
/*----- Lognormal Model with left-truncation and censoring -----*/
data test_sev2(keep=y threshold limit
               label='A Lognormal Sample With Censoring and Truncation');
  set test_sev1;
  label y='Censored & Truncated Response';
  if _n_ = 1 then call streaminit(45679);

  /* make about 20% of the observations left-truncated */
  if (rand('UNIFORM') < 0.2) then
    threshold = y * (1 - rand('UNIFORM'));
  else
    threshold = .;
  /* make about 15% of the observations right-censored */
  iscens = (rand('UNIFORM') < 0.15);
  if (iscens) then
    limit = y;
  else
    limit = .;
run;
```

The following DATA step loads the data set Work.Test_sev2 into a data table in your CAS session that is associated with the mycas CAS engine libref:

```
data mycas.test_sev2;
  set test_sev2;
run;
```

The following statements fit four predefined distributions (lognormal, Burr, gamma, and Weibull) and use the AICC to identify the distribution that best fits the data:

```
proc sevselect data=mycas.test_sev2 crit=aicc
               print=all plots=all;
  loss y / lt=threshold rc=limit;

  dist logn burr gamma weibull;
run;
```

The LOSS statement specifies the left-truncation and right-censoring variables. The DIST statement specifies the candidate distributions. The PRINT= option in the PROC SEVSELECT statement requests that all the displayed output be prepared. The PLOTS= option in the PROC SEVSELECT statement requests that all available plots be prepared.

Some of the key results that PROC SEVSELECT prepares are shown in Figure 15.6 through Figure 15.13. In addition to the estimates of the range, mean, and standard deviation of Y , the “Descriptive Statistics for y ” table shown in Figure 15.6 also indicates the number of observations that are left-truncated or right-censored. The “Model Selection” table in Figure 15.6 shows that models with all the candidate distributions have converged and that the Logn (lognormal) model has the best fit for the data according to the AICC criterion.

Figure 15.6 Summary Results for the Truncated and Censored Data

Descriptive Statistics for y			
Observations			100
Observations Used for Estimation			100
Minimum			2.30264
Maximum			8.34116
Mean			4.62007
Standard Deviation			1.23627
Left Truncated Observations			23
Right Censored Observations			14

Model Selection			
Distribution	Converged	AICC	Selected
Logn	Yes	298.92672	Yes
Burr	Yes	302.66229	No
Gamma	Yes	299.45293	No
Weibull	Yes	309.26779	No

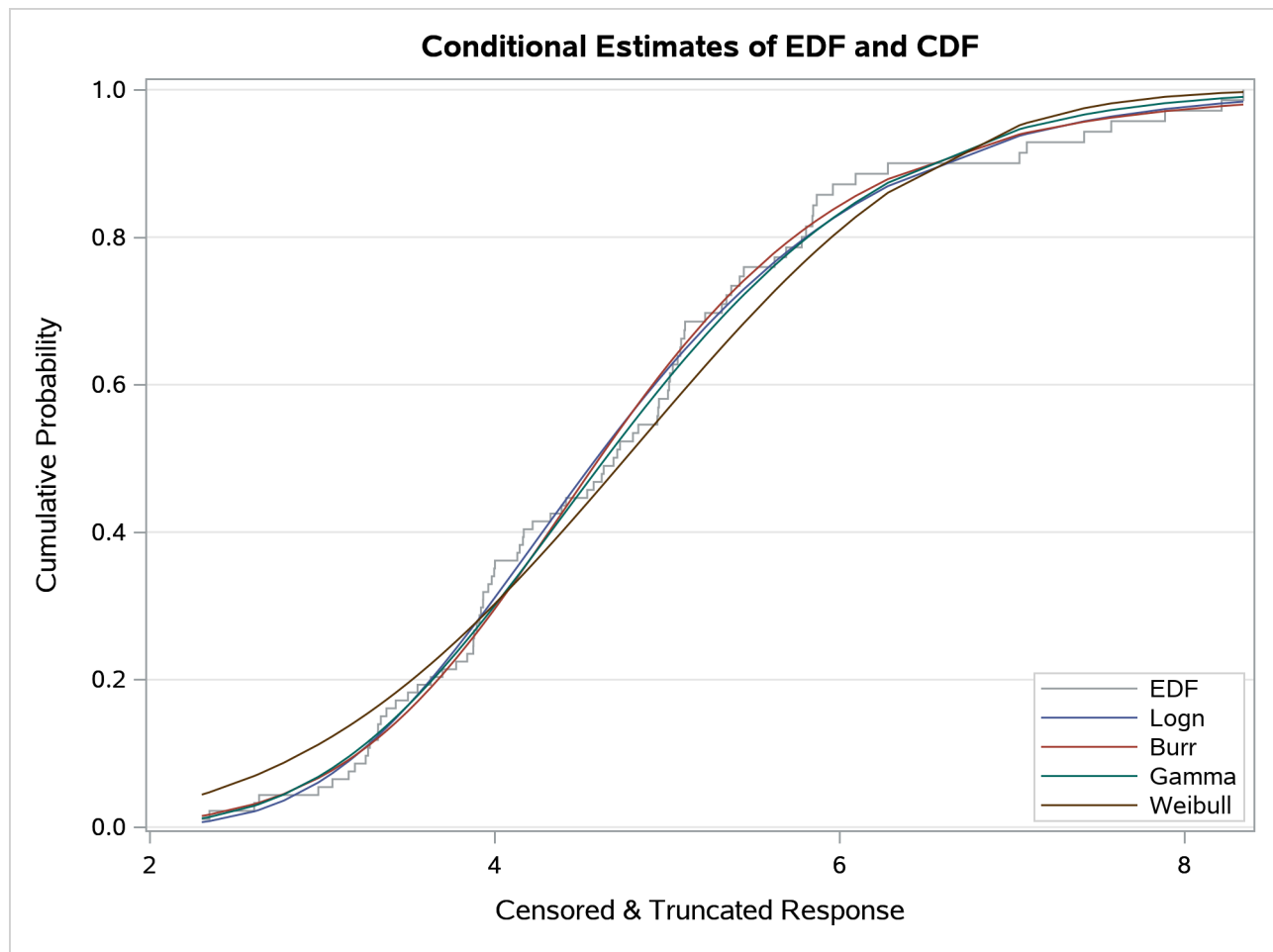
PROC SEVSELECT also prepares a table that shows all the fit statistics for all the candidate models. It is useful to see which model would be the best fit according to each of the criteria. The “All Fit Statistics” table prepared for this example is shown in Figure 15.7. It indicates that the lognormal model is chosen by all the criteria.

Figure 15.7 Comparing All Statistics of Fit for the Truncated and Censored Data

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	294.80301*	298.80301*	298.92672*	304.01335*	0.51824*	0.34736*	0.05159*
Burr	296.41229	302.41229	302.66229	310.22780	0.66984	0.36712	0.05726
Gamma	295.32921	299.32921	299.45293	304.53955	0.62511	0.42921	0.05526
Weibull	305.14408	309.14408	309.26779	314.35442	0.93307	1.40699	0.17465

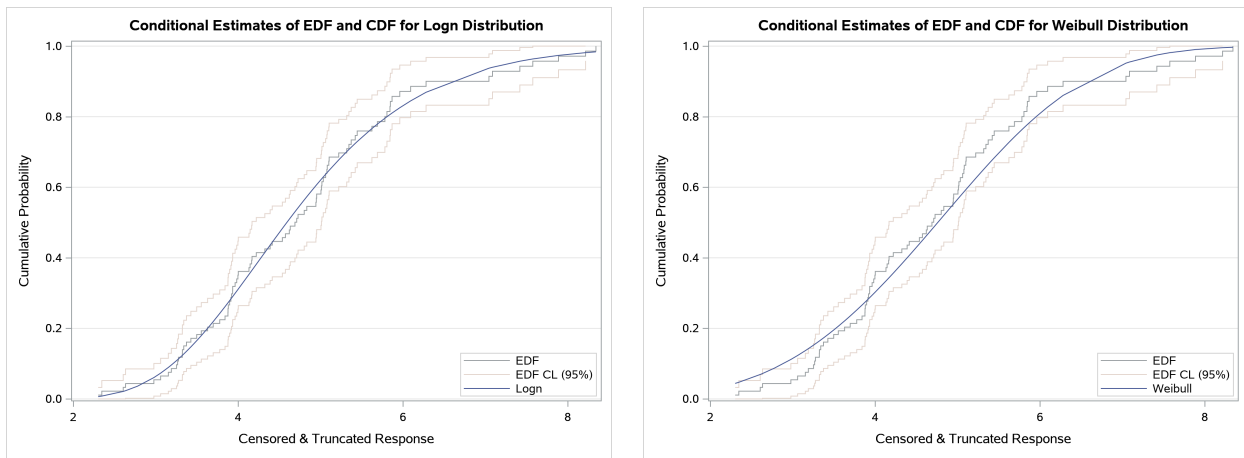
Asterisk (*) denotes the best model in the column.

The plot that compares EDF and CDF estimates is shown in Figure 15.8. When you specify left-truncation, both the EDF and CDF estimates are conditional on the response variable being greater than the smallest left-truncation threshold in the sample.

Figure 15.8 EDF and CDF Estimates for the Truncated and Censored Data

When you specify the `PLOTS=CDFPERDIST` option, PROC SEVSELECT prepares a plot that compares the nonparametric EDF estimates with the parametric CDF estimates for each distribution. These plots for lognormal and Weibull distributions are shown in Figure 15.9. These plots also contain the lower and upper confidence limits of EDF for the specified confidence level. Because no confidence level is specified in the `EDFALPHA=` option in the PROC SEVSELECT statement, a default confidence level of 95% is used; this is equivalent to specifying `EDFALPHA=0.05`. If the CDF estimates lie entirely within the EDF confidence interval, then you can be 95% confident that the parametric and nonparametric estimates are in agreement.

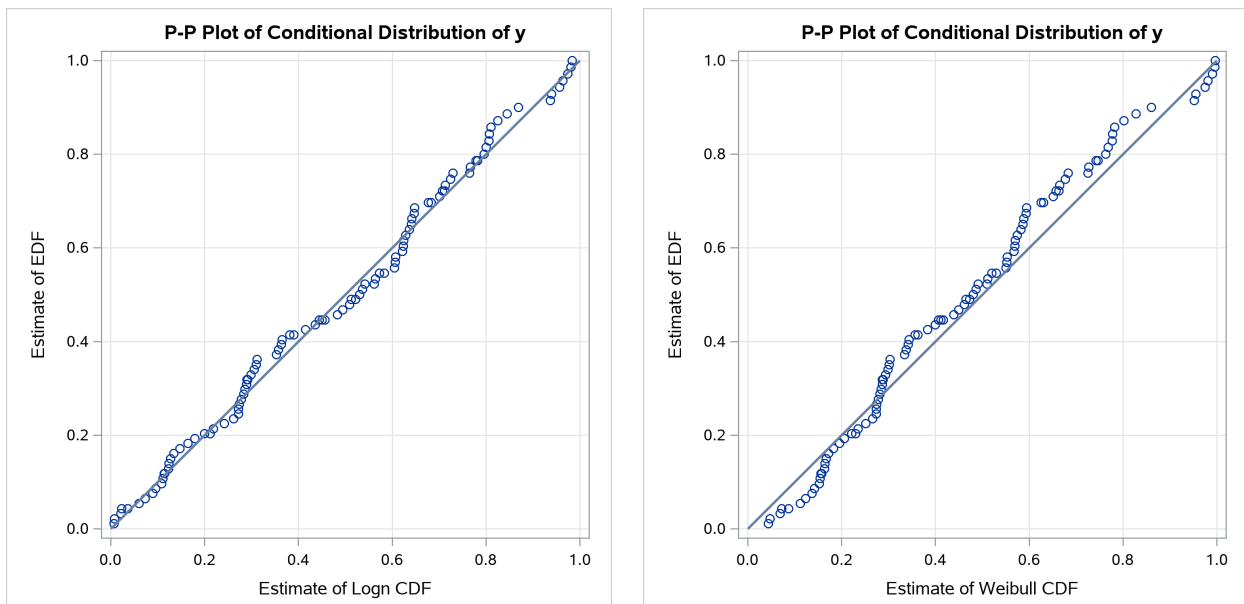
Figure 15.9 Comparing EDF and CDF Estimates for Lognormal and Weibull Models Fitted to Truncated and Censored Data



There are two additional ways to compare nonparametric (empirical) and parametric estimates for each model that has not failed to converge:

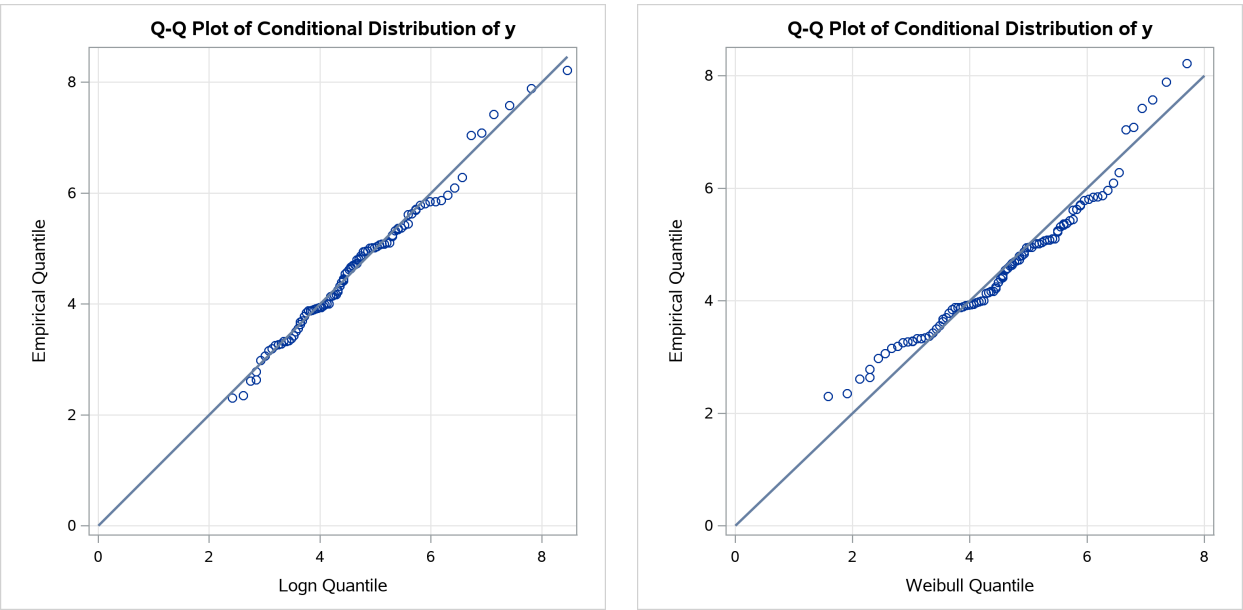
- A P-P plot is a scatter plot of the EDF and CDF estimates. The model for which the points are scattered closer to the unit-slope reference line is a better fit. The P-P plot for the lognormal distribution is shown in Figure 15.10. It indicates that the EDF matches the CDF very closely. In contrast, the P-P plot for the Weibull distribution, also shown in Figure 15.10, indicates a poor fit.

Figure 15.10 P-P Plots for Lognormal and Weibull Models Fitted to Truncated and Censored Data



- A Q-Q plot is a scatter plot of empirical quantiles and the quantiles of a parametric distribution. As in the P-P plot, points that are scattered closer to the unit-slope reference line indicate a better fit. The Q-Q plots of lognormal and Weibull distributions are shown in Figure 15.11; these plots confirm the conclusions that are arrived at by comparing the P-P plots.

Figure 15.11 Q-Q Plots for Lognormal and Weibull Models Fitted to Truncated and Censored Data



Specifying Initial Values for Parameters

All the predefined distributions have parameter initialization functions built into them. For the current example, Figure 15.12 shows the initial values that are obtained by the predefined method for the Burr distribution. It also shows the summary of the optimization process and the final parameter estimates.

Figure 15.12 Burr Model Summary for the Truncated and Censored Data

Initial Parameter Values and Bounds					
Parameter	Initial Value	Lower Bound	Upper Bound		
Theta	4.78102	1.05367E-8	Infy		
Alpha	2.00000	1.05367E-8	Infy		
Gamma	2.00000	1.05367E-8	Infy		
Optimization Summary					
Optimization Technique		Trust Region			
Iterations		8			
Function Calls		23			
Log Likelihood		-148.2061441			
Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.76980	0.62492	7.63	<.0001
Alpha	1	1.16363	0.58859	1.98	0.0509
Gamma	1	5.94081	1.05004	5.66	<.0001

You can specify a different set of initial values if estimates are available from fitting the distribution to similar

data. For this example, the parameters of the Burr distribution can be initialized with the final parameter estimates of the Burr distribution that were obtained in the first example (shown in Figure 15.5). One of the ways in which you can specify the initial values is as follows:

```
/*----- Specifying initial values using INIT= option -----*/
proc sevselect data=mycas.test_sev2 crit=aicc print=all;
  loss y / lt=threshold rc=limit;

  dist burr(init=(theta=4.62348 alpha=1.15706 gamma=6.41227));
run;
```

The names of the parameters that are specified in the INIT option must match the parameter names in the definition of the distribution. The results obtained with these initial values are shown in Figure 15.13. These results indicate that new set of initial values causes the optimizer to reach the same solution with fewer iterations and function evaluations as compared to the default initialization.

Figure 15.13 Burr Model Optimization Summary for the Truncated and Censored Data

The SEVSELECT Procedure

Burr Distribution

Optimization Summary					
Optimization Technique		Trust Region			
Iterations		5			
Function Calls		16			
Log Likelihood		-148.2061441			

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	4.76980	0.62492	7.63	<.0001
Alpha	1	1.16363	0.58859	1.98	0.0509
Gamma	1	5.94081	1.05004	5.66	<.0001

An Example of Modeling Regression Effects

Consider a scenario in which the magnitude of the response variable might be affected by some regressor (exogenous or independent) variables. The SEVSELECT procedure enables you to model the effect of such variables on the distribution of the response variable via an exponential link function. In particular, if you have k random regressor variables denoted by x_j ($j = 1, \dots, k$), then the distribution of the response variable Y is assumed to have the form

$$Y \sim \exp\left(\sum_{j=1}^k \beta_j x_j\right) \cdot \mathcal{F}(\Theta)$$

where $\mathcal{F}$ denotes the distribution of Y with parameters Θ and β_j ($j = 1, \dots, k$) denote the regression parameters (coefficients).

For the effective distribution of Y to be a valid distribution from the same parametric family as $\mathcal{F}$, it is necessary for $\mathcal{F}$ to have a scale parameter. The effective distribution of Y can be written as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

where θ denotes the scale parameter and Ω denotes the set of nonscale parameters. The scale θ is affected by the regressors as

$$\theta = \theta_0 \cdot \exp\left(\sum_{j=1}^k \beta_j x_j\right)$$

where θ_0 denotes a *base* value of the scale parameter.

Given this form of the model, PROC SEVSELECT allows a distribution to be a candidate for modeling regression effects only if it has an untransformed or a log-transformed scale parameter.

All the predefined distributions, except the lognormal distribution, have a direct scale parameter (that is, a parameter that is a scale parameter without any transformation). For the lognormal distribution, the parameter μ is a log-transformed scale parameter. This can be verified by replacing μ with a parameter $\theta = e^\mu$, which results in the following expressions for the PDF f and the CDF F in terms of θ and σ , respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x; \theta, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x) - \log(\theta)}{\sigma}\right)^2} \quad \text{and} \quad F(x; \theta, \sigma) = \Phi\left(\frac{\log(x) - \log(\theta)}{\sigma}\right)$$

With this parameterization, the PDF satisfies the $f(x; \theta, \sigma) = \frac{1}{\theta} f\left(\frac{x}{\theta}; 1, \sigma\right)$ condition and the CDF satisfies the $F(x; \theta, \sigma) = F\left(\frac{x}{\theta}; 1, \sigma\right)$ condition. This makes θ a scale parameter. Hence, $\mu = \log(\theta)$ is a log-transformed scale parameter and the lognormal distribution is eligible for modeling regression effects.

The following DATA step simulates a lognormal sample whose scale is decided by the values of the three regressors X1, X2, and X3 as follows:

$$\mu = \log(\theta) = 1 + 0.75 X1 - X2 + 0.25 X3$$

```
/*----- Lognormal Model with Regressors -----*/
data test_sev3(keep=y x1-x3
               label='A Lognormal Sample Affected by Regressors');
  array x{*} x1-x3;
  array b{4} _TEMPORARY_ (1 0.75 -1 0.25);
  call streaminit(45678);
  label y='Response Influenced by Regressors';
  Sigma = 0.25;
  do n = 1 to 100;
    Mu = b(1); /* log of base value of scale */
    do i = 1 to dim(x);
      x(i) = rand('UNIFORM');
      Mu = Mu + b(i+1) * x(i);
    end;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
  end;
run;
```

The following DATA step loads the data set Work.Test_sev3 into a data table in your CAS session that is associated with the mycas CAS engine libref:


```
data mycas.test_sev3;
  set test_sev3;
run;
```

The following PROC SEVSELECT step fits the lognormal, Burr, and gamma distribution models to these data. The regressors are specified in the SCALEMODEL statement.

```
proc sevselect data=mycas.test_sev3 crit=aicc print=all;
  loss y;
  scalemodel x1-x3;

  dist logn burr gamma;
run;
```

Some of the key results that PROC SEVSELECT prepares are shown in Figure 15.14 through Figure 15.18. The descriptive statistics of all the variables are shown in Figure 15.14.

Figure 15.14 Summary Results for the Regression Example

The SEVSELECT Procedure

Descriptive Statistics for y					
Observations					100
Observations Used for Estimation					100
Minimum					1.17863
Maximum					6.65269
Mean					2.99859
Standard Deviation					1.12845

Descriptive Statistics for Regressors					
Variable	N	Minimum	Maximum	Mean	Standard Deviation
x1	100	0.0005115	0.97971	0.51689	0.28206
x2	100	0.01883	0.99937	0.47345	0.28885
x3	100	0.00255	0.97558	0.48301	0.29709

The comparison of the fit statistics of all the models is shown in Figure 15.15. It indicates that the lognormal model is the best model according to each of the likelihood-based statistics, whereas the gamma model is the best model according to two of the three EDF-based statistics.

Figure 15.15 Comparison of Statistics of Fit for the Regression Example

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	187.49609*	197.49609*	198.13439*	210.52194*	1.97544	17.24618	1.21665
Burr	190.69154	202.69154	203.59476	218.32256	2.09334	13.93436*	1.28529
Gamma	188.91483	198.91483	199.55313	211.94069	1.94472*	15.84787	1.17617*

Asterisk (*) denotes the best model in the column.

The model information and the convergence results of the lognormal model are shown in Figure 15.16. The iteration history gives you a summary of how the optimizer is traversing the surface of the log-likelihood function in its attempt to reach the optimum. Both the change in the log likelihood and the maximum gradient of the objective function with respect to any of the parameters typically approach 0 if the optimizer converges.

Figure 15.16 Convergence Results for the Lognormal Model with Regressors

The SEVSELECT Procedure				
Logn Distribution				
Model Information				
Distribution	Logn			
Description	Lognormal Distribution			
Distribution Parameters	2			
Regression Parameters	3			
Convergence Status				
Convergence criterion (GCONV=1E-8) satisfied.				
Optimization Iteration History				
Iter	Function Calls	Log Likelihood	Change	Maximum Gradient
0	2	93.75285		6.16002
1	4	93.74805	-0.00481	0.11031
2	6	93.74805	-1.5017E-6	0.0000338
3	10	93.74805	-1.279E-13	3.1832E-12
Optimization Summary				
Optimization Technique		Trust Region		
Iterations		3		
Function Calls		10		
Log Likelihood		-93.74804701		

The final parameter estimates of the lognormal model are shown in Figure 15.17. All the estimates are significantly different from 0. The estimate that is reported for the parameter MU is the base value for the log-transformed scale parameter μ . Let x_i ($1 \leq i \leq 3$) denote the observed value for regressor X_i . If the lognormal distribution is chosen to model Y , then the effective value of the parameter μ varies with the observed values of regressors as

$$\mu = 1.04047 + 0.65221 x_1 - 0.91116 x_2 + 0.16243 x_3$$

These estimated coefficients are reasonably close to the population parameters (that is, within one or two standard errors).

Figure 15.17 Parameter Estimates for the Lognormal Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.04047	0.07614	13.66	<.0001
Sigma	1	0.22177	0.01609	13.78	<.0001
x1	1	0.65221	0.08167	7.99	<.0001
x2	1	-0.91116	0.07946	-11.47	<.0001
x3	1	0.16243	0.07782	2.09	0.0395

The estimates of the gamma distribution model, which is the best model according to a majority of the EDF-based statistics, are shown in [Figure 15.18](#). The estimate that is reported for the parameter *Theta* is the base value for the scale parameter θ . If the gamma distribution is chosen to model *Y*, then the effective value of the scale parameter is $\theta = 0.14293 \exp(0.64562 x_1 - 0.89831 x_2 + 0.14901 x_3)$.

Figure 15.18 Parameter Estimates for the Gamma Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	0.14293	0.02329	6.14	<.0001
Alpha	1	20.37726	2.93277	6.95	<.0001
x1	1	0.64562	0.08224	7.85	<.0001
x2	1	-0.89831	0.07962	-11.28	<.0001
x3	1	0.14901	0.07870	1.89	0.0613

Syntax: SEVSELECT Procedure

The following statements are available in the SEVSELECT procedure:

```

PROC SEVSELECT options ;
  BY variable-list ;
  LOSS < response-variable > < / censoring-truncation-options > ;
  WEIGHT weight-variable ;
  DIST distribution-name-or-keyword < (distribution-option) < distribution-name-or-keyword
    < (distribution-option) >> ... > < / options > ;
  CLASS variable < (options) >> ... < variable < (options) >> < / global-options > ;
  EFFECT name=effect-type(variables < / options >) ;
  SCALEMODEL regression-effect-list < / scalemodel-options > ;
  SELECTION < METHOD=method < (method-options) >> < options > ;
  DISPLAY < table-list > < / options > ;
  DISPLAYOUT table-spec-list < / options > ;
  OUTPUT < OUT=CAS-libref.data-table > output-options ;
  OUTSCORELIB < OUTLIB= > fcmp-library-name options ;
  NLOPTIONS options ;
Programming statements ;

```

Functional Summary

Table 15.1 summarizes the statements and options that control the SEVSELECT procedure.

Table 15.1 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies the response variable to model along with censoring and truncation effects	LOSS	
Specifies the weight variable	WEIGHT	
Specifies distributions to fit	DIST	
Specifies the classification variables	CLASS	
Specifies the constructed regression effects	EFFECT	
Specifies the regression effects to model	SCALEMODEL	
Specifies the options for selecting the best subset of regression effects	SELECTION	
Specifies the ODS tables to display (an alternative to the PRINT= option)	DISPLAY	
Specifies the ODS tables to save as CAS output tables	DISPLAYOUT	
Specifies the scoring functions and quantiles to write	OUTPUT	
Specifies the library to write scoring functions to	OUTSCORELIB	
Specifies the optimization options	NLOPTIONS	
Specifies programming statements that define an objective function	Programming statements	
Input and Output Options		
Specifies the input data table	PROC SEVSELECT	DATA=
Specifies the input data table for parameter estimates	PROC SEVSELECT	INEST=
Specifies the output data table for estimates of scoring functions and quantiles	OUTPUT	OUT=
Specifies the output data table for parameter estimates	PROC SEVSELECT	OUTEST=
Specifies the output data table for model information	PROC SEVSELECT	OUTMODELINFO=
Specifies the output data table for statistics of fit	PROC SEVSELECT	OUTSTAT=
Specifies the output item store for context and estimation results	PROC SEVSELECT	OUTSTORE=
Data Interpretation Options		
Specifies left-censoring	LOSS	LEFTCENSORED=
Specifies left-truncation	LOSS	LEFTTRUNCATED=

Table 15.1 *continued*

Description	Statement	Option
Specifies the probability of observability	LOSS	PROBOBSERVED=
Specifies right-censoring	LOSS	RIGHTCENSORED=
Specifies right-truncation	LOSS	RIGHTTRUNCATED=
Distribution Processing Options		
Specifies the CAS table to read the distribution definitions from	DIST	INFUNCDEF=
Specifies that distributions be listed in the log without estimating any models that use them	DIST	LISTONLY
Specifies the CAS table to write the distribution definitions to	DIST	OUTFUNCDEF=
Specifies that distributions be validated without estimating any models that use them	DIST	VALIDATEONLY
Model Estimation Options		
Specifies the method for computing mixture distribution	SCALEMODEL	DFMIXTURE=
Specifies that informative missingness be used to model missing values of regressors	SCALEMODEL	INFORMATIVE
Specifies initial values for model parameters	DIST	INIT=
Specifies the sample to be used for computing initial parameter estimates	PROC SEVSELECT	INITSAMPLE
Specifies that observations do not need to be balanced among threads of computation	PROC SEVSELECT	NOBALANCETHREADS
Specifies role of constant distribution parameters in fit statistic calculations	PROC SEVSELECT	NOCONSTFITSTATS
Specifies the objective function symbol	PROC SEVSELECT	OBJECTIVE=
Specifies the offset variable in the scale regression model	SCALEMODEL	OFFSET=
Specifies the denominator for computing covariance estimates	PROC SEVSELECT	VARDEF=
Regression Effect Selection Options		
Specifies whether and how the model hierarchy requirement is applied	SELECTION	HIERARCHY=
Specifies the set of regression effects to be forced in all models	SCALEMODEL	INCLUDE=
Specifies the selection method	SELECTION	METHOD=
Specifies the selection order for displaying effects in the selected model	SELECTION	ORDERSELECT
Specifies the optimization technique to use for the intermediate selection steps	PROC SEVSELECT	SELECTNLOTECH=
Specifies the set of regression effects to begin the selection process	SCALEMODEL	START=

Table 15.1 *continued*

Description	Statement	Option
Specifies how to apply the STOP= criterion	SELECTION	STOPHORIZON=
Regression Effect Selection Method Options		
Specifies a criterion for choosing the best model at each step	SELECTION	CHOOSE=
Specifies that the competitive form of the stepwise selection method be used	SELECTION	COMPETITIVE
Specifies the maximum number of effects in the model	SELECTION	MAXEFFECTS=
Specifies the maximum number of selection steps	SELECTION	MAXSTEPS=
Specifies the minimum number of effects in the model	SELECTION	MINEFFECTS=
Specifies a criterion to decide which effects enter or leave the model at each step	SELECTION	SELECT=
Specifies a criterion for stopping the selection process	SELECTION	STOP=
Empirical Distribution Function (EDF) Estimation Options		
Specifies the nonparametric method of CDF estimation	PROC SEVSELECT	EMPIRICALCDF=
EMPIRICALCDF=MODIFIEDKM Options		
Specifies the α value for the lower bound on risk set size	PROC SEVSELECT	ALPHA=
Specifies the c value for the lower bound on risk set size	PROC SEVSELECT	C=
Specifies the absolute lower bound on risk set size	PROC SEVSELECT	RSLB=
EMPIRICALCDF=TURNBULL Options		
Specifies that the final EDF estimates be maximum likelihood estimates	PROC SEVSELECT	ENSUREMLE
Specifies the relative convergence criterion	PROC SEVSELECT	EPS=
Specifies the maximum number of iterations	PROC SEVSELECT	MAXITER=
Specifies the threshold below which an EDF estimate is deemed to be 0	PROC SEVSELECT	ZEROPROB=
OUT= Data Table Generation Options		
Specifies the variables to copy from the DATA= data table to the OUT= data table	OUTPUT	COPYVARS=
Specifies the scoring functions to estimate	OUTPUT	FUNCTIONS=
Specifies the quantiles to estimate	OUTPUT	QUANTILES=

Table 15.1 *continued*

Description	Statement	Option
OUTEST= Data Table Generation Options		
Specifies that the OUTEST= data table contain covariance estimates	PROC SEVSELECT	COVOUT
Specifies that only the selected regression parameters be written	PROC SEVSELECT	SELECTOUT
Specifies that estimates of parameters not in the final model be 0	PROC SEVSELECT	ZEROEST
Scoring Function Generation Options		
Writes scoring functions of all models to one package	OUTSCORELIB	COMMONPACKAGE
Specifies the output data table for BY-group identifiers	OUTSCORELIB	OUTBYID=
Specifies the output library for scoring functions	OUTSCORELIB	OUTLIB=
Displayed Output Options		
Specifies the criterion to report in the model selection table	PROC SEVSELECT	CRITERION=
Specifies the level of detail to be produced about the selection process	SELECTION	DETAILS=
Specifies the confidence level for reporting the confidence interval for EDF estimates	PROC SEVSELECT	EDFALPHA=
Limits or suppresses the display of class levels	PROC SEVSELECT	NOCLPRINT
Suppresses all displayed and graphical output	PROC SEVSELECT	NOPRINT
Specifies which graphical output to prepare	PROC SEVSELECT	PLOTS=
Specifies which graphical output to prepare about the selection process	SELECTION	PLOTS=
Specifies which output to display	PROC SEVSELECT	PRINT=

PROC SEVSELECT Statement

PROC SEVSELECT *options* ;

The PROC SEVSELECT statement invokes the procedure. You can specify two types of *options* in the PROC SEVSELECT statement. One set of *options* controls input and output. The other set of *options* controls the model estimation and selection process.

The following *options* control the input data tables used by PROC SEVSELECT and various forms of output generated by PROC SEVSELECT. The *options* are listed in alphabetical order.

COVOUT

specifies that the OUTEST= data table contain the estimate of the covariance structure of the parameters. This option has no effect if you do not specify the OUTEST= option. For more information about how the covariance is reported in the OUTEST= data table, see the section “[OUTEST= Data Table](#)” on page 924.

CRITERION=*criterion-option*

CRITERIA=*criterion-option*

CRIT=*criterion-option*

specifies the criterion to report in the model selection table.

If you specify two or more candidate models for estimation, then the one that has the best value for the specified criterion is marked as the best model in the model selection table that is displayed by default or when you specify the PRINT=SELECTION option. If you specify the OUTSTAT= data table, then the observation for the model that has the best value for the specified criterion has a value of 1 for the variable `_SELECTED_`.

You can specify one of the following *criterion-options*:

AD	specifies the Anderson-Darling (AD) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
AIC	specifies Akaike’s information criterion (AIC) as the selection criterion. A lower value is deemed better.
AICC	specifies the finite-sample corrected Akaike’s information criterion (AICC) as the selection criterion. A lower value is deemed better.
SBC BIC	specifies the Schwarz Bayesian information criterion (SBC) as the selection criterion. A lower value is deemed better.
CUSTOM	specifies the custom objective function as the selection criterion. You can specify this criterion only if you also specify the OBJECTIVE= option. A lower value is deemed better.
CVM	specifies the Cramér–von Mises (CvM) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
KS	specifies the Kolmogorov-Smirnov (KS) statistic value, which is computed by using the empirical distribution function (EDF) estimate, as the selection criterion. A lower value is deemed better.
LOGLIKELIHOOD LL	specifies $-2 * \log(L)$ as the selection criterion, where L is the likelihood of the data. A lower value is deemed better.

If you do not specify the CRITERION= option, then by default, the procedure uses CRITERION=CUSTOM if you specify the [OBJECTIVE=](#) option, or CRITERION=LOGLIKELIHOOD if you do not specify the [OBJECTIVE=](#) option.

For more information about these *criterion-options*, see the section “[Statistics of Fit](#)” on page 890.

DATA=CAS-libref.data-table

names the input data table for PROC SEVSELECT to use. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810.

data-table specifies the name of the input data table.

EDFALPHA=confidence-level

specifies the confidence level in the range (0,1) to use for computing the confidence intervals for the EDF estimates. The lower and upper confidence limits that correspond to this level are displayed in the plot that is created when you specify the PLOTS=CDFPERDIST option.

If you do not specify the EDFALPHA= option, then PROC SEVSELECT uses a default value of 0.05.

INEST=CAS-libref.data-table

names the input data table that contains the initial values of the parameter estimates to start the optimization process.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

The initial values that you specify in the INIT= option in the DIST statement take precedence over any initial values that you specify in the INEST= data table. For more information about the variables in this data table, see the section “[INEST= Data Table](#)” on page 923.

NOCLPRINT<=number>

suppresses the display of the “Class Level Information” table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps reduce the size of the “Class Level Information” table if some classification variables have a large number of levels. This option has no effect if you do not specify the CLASS statement.

NOPRINT

turns off all displayed and graphical output. If you specify this option, then any value that you specify for the PRINT= option is ignored.

OUTEST=CAS-libref.data-table

names the output data table to contain estimates of the parameter values and their standard errors for each model whose parameter estimation process converges.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

For more information about the variables in this data table, see the section “[OUTEST= Data Table](#)” on page 924.

OUTMODELINFO=CAS-libref.data-table

names the output data table to contain the information about each candidate distribution.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

For more information about the variables in this data table, see the section “[OUTMODELINFO= Data Table](#)” on page 927.

OUTSTAT=CAS-libref.data-table

names the output data table to contain the values of statistics of fit for each model whose parameter estimation process converges.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

For more information about the variables in this data table, see the section “[OUTSTAT= Data Table](#)” on page 927.

OUTSTORE=CAS-libref.data-table

names the output data table to contain the context and results of the severity model estimation process in the form of an item store, which has a binary format that cannot be modified. You can specify this item store data table in a subsequent PROC CCDDM step by using the SEVERITYSTORE= option.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “[Using CAS Sessions and CAS Engine Librefs](#)” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the [DATA=](#) option.

PLOTS <(global-plot-options)> <=(plot-request-option)>

PLOTS <(global-plot-options)> <=(plot-request-option ... plot-request-option)>

specifies which graphical output to create. If you specify more than one *global-plot-option*, then separate them with spaces and enclose them in parentheses. If you specify more than one *plot-request-option*, then separate them with spaces and enclose them in parentheses.

When the active CAS session contains more than one worker node, PROC SEVSELECT prepares the plots by using an aggregation algorithm that merges the plot data from individual worker nodes. Each worker node in turn prepares its plot data by using a sample of the local data to compute the empirical quantiles for a uniform grid of points along the empirical distribution function (EDF) axis; this data sample is the same sample that it uses to compute the EDF-based fit statistics. So the plots are approximate. You can still use them to compare the fits of different distributions visually. However, to get a more accurate visual comparison, it is recommended that you request plots only when you run the PROC SEVSELECT step in a CAS session that uses no more than one worker node.

You can specify the following *global-plot-options*:

HISTOGRAM

plots the histogram of the response variable on the probability density function (PDF) plots.

KERNEL

plots the kernel estimate of the probability density of the response variable on the PDF plots.

You can specify the following *plot-request-options*:

ALL

creates all the graphical output.

CDF

creates a plot that compares the cumulative distribution function (CDF) estimates of all the candidate distribution models to the empirical distribution function (EDF) estimate. The plot does not contain CDF estimates for models whose parameter estimation process does not converge.

CDFPERDIST

creates a plot of the CDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

CONDITIONALPDF <(*cpdf-options*)>**CONDPDF** <(*cpdf-options*)>

creates a plot that compares the conditional probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain conditional PDF estimates for models whose parameter estimation process does not converge.

A conditional PDF of a loss random variable Y in an interval $(Y_l, Y_r]$ is the probability that a specific loss value is observed, given that the loss values belong to that interval. Formally, the conditional PDF of y , denoted by $f^c(y)$, for the $(Y_l, Y_r]$ interval is defined as $f^c(y) = \Pr[Y = y | Y_l < Y \leq Y_r]$. If $f(y)$ and $F(y)$ denote the PDF and CDF, respectively, at loss value y , then $f^c(y)$ for the $(Y_l, Y_r]$ interval is computed as $f^c(y) = f(y)/(F(Y_r) - F(Y_l))$. The scaling factor of $1/(F(Y_r) - F(Y_l))$ ensures that the conditional PDF is a true PDF that integrates to 1 in the $(Y_l, Y_r]$ interval.

The conditional PDF comparison plot contains at most three regions (intervals) of mutually exclusive ranges of the loss variable's value,

- left-tail: $(y_{\min} - \epsilon, L]$
- center: $(L, R]$
- right-tail: $(R, y_{\max}]$

where $y_{\min}$ and $y_{\max}$ denote the smallest and largest values, respectively, of the loss variable in the DATA= data table and ϵ denotes a small machine-precision constant for a double-precision value.

You can specify the following *cpdf-options* to control how the values of L and R are computed and which regions are displayed:

LEFTQ=number**LEFT=number****L=number**

specifies a *number*, which must be between 0 and 1, that sets the CDF value at the end of the left-tail region. The left-tail region always starts at the minimum loss variable value in the DATA= data table. If *number* is p_l , then PROC SEVSELECT sets L (the end of the left-tail region) equal to the 100 p_l th percentile, which is an empirical estimate of $Q(p_l)$, where Q denotes the quantile function.

RIGHTQ=number**RIGHT=number****R=number**

specifies a *number*, which must be between 0 and 1, that sets the CDF value at the start of the right-tail region. The right-tail region always ends at the maximum loss variable value in the DATA= data table. If *number* is p_r , then PROC SEVSELECT sets R (the start of the right-tail region) equal to the 100 p_r th percentile, which is an empirical estimate of $Q(p_r)$, where Q denotes the quantile function.

SHOWREGION=(region-options)**SHOW=(region-options)**

specifies one or more regions to display in the plot. You can specify any combination of the following *region-options*. If you specify more than one *region-option*, you must enclose the *region-options* in parentheses.

CENTER C	displays the center region of the plot, which is the region between the end of the left-tail region and the beginning of the right-tail region. If you specify this option, you must also specify valid values for both the LEFTQ= and RIGHTQ= options.
LEFT L	displays the left-tail region of the plot. If you specify this option, you must also specify a valid value for the LEFTQ= option.
RIGHT R	displays the right-tail region of the plot. If you specify this option, you must also specify a valid value for the RIGHTQ= option.

If you do not specify the SHOWREGION option, then PROC SEVSELECT determines the default displayed regions as follows:

- If you do not specify either the **LEFTQ=** or **RIGHTQ=** option, then this is equivalent to specifying (LEFTQ=0.25 RIGHTQ=0.75), and PROC SEVSELECT displays all three regions (left-tail, center, and right-tail).
- If you specify valid values for both the **LEFTQ=** and **RIGHTQ=** options, then PROC SEVSELECT displays all three regions (left-tail, center, and right-tail).
- If you specify a valid value for the **LEFTQ=** option but do not specify the **RIGHTQ=** option, then PROC SEVSELECT displays two regions: the left-tail region and the remaining region that combines the center and right-tail regions.
- If you specify a valid value for the **RIGHTQ=** option but do not specify the **LEFTQ=** option, then PROC SEVSELECT displays two regions: the right-tail region and the remaining region that combines the center and left-tail regions.

NOTE: PROC SEVSELECT computes the percentile estimates, which serve as region boundaries L and R , by using the EDF estimates that it computes by using the EDF estimation process as described in the section “[Empirical Distribution Function Estimation Methods](#)” on page 883. For large input data tables or distributed mode of execution, the EDF estimates are approximate; therefore, the percentile estimates are also approximate—that is, they are not the same as the percentile estimates that would have been computed if the entire input data were used.

CONDITIONALPDFPERDIST *<(cpdf-options)>*

CONDPDFDIST *<(cpdf-options)>*

creates a plot of the conditional PDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

The *cpdf-options* are identical to those listed for the **CONDITIONALPDF** plot option, except that they are interpreted in the context of each candidate distribution individually. You can specify a different set of values for the *cpdf-options* in the **CONDITIONALPDFPERDIST** option than you specify in the **CONDITIONALPDF** option.

NONE

creates none of the graphical output. If you specify this option, then it overrides all the other *plot-request-options*. The default graphical output is also suppressed.

PDF

creates a plot that compares the probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain PDF estimates for models whose parameter estimation process does not converge.

PDFPERDIST

creates a plot of the PDF estimates of each candidate distribution model. A plot is not created for models whose parameter estimation process does not converge.

PP

creates the probability-probability plot (known as the P-P plot), which compares the CDF estimate of each candidate distribution model to the empirical distribution function (EDF). The data that are shown in this plot are used for computing the EDF-based statistics of fit.

QQ

creates the quantile-quantile plot (known as the Q-Q plot), which compares the empirical quantiles to the quantiles of each candidate distribution model.

PRINT *<(global-display-options)> <=display-option>*

PRINT *<(global-display-options)> <= (display-option ... display-option) >*

specifies which output to display. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses.

You can specify the following *global-display-options*:

ONLY

turns off the default displayed output and displays only the requested output.

SYMMETRIC=(*< symmetric-matrix-print-options >*)**SYMM=**(*< symmetric-matrix-print-options >*)

controls the contents of tables that display a symmetric matrix. You can specify the following *symmetric-matrix-print-options*:

SHAPE=*shape*

specifies the shape of the symmetric matrix. You can specify one of the following *shape* values:

FULL F	displays the full matrix.
LOWER L	displays the lower triangular matrix.
UPPER U	displays the upper triangular matrix.

By default, SHAPE=FULL.

NOUNITDIAG | ND

turns off the display of the diagonal for symmetric matrices that have a unit diagonal. By default, PROC SEVSELECT displays the unit diagonal.

If you do not specify the SYMMETRIC= option, then the procedure uses SYMMETRIC=(SHAPE=FULL) by default.

You can specify the following *display-options*:

ALL

displays all the output.

ALLFITSTATS

displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge.

CONVSTATUS

displays the convergence status of the parameter estimation process.

CORRB**CORR**

displays the estimated correlation matrix of parameters. You can control the display of this matrix by specifying the SHAPE= and NOUNITDIAG suboptions of the SYMMETRIC= *global-display-option*.

COVB**COV**

displays the estimated variance-covariance matrix of parameters. You can control the display of this matrix by specifying the SHAPE= suboption of the SYMMETRIC= *global-display-option*.

DESCSTATS

displays the descriptive statistics for the response variable. If you specify the SCALEMODEL statement, then this option also displays the descriptive statistics for the regression effects that do not contain a CLASS variable.

DISTINFO

displays the information about each specified distribution. For each distribution, the information includes the name, description, validity status, and number of distribution parameters.

ESTIMATES**PARMEST**

displays the final estimates of parameters. The estimates are not displayed for models whose parameter estimation process does not converge.

ESTIMATIONDETAILS

displays the details of the estimation process for all the models in one table.

INITIALVALUES

displays the initial values and bounds used for estimating each model.

NLOHISTORY

displays the iteration history of the nonlinear optimization process used for estimating the parameters.

NLOSUMMARY

displays the summary of the nonlinear optimization process used for estimating the parameters.

NONE

displays none of the output. If you specify this option, then it overrides all other display options. The default displayed output is also suppressed.

SELECTION**SELECT**

displays the model selection table.

STATISTICS**FITSTATS**

displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

If you do not specify the PRINT= option or if you do not specify the ONLY *global-display-option*, then the default displayed output is equivalent to specifying PRINT=(SELECTION CONVSTATUS NLOSUMMARY STATISTICS ESTIMATES).

SELECTOUT

specifies that only the regression parameters that correspond to the selected effects be written to the OUTEST= data table. If you omit this option, then the special missing value .N is written for such regression parameters. For more information, see the section “[OUTEST= Data Table](#)” on page 924.

This option has no effect if you do not specify the SCALEMODEL and SELECTION statements.

VARDEF=DF | N

specifies the denominator to use for computing the covariance estimates. You can specify one of the following values:

- DF** specifies that the number of nonmissing observations minus the model degrees of freedom (number of parameters) be used.
- N** specifies that the number of nonmissing observations be used.

For more information about the covariance estimation, see the section “[Estimating Covariance and Standard Errors](#)” on page 876.

ZEROEST

specifies that zero be written to the OUTEST= data table as an estimate of the regression parameter that is not in the final model (because it is either collinear or not selected). If you omit this option, then the special missing value .R or .N is written for such regression parameters. For more information, see the section “[OUTEST= Data Table](#)” on page 924.

This option has no effect if you do not specify the SCALEMODEL statement.

The following *options* control the model estimation and selection process:

EMPIRICALCDF | EDF=method

specifies the method to use for computing the nonparametric or empirical estimate of the cumulative distribution function of the data. You can specify one of the following values for *method*:

AUTOMATIC | AUTO

specifies that the method be chosen automatically based on the data specification.

If you do not specify any censoring or truncation, then the standard empirical estimation method (STANDARD) is chosen. If you specify both right-censoring and left-censoring, then Turnbull’s estimation method (TURNBULL) is chosen. For all other combinations of censoring and truncation, the Kaplan-Meier method (KAPLANMEIER) is chosen.

KAPLANMEIER | KM

specifies that the product limit estimator proposed by Kaplan and Meier (1958) be used. Specification of this method has no effect when you specify both right-censoring and left-censoring.

MODIFIEDKM | MKM <(options)>

specifies that the modified product limit estimator be used. Specification of this method has no effect when you specify both right-censoring and left-censoring.

This method allows Kaplan-Meier’s product limit estimates to be more robust by ignoring the contributions to the estimate due to small risk-set sizes. The risk set is the set of observations at the risk of failing, where an observation is said to fail if it has not been processed yet and might experience censoring or truncation. You can specify the minimum risk-set size that makes it eligible to be included in the estimation either as an absolute lower bound on the size (RSLB= option) or a relative lower bound determined by the formula cn^α proposed by Lai and Ying (1991). You can specify the values of c and α by using the C= and ALPHA= options, respectively. By default, the relative lower bound is used with values of $c = 1$ and $\alpha = 0.5$. However, you can modify the default by using the following *options*:

ALPHA | A=number

specifies the value to use for α when the lower bound on the risk set size is defined as cn^α . This value must satisfy $0 < \alpha < 1$.

C=number

specifies the value to use for c when the lower bound on the risk set size is defined as cn^α . This value must satisfy $c > 0$.

RSLB=number

specifies the absolute lower bound on the risk set size to be included in the estimate.

NOTURNBULL

specifies that the method be chosen automatically based on the data specification and that Turnbull's method not be used. This option is the default.

This method first replaces each left-censored or interval-censored observation with an uncensored observation. If the resulting set of observations has any truncated or right-censored observations, then the Kaplan-Meier method (KAPLANMEIER) is chosen. Otherwise, the standard empirical estimation method (STANDARD) is chosen. The observations are modified only for the purpose of computing the EDF estimates; the modification does not affect the parameter estimation process.

STANDARD | STD

specifies that the standard empirical estimation method be used. If you specify both right-censoring and left-censoring, then the specification of this method has no effect. If you specify any other combination of censoring or truncation effects, then this method ignores such effects, and can thus result in estimates that are more biased than those obtained with other methods that are more suitable for censored or truncated data.

TURNBULL | EM <(options)>

specifies that the Turnbull's method be used. This method is used when you specify both right-censoring and left-censoring. An iterative expectation-maximization (EM) algorithm proposed by Turnbull (1976) is used to compute the empirical estimates. If you also specify truncation, then the modification suggested by Frydman (1994) is used.

This method is used if you specify both right-censoring and left-censoring and if you explicitly specify the EMPIRICALCDF=TURNBULL option.

You can modify the default behavior of the EM algorithm by using the following *options*:

ENSUREMLE

specifies that the final EDF estimates be maximum likelihood estimates. The Kuhn-Tucker conditions are computed for the likelihood maximization problem and checked to ensure that EM algorithm converges to maximum likelihood estimates. The method generalizes the method proposed by Gentleman and Geyer (1994) by taking into account any truncation information that you might specify.

EPS=number

specifies the maximum relative error to be allowed between estimates of two consecutive iterations. This criterion is used to check the convergence of the algorithm. If you do not specify this option, then PROC SEVSELECT uses a default value of 1.0E–8.

MAXITER=number

specifies the maximum number of iterations to attempt to find the empirical estimates. If you do not specify this option, then PROC SEVSELECT uses a default value of 500.

ZEROPROB=number

specifies the threshold below which an empirical estimate of the probability is considered zero. This option is used to decide if the final estimate is a maximum likelihood estimate. This option does not have an effect if you do not specify the ENSUREMLE option. If you specify the ENSUREMLE option, but do not specify this option, then PROC SEVSELECT uses a default value of 1.0E–8.

For more information about each of the methods, see the section “[Empirical Distribution Function Estimation Methods](#)” on page 883.

INITSAMPLE (*initsample-option*)**INITSAMPLE** (*initsample-option* ... *initsample-option*)

specifies that a sample of the input data be used for initializing the distribution parameters. If you specify more than one *initsample-option*, then separate them with spaces.

When you do not specify initial values for the distribution parameters, PROC SEVSELECT needs to compute the empirical distribution function (EDF) estimates as part of the default method for parameter initialization. The EDF estimation process can be expensive, especially when you specify censoring or truncation effects for the loss variable. Furthermore, it is not amenable to parallelism due to the sequential nature of the algorithm for truncation effects. You can use the INITSAMPLE option to specify that only a fraction of the input data be used in order to reduce the time taken to compute the EDF estimates. PROC SEVSELECT uses the uniform random sampling method to select the sample, the size and randomness of which are controlled by the following *initsample-options*:

FRACTION=number

specifies the fraction, between 0 and 1, of the input data to be used for sampling.

SEED=number

specifies the seed to be used for the uniform random number generator. This option enables you to select the same sample from the same input data across different runs of PROC SEVSELECT, which can be useful for replicating the results across different runs. If you do not specify the seed value, PROC SEVSELECT generates a seed that is based on the system clock.

SIZE=number

specifies the size of the sample. If the data are distributed across different nodes, then this size applies to the sample that is prepared at each node. For example, let the input data table of size 100,000 observations be distributed across 10 nodes such that each node has 10,000 observations. If you specify SIZE=1000, then each node computes a local EDF estimate by using a sample of size 1,000 selected randomly from its 10,000 observations. If you specify both of the SIZE= and FRACTION= options, then the value that you specify in the SIZE= option is used and the FRACTION= option is ignored.

If you do not specify the INITSAMPLE option, then a uniform random sample of at most 10,000 observations is used for EDF estimation on each node that has observations.

NOBALANCETHREADS

NOBALTHR

specifies that the observations do not need to be balanced among threads of computation.

Starting with SAS Econometrics 8.5, the parameter estimation phase balances the observations among all threads of computation by default; this helps achieve significantly faster estimation times. However, this balanced processing can cause small numerical differences compared to the prior releases that did not balance observations among threads of computation. You should specify this option only if you want to restore the numerical behavior of releases prior to 8.5. However, use of this option can lead to a significantly slower estimation phase in most cases, and therefore it is strongly discouraged.

NOCONSTFITSTATS

NOCONSTSOF

excludes the constant distribution parameters, if any are specified, from the calculations of likelihood-based fit statistics (AIC, AICC, and SBC) that depend on the number of model parameters.

By default, constant distribution parameters are assumed to be estimable (even if PROC SEVSELECT does not estimate them) and are included in the calculations of the likelihood-based fit statistics.

OBJECTIVE=*symbol-name*

names the symbol that represents the objective function in the SAS programming statements that you specify. For each model to be estimated, PROC SEVSELECT executes the programming statements to compute the value of this symbol for each observation. The values are added across all observations to obtain the value of the objective function. The optimization algorithm estimates the model parameters such that the objective function value is *minimized*. A separate optimization problem is solved for each candidate distribution. If you specify a BY statement, then a separate optimization problem is solved for each candidate distribution within each BY group.

For more information about writing SAS programming statements to define your own objective function, see the section “[Custom Objective Functions](#)” on page 920.

SELECTNLOTECH=CONGRA | DBLDOG | NEWRAP | NMSIMP | NRRIDG | QUANEW | TRUREG

specifies the nonlinear optimization technique to use for the intermediate steps of the regression effect selection process. The can specify one of the following techniques:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization that combines a line-search algorithm with ridging.
NMSIMP	performs a Nelder-Mead simplex optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust region optimization.

This option has no effect if you do not specify the SELECTION statement.

If you omit this option, then the default optimization technique for the intermediate steps of the selection process is the same technique that PROC SEVSELECT uses to estimate the parameters of the final selected model. You can control that technique by specifying the **TECHNIQUE=** option in the **NLOPTIONS** statement.

BY Statement

BY *variable-list* ;

A BY statement can be used in the SEVSELECT procedure to process the input data table in groups of observations defined by the BY variables.

If you specify the BY statement, then unlike procedures that do not use a CAS server, PROC SEVSELECT does not need the input data table to be sorted in the order of the BY variables. As a consequence, the NOTSORTED and DESCENDING options have no effect and PROC SEVSELECT ignores them.

CLASS Statement

CLASS *variable* <(options)>... <*variable* <(options)>> </global-options> ;

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. You can list the response variable for binary models in the CLASS statement, but this is not required. Table 15.2 summarizes the values that you can use for either an *option* or a *global-option*. The options are fully documented in the section “CLASS Statement” on page 14 in Chapter 3, “Shared Concepts.”

Table 15.2 CLASS Statement Options

Option	Description
DESCENDING	Reverses the sort order
MISSING	Treats missing values as valid levels
ORDER=	Specifies the sort order for the levels
PARAM=	Specifies the parameterization of the variable
REF=	Specifies the reference level of the variable
SPLIT	Splits levels of CLASS variables into independent effects

DISPLAY Statement

DISPLAY < *table-list* > < / *options* > ;

The DISPLAY statement enables you to specify a list of display tables to display or exclude. This statement is similar to the ODS SELECT, ODS EXCLUDE, and ODS TRACE statements. However, the DISPLAY statement can improve performance when a large number of tables could be generated (such as in BY-group processing). The procedure processes the DISPLAY statement on a CAS server and thus sends only a subset of ODS tables to the SAS client. Because ODS statements are processed on a SAS client, first all the generated display tables are sent to the client, and then the client creates a subset.

If you use both DISPLAY and ODS statements together, the DISPLAY statement takes precedence over the ODS statements. Note that the ODS EXCLUDE statement processes tables that are sent to the client after they have been filtered by the DISPLAY statement. In some cases, it might appear that the ODS EXCLUDE statement is taking precedence because it can further filter the tables. For more information about ODS, see *SAS Output Delivery System: Procedures Guide*.

You can specify the *table-list* as a list of table names, paths, partial pathnames, and regular expressions.

The table names that you can specify are listed in the section “[Displayed Output](#)” on page 929. A path is a table name that is prefixed with dot-separated grouping information. For example, a SelectionSummary table that a procedure produces during a selection routine might have the path *Bygroup1.Summary.SelectionSummary*. A partial pathname does not include all groups; for example, *SelectionSummary* and *Summary.SelectionSummary* are partial pathnames for *Bygroup1.Summary.SelectionSummary*.

When you specify a table name or partial pathname, all display tables whose paths end in the specified name are selected for display or exclusion. For example, both *SelectionSummary* and *Summary.SelectionSummary* select *Bygroup1.Summary.SelectionSummary*.

A regular expression is enclosed in forward slashes (/). For example, specifying “/tions/” selects all pathnames that contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is selected. Specifying “!/tions/” selects all pathnames that do not contain the substring “tions”; in particular, the *Bygroup1.Summary.SelectionSummary* table is not selected.

You can specify the following *options* after a slash (/):

CASESENSITIVE

performs a case-sensitive comparison of table names in the *table-list* to display table names when tables are subsetting for display. To preserve case, you must enclose table names in the *table-list* in quotation marks.

EXCLUDE

displays all display tables except those that you specify in the *table-list*.

EXCLUDEALL

suppresses display of all tables. This option takes precedence over the other options.

TRACE

displays the display table names, labels, and paths.

DISPLAYOUT Statement

DISPLAYOUT *table-spec-list* < / *options* > ;

The DISPLAYOUT statement enables you to create CAS output tables from your displayed output. This statement is similar to the ODS OUTPUT statement. For more information about ODS, see [SAS Output Delivery System: Procedures Guide](#).

The *table-spec-list* specifies a list of CAS output tables to create. Each entry in the list has either a *key=value* format or a *key* format:

key=value specifies *key* as the ODS table name, path, or partial pathname, and specifies *value* as the CAS output table name.

key specifies *key* as the ODS table name and also as the CAS output table name.

The ODS table names that you can specify are listed in the section “[Displayed Output](#)” on page 929. You cannot specify the ODS table named OutputCasTables in the *table-spec-list*.

Table names and partial pathnames are discussed under the [DISPLAY](#) statement. The DISPLAYOUT statement does not support regular expressions.

You can specify the following *options* after a slash (/):

INCLUDEALL

creates output CAS tables for all display tables. The name of the created output CAS table is the same as the corresponding display table name. If you specify this option, the *table-spec-list* specification is ignored.

NOREPLACE

does not replace any existing CAS output table of the same name.

REPEATED

replicates all CAS output tables on all nodes.

DIST Statement

DIST *distribution-name-or-keyword* < (*distribution-option*) < *distribution-name-or-keyword* < (*distribution-option*) > > ... > < / *options* > ;

The DIST statement specifies candidate distributions to be estimated by the SEVSELECT procedure. You can specify multiple DIST statements, and each statement can contain one or more distribution specifications.

For your convenience, PROC SEVSELECT provides the following 10 different predefined distributions (the name in parentheses is the name to use in the DIST statement): Burr (BURR), exponential (EXP), gamma (GAMMA), generalized Pareto (GPD), inverse Gaussian or Wald (IGAUSS), lognormal (LOGN), Pareto (PARETO), Tweedie (TWEEDIE), scaled Tweedie (STWEEDIE), and Weibull (WEIBULL). These are described in detail in the section “[Predefined Distributions](#)” on page 863.

You can specify any of the predefined distributions or any distribution that you have defined. If a distribution that you specify is not a predefined distribution, then you must submit the CMPLIB= system option with appropriate libraries before you submit the PROC SEVSELECT step to enable the procedure to find the functions associated with your distribution. The predefined distributions are defined in the Sashelp.Svrtldist library. However, you are not required to specify this library in the CMPLIB= system option. For more information about defining your own distributions, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895.

As a convenience, you can also use a shortcut keyword to indicate a list of distributions. You can specify one or more of the following keywords:

ALL

specifies all the predefined distributions and the distributions that you have defined in the libraries that you specify in the CMPLIB= system option. In addition to the eight predefined distributions included by the PREDEFINED keyword, this list also includes the Tweedie and scaled Tweedie distributions that are defined in the Sashelp.Svrtldist library.

PREDEFINED

specifies the list of eight predefined distributions: BURR, EXP, GAMMA, GPD, IGAUSS, LOGN, PARETO, and WEIBULL. Although the TWEEDIE and STWEEDIE distributions are available in the Sashelp.Svrtldist library along with these eight distributions, they are not included by this keyword. If you want to fit the TWEEDIE and STWEEDIE distributions, then you must specify them explicitly or use the ALL keyword.

USER

specifies the list of all the distributions that you have defined in the libraries that you specify in the CMPLIB= system option. This list does not include the distributions defined in the Sashelp.Svrtldist library, even if you specify the Sashelp.Svrtldist library in the CMPLIB= option.

The use of these keywords, especially ALL, can result in a large list of distributions, which might take a longer time to estimate. A warning is printed to the SAS log if the number of total distribution models to estimate exceeds 10.

The following *distribution-option* values can be used in the DIST statement for a distribution name that is not a shortcut keyword:

INIT=(name=value ... name=value)

specifies the initial values to be used for the distribution parameters to start the parameter estimation process. You must specify the values by parameter names, and the parameter names must match the names used in the model definition. For example, let a model M's definition contain an M_PDF function with the following signature:

```
function M_PDF(x, alpha, beta);
```

For this model, the names **alpha** and **beta** must be used for the INIT option. The names are case-insensitive. If you do not specify initial values for some parameters in the INIT statement, then a default value of 0.001 is assumed for those parameters. If you specify an incorrect parameter, PROC SEVSELECT prints a warning to the SAS log and does not fit the model. All specified values must be nonmissing.

If you are modeling regression effects, then the initial value of the first distribution parameter (**alpha** in the preceding example) should be the initial *base* value of the scale parameter or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 878.

The use of INIT= option is one of the three methods available for initializing the parameters. For more information, see the section “[Parameter Initialization](#)” on page 877. If none of the initialization methods is used, then PROC SEVSELECT initializes all parameters to 0.001.

You can specify the following *options* in the DIST statement:

INFUNCDEF="CAS-table-name"

INTAB="CAS-table-name"

INFUNCDEF=CAS-table-name<(CASLIB="caslib")>

INTAB=CAS-table-name<(CASLIB="caslib")>

specifies a data table on the CAS server that contains the distribution function definitions. If you use the first form to specify the data table, then PROC SEVSELECT assumes that the table is in the caslib that is active in the current CAS session. To specify a different caslib, use the second form.

LISTONLY

prints the list of all candidate distributions to the SAS log without doing any further processing on them. This option is especially useful when you use a shortcut keyword to include a list of distributions. It enables you to find out which distributions are included by the keyword.

OUTFUNCDEF="CAS-table-name" <option>

OUTTAB="CAS-table-name" <option>

specifies the output data table on the CAS server to write the distribution function definitions to. The table is created in the caslib that is active for the current CAS session.

This option is useful for creating a data table that you can use to invoke the actions in the **severity** action set from the PROC CAS, Lua, or Python client.

You can specify the following *option*:

GLOBAL**PROMOTED**

promotes the data table to make it available globally.

If you do not specify this option, the table is session-local, which means it is accessible only to the current CAS session that is executing the PROC SEVSELECT step. When the current CAS session is terminated, the OUTFUNCDEF= table is deleted.

When you specify this option, the table persists in the memory of the CAS server and is available to other CAS sessions that you might start after the CAS session that is executing the current PROC SEVSELECT step is terminated.

If you do not specify the OUTFUNCDEF= or INFUNCDEF= option, then PROC SEVSELECT creates a session-local data table named svrtdist in the caslib that is active for the current CAS session.

VALIDATEONLY**VALIDATEONLY (NOSCALEMODEL)**

checks all candidate distributions for validity without doing any further processing on them. The first form checks whether each distribution's first parameter is a scale parameter, which is a requirement for fitting a scale regression model for that distribution. You can disable that check by using the second form, which specifies the NOSCALEMODEL option; this is useful if you do not want to fit a scale regression model for a distribution.

PROC SEVSELECT writes the distribution information to the DistributionInfo ODS table.

This option is especially useful when you specify your own distributions or when you specify the `_USER_` or `_ALL_` keyword in the DIST statement. It enables you to check whether your custom distribution definitions satisfy PROC SEVSELECT's requirements for the specified modeling task. It is recommended that you specify the SCALEMODEL statement if you intend to fit a model with regression effects, because the SCALEMODEL statement instructs PROC SEVSELECT to perform additional checks to validate whether regression effects can be modeled on each candidate distribution.

This option is also useful in conjunction with the `OUTFUNCDEF=` option when you define or update a custom distribution and want to write its definition to the CAS table that you can use to invoke the actions in the **severity** action set from the PROC CAS, Lua, or Python client.

EFFECT Statement

EFFECT *name=effect-type* (*variables* < / *options* >) ;

The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are referred to as *constructed effects* to distinguish them from the usual model effects that are formed from continuous or classification variables, as discussed in the section “GLM Parameterization of Classification Variables and Effects” on page 52 in Chapter 3, “Shared Concepts.”

You can specify the following *effect-types*:

COLLECTION

specifies a collection effect that defines one or more variables as a single effect that has multiple degrees of freedom. The variables in a collection are considered as a unit for purposes of estimation and inference.

MULTIMEMBER MM	specifies a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement.
POLYNOMIAL POLY	specifies a multivariate polynomial effect in the specified numeric variables.
SPLINE	specifies a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables.

Table 15.3 summarizes the *options* available in the EFFECT statement.

Table 15.3 EFFECT Statement Options

Option	Description
Collection Effects Options	
DETAILS	Displays the constituents of the collection effect
Multimember Effects Options	
DETAILS	Displays the levels of the multimember effect
NOEFFECT	Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns
STDIZE	Standardizes the design matrix entries so that each observation has a sum of 1
WEIGHT=	Specifies the weight variable for the contributions of each classification effect
Polynomial Effects Options	
DEGREE=	Specifies the degree of the polynomial
DETAILS	Displays details of the specified polynomial
MDEGREE=	Specifies the maximum degree of any variable in a term of the polynomial
NOSEPARATE	Treats the polynomial as a single effect with multiple degrees of freedom
STANDARDIZE=	Specifies centering and scaling suboptions for the variables that define the polynomial

Table 15.3 *continued*

Option	Description
Spline Effects Options	
BASIS=	Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect
DATABOUNDARY	Uses the extremes of the data as boundary knots for a B-spline basis
DEGREE=	Specifies the degree of the spline effect
DETAILS	Displays the knots and locations for each spline basis function
KNOTMAX=	Requests equally spaced right-side boundary knots starting at the variables' maximum and ending at the KNOTMAX= value
KNOTMETHOD=	Specifies how to construct the knots for the spline effect
KNOTMIN=	Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables' minimum value
NATURALCUBIC	Specifies a natural cubic spline basis for the spline effect
SEPARATE	Treats the spline basis for each variable as a separate effect when multiple variables are specified
SPLIT	Treats each design matrix column as a separate effect for selection methods

For more information about the syntax of these *effect-types* and how columns of constructed effects are computed, see the section “**EFFECT Statement**” on page 20 in Chapter 3, “**Shared Concepts**.”

LOSS Statement

LOSS < *response-variable-name* > < / *censoring-truncation-options* > ;

The LOSS statement specifies the name of the response or loss variable whose distribution needs to be modeled. You can also specify additional options to indicate any truncation or censoring of the response. The specification of response variable is optional if you specify at least one type of censoring. You must specify a response variable if you do not specify any censoring. If you specify more than one LOSS statement, then the first statement is used.

All the analysis variables that you specify in this statement must be present in the input data table that you specify by using the DATA= option in the PROC SEVSELECT statement. The response variable is expected to have nonmissing values. If the variable has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

The following *censoring-truncation-options* can be used in the LOSS statement:

LEFTCENSORED | **LC=***variable-name*

LEFTCENSORED | **LC=***number*

specifies the left-censoring variable or a global left-censoring limit.

You can use the *variable-name* argument to specify a data table variable that contains the left-censoring limit. If the value of this variable is missing, then PROC SEVSELECT assumes that such observations are not left-censored.

Alternatively, you can use the *number* argument to specify a left-censoring limit value that applies to all the observations in the data table. This limit must be a nonzero positive number.

By the definition of left-censoring, an exact value of the response is not known when it is less than or equal to the left-censoring limit. If you specify the response variable and the value of that variable is less than or equal to the value of the left-censoring limit for some observations, then PROC SEVSELECT treats such observations as left-censored and the value of the response variable is ignored. If you specify the response variable and the value of that variable is greater than the value of the left-censoring limit for some observations, then PROC SEVSELECT assumes that such observations are not left-censored and the value of the left-censoring limit is ignored.

If you specify both right-censoring and left-censoring limits, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about left-censoring, see the section “[Censoring and Truncation](#)” on page 873.

LEFTTRUNCATED | **LT=***variable-name* < (*left-truncation-option*) >

LEFTTRUNCATED | **LT=***number* < (*left-truncation-option*) >

specifies the left-truncation variable or a global left-truncation threshold.

You can use the *variable-name* argument to specify a data table variable that contains the left-truncation threshold. If the value of this variable is missing or 0 for some observations, then PROC SEVSELECT assumes that such observations are not left-truncated.

Alternatively, you can use the *number* argument to specify a left-truncation threshold that applies to all the observations in the data table. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of left-truncation, you can observe only a value that is greater than the left-truncation threshold. If a response variable value is less than or equal to the left-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about left-truncation, see the section “[Censoring and Truncation](#)” on page 873.

You can specify the following *left-truncation-option* for an alternative interpretation of the left-truncation threshold:

PROBOBSERVED | **POBS=***number*

specifies the probability of observability, which is defined as the probability that the underlying severity event is observed (and recorded) for the specified left-threshold value.

The specified *number* must lie in the (0.0, 1.0] interval. A value of 1.0 is equivalent to specifying that there is no left-truncation, because it means that no severity events can occur with a value less than or equal to the threshold. If you specify value of 1.0, PROC SEVSELECT prints a warning to the SAS log and proceeds by assuming that LEFTTRUNCATED= option is not specified.

For more information, see the section “[Probability of Observability](#)” on page 874.

RIGHTCENSORED | **RC=***variable-name*

RIGHTCENSORED | **RC=***number*

specifies the right-censoring variable or a global right-censoring limit.

You can use the *variable-name* argument to specify a data table variable that contains the right-censoring limit. If the value of this variable is missing, then PROC SEVSELECT assumes that such observations are not right-censored.

Alternatively, you can use the *number* argument to specify a right-censoring limit value that applies to all the observations in the data table. This limit must be a nonzero positive number.

By the definition of right-censoring, an exact value of the response is not known when it is greater than or equal to the right-censoring limit. If you specify the response variable and the value of that variable is greater than or equal to the value of the right-censoring limit for some observations, then PROC SEVSELECT treats such observations as right-censored and the value of the response variable is ignored. If you specify the response variable and the value of that variable is less than the value of the right-censoring limit for some observations, then PROC SEVSELECT assumes that such observations are not right-censored and the value of the right-censoring limit is ignored.

If you specify both right-censoring and left-censoring limits, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about right-censoring, see the section “[Censoring and Truncation](#)” on page 873.

RIGHTTRUNCATED | **RT=***variable-name*

RIGHTTRUNCATED | **RT=***number*

specifies the right-truncation variable or a global right-truncation threshold.

You can use the *variable-name* argument to specify a data table variable that contains the right-truncation threshold. If the value of this variable is missing for some observations, then PROC SEVSELECT assumes that such observations are not right-truncated.

Alternatively, you can use the *number* argument to specify a right-truncation threshold that applies to all the observations in the data table. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of right-truncation, you can observe only a value that is less than or equal to the right-truncation threshold. If a response variable value is greater than the right-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about right-truncation, see the section “[Censoring and Truncation](#)” on page 873.

NLOPTIONS Statement

NLOPTIONS *options* ;

The SEVSELECT procedure uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization of the likelihood function to obtain the estimates of distribution and regression parameters. You can use the NLOPTIONS statement to control different aspects of this optimization process. If you specify more than one NLOPTIONS statement, then the first statement is used.

For most problems, the default settings of the optimization process are adequate. However, in some cases it might be useful to change the optimization technique or to change the maximum number of iterations. The following statement uses the MAXITER= option to set the maximum number of iterations to 200 and uses the TECH= option to change the optimization technique to the double-dogleg optimization (DBLDOG) rather than the default technique, the trust region optimization (TRUREG), used in the SEVSELECT procedure:

```
nloptions tech=dbldog maxiter=200;
```

For more information about the options you can specify in the NLOPTIONS statement, see the section “[Optimization Options](#)” on page 42. For more information about the optimization methods, see the section “[Choosing an Optimization Algorithm](#)” on page 65.

OUTPUT Statement

OUTPUT < **OUT**=*CAS-libref.data-table* > *output-options* ;

The OUTPUT statement creates the output data table that contains the estimates of scoring functions and quantiles.

You must specify the following option:

OUT=*CAS-libref.data-table*

names the output data table for PROC SEVSELECT to use. You must specify this option before any other options. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to where the data table is to be stored, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about <i>CAS-libref</i> , see the section “ Using CAS Sessions and CAS Engine Librefs ” on page 810.
-------------------	---

<i>data-table</i>	specifies the name of the output data table.
-------------------	--

The *CAS-libref* must be identical to the *CAS-libref* that you specify in the **DATA**= option.

To control the contents of the OUT= data table, specify the following *output-options*:

COPYVARS=*variable-list*

specifies the names of the variables that you want to copy from the input DATA= data table to the OUT= data table. If you want to specify more than one name, then separate them by spaces.

If you specify the BY statement, then the BY variables are not automatically copied to the OUT= data table, so you must specify the BY variables in the COPYVARS= option.

FUNCTIONS=(*function*<(arg)><=*variable*> <*function*<(arg)><=*variable*>> ...)

specifies the scoring functions that you want to estimate. For each scoring function that you want to estimate, specify the suffix of the scoring function as the *function*. For each *function* that you specify in this option and for each distribution *D* that you specify in the DIST statement, the FCMP function *D_function* must be available in the search path that you specify by using the CMPLIB= system option. The signature of *D_function* must be identical to the signature of the required distribution function such as *D_CDF* or *D_LOGCDF*. For example, for the function 'FOO_BAR' to be a scoring function, you must specify the distribution 'FOO' in the DIST statement, and you must define 'FOO_BAR' in the following manner if the distribution 'FOO' has parameters named 'P1' and 'P2':

```
function FOO_BAR(y, P1, P2);
  /* Code to compute BAR by using y, P1, and P2 */
  R = <computed BAR>;
  return (R);
endsub;
```

You can then specify BAR as the *function* in the FUNCTIONS= option. For more information about the signature that defines a distribution function, see the description of the [D_CDF](#) function in the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895.

If you want to evaluate the scoring function at a specific value of the response variable, then specify a number *arg*, which is enclosed in parentheses immediately after the *function*. If you do not specify *arg* or if you specify a missing value as *arg*, then for each observation in the DATA= data table, PROC SEVSELECT computes the value *v* by using the following table and evaluates the scoring function at *v*, where *y*, *r*, and *l* denote the values of the response variable, right-censoring limit, and left-censoring limit, respectively:

Right-Censored	Left-Censored	<i>v</i>
No	No	<i>y</i>
No	Yes	<i>l</i>
Yes	No	<i>r</i>
Yes	Yes	$(l + r)/2$

You can specify the suffix of the variable that contains the estimate of the scoring function by specifying a valid SAS name as a *variable*. If you do not specify a *variable*, then PROC SEVSELECT uses *function* as the suffix of the variable name.

To illustrate the FUNCTIONS= option with an example, assume that you specify the following DIST and OUTPUT statements:

```
dist exp logn;
output out=mycas.score functions=(cdf pdf(1000)=f1000 mean);
```

Let both exponential (EXP) and lognormal (LOGN) distributions converge. If $\hat{\theta}$ is the final estimate of the scale parameter of the exponential distribution, then PROC SEVSELECT creates the following three scoring function variables for the exponential (EXP) distribution in the mycas.Score data table:

EXP_CDF	contains the CDF estimate $F_{\text{exp}}(v, \hat{\theta})$, where F_{exp} denotes the CDF of the exponential distribution and v is the value that is determined by the preceding table.
EXP_F1000	contains the PDF estimate $f_{\text{exp}}(1000, \hat{\theta})$, where f_{exp} denotes the PDF of the exponential distribution.
EXP_MEAN	contains the mean of the exponential distribution for the scale parameter $\hat{\theta}$.

Similarly, if $\hat{\mu}$ and $\hat{\sigma}$ are the final estimates of the log-scale and shape parameters of the lognormal distribution, respectively, then PROC SEVSELECT creates the following three scoring function variables for the lognormal (LOGN) distribution in the mycas.Score data table:

LOGN_CDF	contains the CDF estimate $F_{\text{logn}}(v, \hat{\mu}, \hat{\sigma})$, where F_{logn} denotes the CDF of the lognormal distribution and v is the value that is determined by the preceding table.
LOGN_F1000	contains the probability density function (PDF) estimate $f_{\text{logn}}(1000, \hat{\mu}, \hat{\sigma})$, where f_{logn} denotes the PDF of the lognormal distribution.
LOGN_MEAN	contains the mean of the lognormal distribution for the parameters $\hat{\mu}$ and $\hat{\sigma}$.

If you specify the SCALEMODEL statement, then the value of the scale parameter of a distribution depends on the values of the regression parameters. So it might be different for different observations. In this example, the values of $\hat{\theta}$ and $\hat{\mu}$ might vary by observation, which might cause the values of the EXP_F1000, EXP_MEAN, LOGN_F1000, and LOGN_MEAN variables to vary by observation. The values of the EXP_CDF and LOGN_CDF variables might vary not only because of the varying values of v but also because of the varying values of $\hat{\theta}$ and $\hat{\mu}$.

If you do not specify the SCALEMODEL statement, then the values of scoring functions for which you specify a nonmissing argument *arg* and scoring functions that do not depend on the response variable value do not vary by observation. In this example, the values of the EXP_F1000, EXP_MEAN, LOGN_F1000, and LOGN_MEAN variables do not vary by observation.

If a distribution does not converge, then the scoring function variables for that distribution contain missing values in all observations.

QUANTILES=*quantile-options*

specifies the quantiles that you want to estimate. To use this option, for each distribution that you specify in the DIST statement, the FCMP function *D_QUANTILE* must be defined in the search path that you specify by using the CMPLIB= system option.

You can specify the following *quantile-options*:

CDF=*CDF-values*

POINTS=*CDF-values*

specifies the CDF values at which you want to estimate the quantiles. *CDF-values* can be one or more numbers, separated by spaces. Each number must be in the interval (0,1).

NAMES=(*variable-names*)

specifies the suffixes of the names of the variables for each of the quantile estimates. If you specify n ($n \geq 0$) names in the *variable-names* option and k values in the CDF= option, and if $n < k$, then PROC SEVSELECT uses the n names to name the variables that correspond to the first n CDF values. For each of the remaining $k - n$ CDF values, p_i ($n < i \leq k$), PROC SEVSELECT creates a variable name P_t , where t is the text representation of $100p_i$ that is formed by retaining at most NDECIMAL= digits after the decimal point and replacing the decimal point with an underscore ('_').

NDECIMAL=*number*

specifies the number of digits to keep after the decimal point when PROC SEVSELECT creates the name of the quantile estimate variable. If you do not specify this option, then the default value is 3.

For example, assume that you specify the following DIST and OUTPUT statements:

```
dist burr;
output out=score quantiles=(cdf=0.9 0.975 0.995 names=(ninety var));
```

PROC SEVSELECT creates three quantile estimate variables, BURR_NINETY, BURR_VAR, and BURR_P99_5, in the mycas.Score data table for the Burr distribution. These variables contain the estimates of $Q_{\text{Burr}}(p, \hat{\theta}, \hat{\alpha}, \hat{\gamma})$, for $p = 0.9, 0.975$, and 0.995 , respectively, where Q_{Burr} denotes the quantile function and $\hat{\theta}$, $\hat{\alpha}$, and $\hat{\gamma}$ denote the parameter estimates of the Burr distribution.

If you specify the SCALEMODEL statement, then the quantile estimate might vary by observation, because the scale parameter of a distribution depends on the values of the regression parameters.

If you do not specify the SCALEMODEL statement, then the quantile estimates do not vary by observation, and if you do not specify any scoring functions in the FUNCTIONS= option whose estimates vary by observation, then the OUT= data table contains only one observation per BY group.

If a distribution does not converge, then the quantile estimate variables for that distribution contain missing values for all observations.

For more information about the variables and observations in the OUT= data table, see the section “[OUT= Data Table](#)” on page 924.

OUTSCORELIB Statement

OUTSCORELIB <**OUTLIB=**> *fcmp-library-name* options ;

The OUTSCORELIB statement specifies the library to write scoring functions to. Scoring functions enable you to easily compute a distribution function on the fitted parameters of the distribution without going through a potentially complex process of extracting the fitted parameter estimates from other output, such as the OUTEST= data table that PROC SEVSELECT creates.

If you specify the SCALEMODEL statement and specify interaction or classification effects, then PROC SEVSELECT ignores the OUTSCORELIB statement and does not generate scoring functions. In other words, if you specify the SCALEMODEL statement, then PROC SEVSELECT generates scoring functions if you specify only singleton continuous effects in the SCALEMODEL statement.

You must specify the following option as the first option in the statement:

OUTLIB=*fcmp-library-name*

names the FCMP library to contain the scoring functions. PROC SEVSELECT writes the scoring functions to the FCMP library named *fcmp-library-name*. If a library or data set named *fcmp-library-name* already exists, PROC SEVSELECT deletes it before proceeding.

This option is similar to the OUTLIB= option that you would specify in a PROC FCMP statement, except that *fcmp-library-name* must be a two-level name, whereas the OUTLIB= option in the PROC FCMP statement requires a three-level name. The third level of a three-level name specifies the package to which the functions belong. You do not need to specify the package name in the *fcmp-library-name*, because PROC SEVSELECT automatically creates the package for you. By default, it creates a separate package for each distribution that has not failed to converge. Each package is named for a distribution. For example, if you define and fit a distribution named *mydist*, and if *mydist* does not fail to converge, then PROC SEVSELECT creates a package named *mydist* in the OUTLIB= library that you specify. Further, suppose that the definition of the *mydist* distribution contains three distribution functions, *mydist_PDF*(*x*,*Parm1*,*Parm2*), *mydist_LOGCDF*(*x*,*Parm1*,*Parm2*), and *mydist_XYZ*(*x*,*Parm1*,*Parm2*). If you specify the OUTSCORELIB statement

```
outscorelib outlib=sasuser.scorefunc;
```

then the library Sasuser.Scorefunc contains the following three functions in a package named *mydist*: *SEV_PDF*(*x*), *SEV_LOGCDF*(*x*), and *SEV_XYZ*(*x*).

The key feature of scoring functions is that they do not require the parameter arguments (*Parm1* and *Parm2* in this example). The fitted parameter estimates are encoded inside the scoring function so that you can compute or score the value of each function for a given value of the loss variable without having to know or extract the parameter estimates through some other means.

For convenience, you can omit the OUTLIB= portion of the specification and specify only the name, as in the following example:

```
outscorelib sasuser.scorefunc;
```

When the SEVSELECT procedure runs successfully, the *fcmp-library-name* is appended to the CMPLIB system option, so you can immediately start using the scoring functions in a DATA step or PROC FCMP step. Note that the generated scoring FCMP functions can be used only in a DATA step that runs on the client machine; you cannot use them in a DATA step that runs on the CAS server.

You can specify the following *options* in the OUTSCORELIB statement:

COMMONPACKAGE

ONEPACKAGE

creates one common package to contain all the scoring functions.

If you specify this option, then all the scoring functions are created in a package called *sevfit*. For each distribution function that has the name *distribution_suffix*, the name of the corresponding scoring function is formed as *SEV_suffix_distribution*. For example, the scoring function of the distribution function 'MYDIST_BAR' is named 'SEV_BAR_MYDIST'.

If you do not specify this option, then all scoring functions for a distribution are created in a package that has the same name as the distribution, and for each distribution function that has the name *distribution_suffix*, the name of the corresponding scoring function is formed as *SEV_suffix*. For example, the scoring function of the distribution function 'MYDIST_BAR' is named 'SEV_BAR'.

OUTBYID=*CAS-libref.data-table*

names the output data table to contain the unique identifier for each BY group. This unique identifier is used as part of the name of the package or scoring function for each distribution. This is a required option when you specify a BY statement in PROC SEVSELECT.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 810. The *CAS-libref* must be identical to the *CAS-libref* that you specify in the *DATA=* option.

The OUTBYID= data table contains one observation per BY group and a variable named *_ID_* in addition to the BY variables that you specify in the BY statement. The *_ID_* variable contains the unique identifier for each BY group. The identifier of the BY group is the decimal representation of the sequence number of the BY group. The first BY group has an identifier of 1, the second BY group has an identifier of 2, the tenth BY group has an identifier of 10, and so on.

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution, PROC SEVSELECT creates as many packages as the number of BY groups. The unique BY-group identifier is used as a suffix for the package name. For example, if your *DATA=* data table has three BY groups and you specify the OUTSCORELIB statement

```
outscorelib outlib=sasuser.byscorefunc outbyid=sasuser.byid;
```

then for the distribution 'MYDIST', the library Sasuser.Byscorefunc contains the three packages 'MYDIST1', 'MYDIST2', and 'MYDIST3', and each package contains one scoring function named 'SEV_BAR' for each distribution function named 'MYDIST_BAR'.

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, PROC SEVSELECT creates as many versions of the distribution function as the number of BY groups. The unique BY-group identifier is used as a suffix for the function name. Extending the previous example, if you specify the OUTSCORELIB statement with the COMMONPACKAGE option,

```
outscorelib outlib=sasuser.byscorefunc outbyid=sasuser.byid commonpackage;
```

then for the distribution function ‘MYDIST_BAR’ of the distribution ‘MYDIST’, the Sasuser.Byscorefunc library contains the following three scoring functions: ‘SEV_BAR_MYDIST1’, ‘SEV_BAR_MYDIST2’, and ‘SEV_BAR_MYDIST3’. All the scoring functions are created in one common package named *sevfit*.

For both the preceding examples, the data table Sasuser.Byid contains three observations, one for each BY group. The value of the `_ID_` variable is 1 for the first BY group, 2 for the second BY group, and 3 for the third BY group.

For more information about scoring functions, see the section “[Scoring Functions](#)” on page 913.

SCALEMODEL Statement

```
SCALEMODEL regression-effect-list < / scalemodel-options > ;
```

The SCALEMODEL statement specifies regression effects. A regression effect is formed from one or more regressor variables according to effect construction rules. Each regression effect forms one element of $\mathbf{X}\boldsymbol{\beta}$ in the linear model structure $\mathbf{X}\boldsymbol{\beta}$ that affects the scale parameter of the distribution. The SCALEMODEL statement in conjunction with the [CLASS](#) statement supports a rich set of effects. Effects are specified by a special notation that uses regressor variable names and operators. There are two types of regressor variables: classification (or CLASS) variables and continuous variables. Classification variables can be either numeric or character and are specified in a [CLASS](#) statement. To include CLASS variables in regression effects, you must specify the CLASS statement so that it appears before the SCALEMODEL statement. A regressor variable that is not declared in the CLASS statement is assumed to be continuous. For information about effect construction rules, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

All the regressor variables must be present in the input data table that you specify by using the DATA= option in the PROC SEVSELECT statement. The scale parameter of each candidate distribution is linked to the linear predictor $\mathbf{X}\boldsymbol{\beta}$ that includes an intercept. If a distribution does not have a scale parameter, then a model based on that distribution is not estimated. If you specify more than one SCALEMODEL statement, then the first statement is used.

The regressor variables are expected to have nonmissing values. If any of the variables has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

For more information about modeling regression effects, see the section “[Estimating Regression Effects](#)” on page 878.

You can specify the following *scalemodel-options* in the SCALEMODEL statement:

```
DFMIXTURE=method-name < (method-options) >
```

specifies the method for computing representative estimates of the cumulative distribution function (CDF) and the probability density function (PDF).

When you specify regression effects, the scale of the distribution depends on the values of the regressors. For a particular distribution family, each observation in the input data table implies a different scaled version of the distribution. To compute estimates of CDF and PDF that are comparable across different

distribution families, PROC SEVSELECT needs to construct a single representative distribution from all such distributions. You can specify one of the following *method-name* values to specify the method that is used to construct the representative distribution. For more information about each of the methods, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

FULL

specifies that the representative distribution be the mixture of N distributions such that each distribution has a scale value that is implied by each of the N observations that are used for estimation. This method is the slowest.

MEAN

specifies that the representative distribution be the one-point mixture of the distribution whose scale value is computed by using the mean of the N values of the linear predictor that are implied by the N observations that are used for estimation. If you do not specify the `DFMIXTURE=` option, then this method is used by default. This is also the fastest method.

QUANTILE <(K=q)>

specifies that the representative distribution be the mixture of a fixed number of distributions whose scale values are computed by using the quantiles from the sample of N values of the linear predictor that are implied by the N observations that are used for estimation.

You can use the `K=` option to specify the number of distributions in the mixture. If you specify `K=q`, then the mixture contains $(q - 1)$ distributions such that each distribution has as its scale one of the $(q - 1)$ -quantiles.

If you do not specify the `K=` option, then PROC SEVSELECT uses the default of 2, which implies the use of a one-point mixture with a distribution whose scale value is the median of all scale values.

RANDOM <(random-method-options)>

specifies that the representative distribution be the mixture of a fixed number of distributions whose scale values are computed by using the values of the linear predictor that are implied by a randomly chosen subset of the set of all observations that are used for estimation. The same subset of observations is used for each distribution family.

You can specify the following *random-method-options* to specify how the subset is chosen:

K=r

specifies the number of distributions to include in the mixture. If you do not specify this option, then PROC SEVSELECT uses the default of 15.

SEED=number

specifies the seed that is used to generate the uniform random sample of observation indices. If you do not specify this option, then PROC SEVSELECT generates a seed internally that is based on the current value of the system clock.

INCLUDE=*n***INCLUDE=*single-effect*****INCLUDE=(*effects*)**

forces effects to be included in all models. If you specify **INCLUDE=*n***, then the first *n* effects that are listed in the **SCALEMODEL** statement are included in all models. If you specify **INCLUDE=*single-effect*** or **INCLUDE=(*single-effect*)**, then the specified effects are forced into all models. The **INCLUDE=** option has no effect if you do not specify the **SELECTION** statement.

INFORMATIVE

models missing values by using extra model effects. These effects consist of dummy variables that take the value 1 when the value of a continuous model variable involved in the effect is missing, and take the value 0 otherwise. The missing value in the original model effect is replaced by the average value of the effect for the nonmissing values. For continuous-by-class effects, such as *A***x*, where *A* is a classification variable and *x* is a continuous variable, informative missingness creates multiple dummy columns and substitutes the effect mean of *x* that corresponds to the respective level of *A*. Missing values for classification variables are treated as valid levels. For more information about informative missingness, see the section “**Informative Missingness**” on page 63 in Chapter 3, “**Shared Concepts**.”

OFFSET=*offset-variable-name*

specifies the name of the offset variable in the scale regression model. An offset variable is a regressor variable whose regression coefficient is known to be 1. For more information, see the section “**Offset Variable**” on page 879.

START=*n***START=*single-effect*****START=(*effects*)**

specifies how to begin the selection process in the FORWARD, FORWARDSWAP, and STEPWISE selection methods from the initial model that you designate. If you specify **START=*n***, then the starting model consists of the first *n* effects that are listed in the **SCALEMODEL** statement. If you specify **START=*single-effect*** or **START=(*single-effect*)**, then the starting model consists of these specified effects.

The **START=** option has no effect if you do not specify the **SELECTION** statement. This option is not supported when you specify **METHOD=BACKWARD** in the **SELECTION** statement.

SELECTION Statement

SELECTION < **METHOD=***method* < (*method-options*) > > < *options* > ;

The **SELECTION** statement performs model selection by examining whether effects should be added to or removed from the model according to rules that are defined by the selection methods. The statement is fully documented in the section “**SELECTION Statement**” on page 36 in Chapter 3, “**Shared Concepts**.” The SEVSELECT procedure supports only a subset of the *method* values and *method-options* that are listed in that section.

NOTE: In the context of the **SELECTION** statement, the term *model* refers to the set of regression effects in the scale regression model. The final severity model consists of the set of distribution parameters and the set of selected regression effects.

You can use the following option to specify the selection method:

METHOD=BACKWARD | FORWARD | FORWARDSWAP | STEPWISE

specifies the selection method to use. You can specify one of the following methods:

BACKWARD	specifies the backward elimination method.
FORWARD	specifies the forward selection method.
FORWARDSWAP	specifies forward-swap selection, which is an extension of the forward selection method.
STEPWISE	specifies the stepwise regression method.

By default, METHOD=STEPWISE. For more information about each of the selection methods, see the section “[Model Selection Methods](#)” on page 59

The SEVSELECT procedure supports a specific set of values for the following *method-options*:

CHOOSE=criterion

chooses from the list of models (at each step of the selection process) the model that yields the best value of the specified criterion. The SEVSELECT procedure supports only the following values for *criterion*:

AIC	specifies Akaike’s information criterion.
AICC	specifies the corrected Akaike’s information criterion.
SBC	specifies the Schwarz Bayesian information criterion.

By default, the CHOOSE= criterion is the same as the SELECT= criterion. For more information, see the detailed description of the [CHOOSE=](#) option.

SELECT=criterion

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. The SEVSELECT procedure supports the following values for *criterion*:

AIC	specifies Akaike’s information criterion.
AICC	specifies the corrected Akaike’s information criterion.
SBC	specifies the Schwarz Bayesian information criterion.

By default, SELECT=SBC. For more information, see the detailed description of the [SELECT=](#) option.

STOP=criterion

specifies a criterion that is used to stop the selection process. The SEVSELECT procedure supports only the following values for *criterion*:

AIC	specifies Akaike’s information criterion.
AICC	specifies the corrected Akaike’s information criterion.
SBC	specifies the Schwarz Bayesian information criterion.

By default, the STOP= criterion is the same as the SELECT= criterion. For more information, see the detailed description of the [STOP=](#) option.

The SEVSELECT procedure supports the following *method-options* as they are described in the section “[SELECTION Statement](#)” on page 36 in Chapter 3, “[Shared Concepts](#)”:

- COMPETITIVE
- MAXEFFECTS=
- MAXSTEPS=
- MINEFFECTS=

The SEVSELECT procedure supports the following statement *options* as they are described in the section “[SELECTION Statement](#)” on page 36 in Chapter 3, “[Shared Concepts](#)”:

- DETAILS=
- HIERARCHY=
- ORDERSELECT
- STOPHORIZON=

WEIGHT Statement

WEIGHT *variable-name* ;

The WEIGHT statement specifies the name of a variable whose values represent the weight of each observation. PROC SEVSELECT associates a weight of w to each observation, where w is the value of the WEIGHT variable for the observation. If the weight value is missing or less than or equal to 0, then the observation is ignored and a warning is written to the SAS log. When you do not specify the WEIGHT statement, each observation is assigned a weight of 1. If you specify more than one WEIGHT statement, then the last statement is used.

The weights are normalized so that they add up to the actual sample size. In particular, the weight of each observation is multiplied by $\frac{N}{\sum_{i=1}^N w_i}$, where N is the sample size. All computations, including the computations of the EDF-based statistics of fit, use normalized weights.

Programming Statements

You can use a series of programming statements that use variables in the input data table that you specify in the DATA= option in the PROC SEVSELECT statement to assign a value to an objective function symbol. You must specify the objective function symbol by using the **OBJECTIVE=** option in the PROC SEVSELECT statement. If you do not specify the **OBJECTIVE=** option in the PROC SEVSELECT statement, then the programming statements are ignored and models are estimated using the maximum likelihood method.

You can use most DATA step statements and functions in your program. Any additional functions, restrictions, and differences are listed in the section “[Custom Objective Functions](#)” on page 920.

Details: SEVSELECT Procedure

Predefined Distributions

For the response variable Y , PROC SEVSELECT assumes the model

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is a continuous probability distribution with parameters Θ . The model hypothesizes that the observed response is generated from a stochastic process that is governed by the distribution $\mathcal{F}$. This model is usually referred to as the error model. Given a representative input sample of response variable values, PROC SEVSELECT estimates the model parameters for any distribution $\mathcal{F}$ and computes the statistics of fit for each model. This enables you to find the distribution that is most likely to generate the observed sample.

A set of predefined distributions is provided with the SEVSELECT procedure. A summary of the distributions is provided in [Table 15.4](#). For each distribution, the table lists the name of the distribution that should be used in the DIST statement, the parameters of the distribution along with their bounds, and the mathematical expressions for the probability density function (PDF) and cumulative distribution function (CDF) of the distribution.

All the predefined distributions, except LOGN and TWEEDIE, are parameterized such that their first parameter is the scale parameter. For LOGN, the first parameter μ is a log-transformed scale parameter. TWEEDIE does not have a scale parameter. The presence of scale parameter or a log-transformed scale parameter enables you to use all of the predefined distributions, except TWEEDIE, as a candidate for estimating regression effects.

A distribution model is associated with each predefined distribution. You can also define your own distribution model, which is a set of functions and subroutines that you define by using the FCMP procedure. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895.

Table 15.4 Predefined PROC SEVSELECT Distributions

Name	Distribution	Parameters	PDF (f) and CDF (F)
BURR	Burr	$\theta > 0, \alpha > 0,$ $\gamma > 0$	$f(x) = \frac{\alpha \gamma z^\gamma}{x(1+z^\gamma)^{(\alpha+1)}}$ $F(x) = 1 - \left(\frac{1}{1+z^\gamma}\right)^\alpha$
EXP	Exponential	$\theta > 0$	$f(x) = \frac{1}{\theta} e^{-z}$ $F(x) = 1 - e^{-z}$
GAMMA	Gamma	$\theta > 0, \alpha > 0$	$f(x) = \frac{z^\alpha e^{-z}}{x \Gamma(\alpha)}$ $F(x) = \frac{\gamma(\alpha, z)}{\Gamma(\alpha)}$
GPD	Generalized Pareto	$\theta > 0, \xi > 0$	$f(x) = \frac{1}{\theta} (1 + \xi z)^{-1-1/\xi}$ $F(x) = 1 - (1 + \xi z)^{-1/\xi}$
IGAUSS	Inverse Gaussian (Wald)	$\theta > 0, \alpha > 0$	$f(x) = \frac{1}{\theta} \sqrt{\frac{\alpha}{2\pi z^3}} e^{\frac{-\alpha(z-1)^2}{2z}}$ $F(x) = \Phi\left((z-1)\sqrt{\frac{\alpha}{z}}\right) + \Phi\left(-(z+1)\sqrt{\frac{\alpha}{z}}\right) e^{2\alpha}$
LOGN	Lognormal	μ (no bounds), $\sigma > 0$	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$ $F(x) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$
PARETO	Pareto	$\theta > 0, \alpha > 0$	$f(x) = \frac{\alpha \theta^\alpha}{(x+\theta)^{\alpha+1}}$ $F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^\alpha$
TWEEDIE	Tweedie**	$p > 1, \mu > 0,$ $\phi > 0$	$f(x) = a(x, \phi) \exp\left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$ $F(x) = \int_0^x f(t) dt$
STWEEDIE	Scaled Tweedie**	$\theta > 0, \lambda > 0,$ $1 < p < 2$	$f(x) = a(x, \theta, \lambda, p) \exp\left(-\frac{x}{\theta} - \lambda\right)$ $F(x) = \int_0^x f(t) dt$
WEIBULL	Weibull	$\theta > 0, \tau > 0$	$f(x) = \frac{\tau}{\theta} z^{\tau-1} e^{-z^\tau}$ $F(x) = 1 - e^{-z^\tau}$

**For more information, see the section “[Tweedie Distributions](#)” on page 865.

Notes:

1. $z = x/\theta$, wherever z is used.
2. θ denotes the scale parameter for all the distributions. For LOGN, $\log(\theta) = \mu$.
3. Parameters are listed in the order in which they are defined in the distribution model.
4. $\gamma(a, b) = \int_0^b t^{a-1} e^{-t} dt$ is the lower incomplete gamma function.
5. $\Phi(y) = \frac{1}{2} \left(1 + \operatorname{erf}\left(\frac{y}{\sqrt{2}}\right)\right)$ is the standard normal CDF.

Tweedie Distributions

Tweedie distributions are a special case of the exponential dispersion family (Jørgensen 1987) with a property that the variance of the distribution is equal to $\phi\mu^p$, where μ is the mean of the distribution, ϕ is a dispersion parameter, and p is an index parameter as discovered by Tweedie (1984). The distribution is defined for all values of p except for values of p in the open interval $(0, 1)$. Many important known distributions are a special case of Tweedie distributions including normal ($p=0$), Poisson ($p=1$), gamma ($p=2$), and the inverse Gaussian ($p=3$). Apart from these special cases, the probability density function (PDF) of the Tweedie distribution does not have an analytic expression. For $p > 1$, it has the form (Dunn and Smyth 2005),

$$f(x; \mu, \phi, p) = a(x, \phi) \exp \left[\frac{1}{\phi} \left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p) \right) \right]$$

where $\kappa(\mu, p) = \mu^{2-p}/(2-p)$ for $p \neq 2$ and $\kappa(\mu, p) = \log(\mu)$ for $p = 2$. The function $a(x, \phi)$ does not have an analytical expression. It is typically evaluated using series expansion methods described in Dunn and Smyth (2005).

For $1 < p < 2$, the Tweedie distribution is a compound Poisson-gamma mixture distribution, which is the distribution of S defined as

$$S = \sum_{i=1}^N X_i$$

where $N \sim \text{Poisson}(\lambda)$ and $X_i \sim \text{gamma}(\alpha, \theta)$ are independent and identically distributed gamma random variables with shape parameter α and scale parameter θ . At $X = 0$, the density is a probability mass that is governed by the Poisson distribution, and for values of $X > 0$, it is a mixture of gamma variates with Poisson mixing probability. The parameters λ , α , and θ are related to the natural parameters μ , ϕ , and p of the Tweedie distribution as

$$\begin{aligned}\lambda &= \frac{\mu^{2-p}}{\phi(2-p)} \\ \alpha &= \frac{2-p}{p-1} \\ \theta &= \phi(p-1)\mu^{p-1}\end{aligned}$$

The mean of a Tweedie distribution is positive for $p > 1$.

Two predefined versions of the Tweedie distribution are provided with the SEVSELECT procedure. The first version, named TWEEDIE and defined for $p > 1$, has the natural parameterization with parameters μ , ϕ , and p . The second version, named STWEEDIE and defined for $1 < p < 2$, is the version with a scale parameter. It corresponds to the compound Poisson-gamma distribution with gamma scale parameter θ , Poisson mean parameter λ , and the index parameter p . The index parameter decides the shape parameter α of the gamma distribution as

$$\alpha = \frac{2-p}{p-1}$$

The parameters θ and λ of the STWEEDIE distribution are related to the parameters μ and ϕ of the TWEEDIE distribution as

$$\begin{aligned}\mu &= \lambda\theta\alpha \\ \phi &= \frac{(\lambda\theta\alpha)^{2-p}}{\lambda(2-p)} = \frac{\theta}{(p-1)(\lambda\theta\alpha)^{p-1}}\end{aligned}$$

You can fit either version when there are no regression variables. Each version has its own merits. If you fit the TWEEDIE version, you have the direct estimate of the overall mean of the distribution. If you are interested in the most practical range of the index parameter $1 < p < 2$, then you can fit the STWEEDIE version, which provides you direct estimates of the Poisson and gamma components that comprise the distribution (an estimate of the gamma shape parameter α is easily obtained from the estimate of p).

If you want to estimate the effect of exogenous (regression) variables on the distribution, then you must use the STWEEDIE version, because PROC SEVSELECT requires a distribution to have a scale parameter in order to estimate regression effects. For more information, see the section “[Estimating Regression Effects](#)” on page 878. The gamma scale parameter θ is the scale parameter of the STWEEDIE distribution. If you are interested in determining the effect of regression variables on the mean of the distribution, you can do so by first fitting the STWEEDIE distribution to determine the effect of the regression variables on the scale parameter θ . Then, you can easily estimate how the mean of the distribution μ is affected by the regression variables using the relationship $\mu = c\theta$, where $c = \lambda\alpha = \lambda(2 - p)/(p - 1)$. The estimates of the regression parameters remain the same, whereas the estimate of the intercept parameter is adjusted by the estimates of the λ and p parameters.

Parameter Initialization for Predefined Distributions

The parameters are initialized by using the method of moments for all the distributions, except for the gamma and the Weibull distributions. For the gamma distribution, approximate maximum likelihood estimates are used. For the Weibull distribution, the method of percentile matching is used.

Given n observations of the severity value y_i ($1 \leq i \leq n$), the estimate of k th raw moment is denoted by m_k and computed as

$$m_k = \frac{1}{n} \sum_{i=1}^n y_i^k$$

The 100 p th percentile is denoted by π_p ($0 \leq p \leq 1$). By definition, π_p satisfies

$$F(\pi_p-) \leq p \leq F(\pi_p)$$

where $F(\pi_p-) = \lim_{h \downarrow 0} F(\pi_p - h)$. PROC SEVSELECT uses the following practical method of computing π_p . Let $\hat{F}_n(y)$ denote the empirical distribution function (EDF) estimate at a severity value y . Let y_p^- and y_p^+ denote two consecutive values in the ascending sequence of y values such that $\hat{F}_n(y_p^-) < p$ and $\hat{F}_n(y_p^+) \geq p$. Then, the estimate $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = y_p^- + \frac{p - \hat{F}_n(y_p^-)}{\hat{F}_n(y_p^+) - \hat{F}_n(y_p^-)}(y_p^+ - y_p^-)$$

Let ϵ denote the smallest double-precision floating-point number such that $1 + \epsilon > 1$. This machine precision constant can be obtained by using the CONSTANT function in Base SAS software.

The details of how parameters are initialized for each predefined distribution are as follows:

BURR Burr proposed 12 types of families of continuous distributions (Burr 1942; Rodriguez and Balan 2006). The predefined BURR distribution in PROC SEVSELECT implements Burr's

type XII distribution. The parameters are initialized by using the method of moments. The k th raw moment of the Burr distribution of type XII is

$$E[X^k] = \frac{\theta^k \Gamma(1 + k/\gamma) \Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)}, \quad -\gamma < k < \alpha\gamma$$

Three moment equations $E[X^k] = m_k$ ($k = 1, 2, 3$) need to be solved for initializing the three parameters of the distribution. In order to get an approximate closed form solution, the second shape parameter $\hat{\gamma}$ is initialized to a value of 2. If $2m_3 - 3m_1m_2 > 0$, then simplifying and solving the moment equations yields the following feasible set of initial values:

$$\hat{\theta} = \sqrt{\frac{m_2m_3}{2m_3 - 3m_1m_2}}, \quad \hat{\alpha} = 1 + \frac{m_3}{2m_3 - 3m_1m_2}, \quad \hat{\gamma} = 2$$

If $2m_3 - 3m_1m_2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \sqrt{m_2}, \quad \hat{\alpha} = 2, \quad \hat{\gamma} = 2$$

EXP

The parameters are initialized by using the method of moments. The k th raw moment of the exponential distribution is

$$E[X^k] = \theta^k \Gamma(k + 1), \quad k > -1$$

Solving $E[X] = m_1$ yields the initial value of $\hat{\theta} = m_1$.

GAMMA

The parameter α is initialized by using its *approximate* maximum likelihood (ML) estimate. For a set of n independent and identically distributed observations y_i ($1 \leq i \leq n$) drawn from a gamma distribution, the log likelihood l is defined as follows:

$$\begin{aligned} l &= \sum_{i=1}^n \log \left(y_i^{\alpha-1} \frac{e^{-y_i/\theta}}{\theta^\alpha \Gamma(\alpha)} \right) \\ &= (\alpha - 1) \sum_{i=1}^n \log(y_i) - \frac{1}{\theta} \sum_{i=1}^n y_i - n\alpha \log(\theta) - n \log(\Gamma(\alpha)) \end{aligned}$$

Using a shorter notation of $\sum$ to denote $\sum_{i=1}^n$ and solving the equation $\partial l / \partial \theta = 0$ yields the following ML estimate of θ :

$$\hat{\theta} = \frac{\sum y_i}{n\alpha} = \frac{m_1}{\alpha}$$

Substituting this estimate in the expression of l and simplifying gives

$$l = (\alpha - 1) \sum \log(y_i) - n\alpha - n\alpha \log(m_1) + n\alpha \log(\alpha) - n \log(\Gamma(\alpha))$$

Let d be defined as follows:

$$d = \log(m_1) - \frac{1}{n} \sum \log(y_i)$$

Solving the equation $\partial l / \partial \alpha = 0$ yields the following expression in terms of the digamma function, $\psi(\alpha)$:

$$\log(\alpha) - \psi(\alpha) = d$$

The digamma function can be approximated as follows:

$$\hat{\psi}(\alpha) \approx \log(\alpha) - \frac{1}{\alpha} \left(0.5 + \frac{1}{12\alpha + 2} \right)$$

This approximation is within 1.4% of the true value for all the values of $\alpha > 0$ except when α is arbitrarily close to the positive root of the digamma function (which is approximately 1.461632). Even for the values of α that are close to the positive root, the absolute error between true and approximate values is still acceptable ($|\hat{\psi}(\alpha) - \psi(\alpha)| < 0.005$ for $\alpha > 1.07$). Solving the equation that arises from this approximation yields the following estimate of α :

$$\hat{\alpha} = \frac{3 - d + \sqrt{(d - 3)^2 + 24d}}{12d}$$

If this approximate ML estimate is infeasible, then the method of moments is used. The k th raw moment of the gamma distribution is

$$E[X^k] = \theta^k \frac{\Gamma(\alpha + k)}{\Gamma(\alpha)}, \quad k > -\alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial value for α :

$$\hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then α is initialized as follows:

$$\hat{\alpha} = 1$$

After computing the estimate of α , the estimate of θ is computed as follows:

$$\hat{\theta} = \frac{m_1}{\hat{\alpha}}$$

Both the maximum likelihood method and the method of moments arrive at the same relationship between $\hat{\alpha}$ and $\hat{\theta}$.

GPD

The parameters are initialized by using the method of moments. Notice that for $\xi > 0$, the CDF of the generalized Pareto distribution (GPD) is:

$$\begin{aligned} F(x) &= 1 - \left(1 + \frac{\xi x}{\theta} \right)^{-1/\xi} \\ &= 1 - \left(\frac{\theta/\xi}{x + \theta/\xi} \right)^{1/\xi} \end{aligned}$$

This is equivalent to a Pareto distribution with scale parameter $\theta_1 = \theta/\xi$ and shape parameter $\alpha = 1/\xi$. Using this relationship, the parameter initialization method used for the PARETO distribution is used to get the following initial values for the parameters of the GPD distribution:

$$\hat{\theta} = \frac{m_1 m_2}{2(m_2 - m_1^2)}, \quad \hat{\xi} = \frac{m_2 - 2m_1^2}{2(m_2 - m_1^2)}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \frac{m_1}{2}, \quad \hat{\xi} = \frac{1}{2}$$

IGAUSS

The parameters are initialized by using the method of moments. The standard parameterization of the inverse Gaussian distribution (also known as the Wald distribution), in terms of the location parameter μ and shape parameter λ , is as follows (Klugman, Panjer, and Willmot 1998, p. 583):

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x - \mu)^2}{2\mu^2 x}\right)$$

$$F(x) = \Phi\left(\left(\frac{x}{\mu} - 1\right) \sqrt{\frac{\lambda}{x}}\right) + \Phi\left(-\left(\frac{x}{\mu} + 1\right) \sqrt{\frac{\lambda}{x}}\right) \exp\left(\frac{2\lambda}{\mu}\right)$$

For this parameterization, it is known that the mean is $E[X] = \mu$ and the variance is $\text{Var}[X] = \mu^3/\lambda$, which yields the second raw moment as $E[X^2] = \mu^2(1 + \mu/\lambda)$ (computed by using $E[X^2] = \text{Var}[X] + (E[X])^2$).

The predefined IGAUSS distribution in PROC SEVSELECT uses the following alternate parameterization to allow the distribution to have a scale parameter, θ :

$$f(x) = \sqrt{\frac{\alpha\theta}{2\pi x^3}} \exp\left(\frac{-\alpha(x - \theta)^2}{2x\theta}\right)$$

$$F(x) = \Phi\left(\left(\frac{x}{\theta} - 1\right) \sqrt{\frac{\alpha\theta}{x}}\right) + \Phi\left(-\left(\frac{x}{\theta} + 1\right) \sqrt{\frac{\alpha\theta}{x}}\right) \exp(2\alpha)$$

The parameters θ (scale) and α (shape) of this alternate form are related to the parameters μ and λ of the preceding form such that $\theta = \mu$ and $\alpha = \lambda/\mu$. Using this relationship, the first and second raw moments of the IGAUSS distribution are

$$E[X] = \theta$$

$$E[X^2] = \theta^2 \left(1 + \frac{1}{\alpha}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 1$$

LOGN

The parameters are initialized by using the method of moments. The k th raw moment of the lognormal distribution is

$$E[X^k] = \exp\left(k\mu + \frac{k^2\sigma^2}{2}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\mu} = 2 \log(m_1) - \frac{\log(m_2)}{2}, \quad \hat{\sigma} = \sqrt{\log(m_2) - 2 \log(m_1)}$$

PARETO The parameters are initialized by using the method of moments. The k th raw moment of the Pareto distribution is

$$E[X^k] = \frac{\theta^k \Gamma(k+1) \Gamma(\alpha-k)}{\Gamma(\alpha)}, -1 < k < \alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = \frac{m_1 m_2}{m_2 - 2m_1^2}, \quad \hat{\alpha} = \frac{2(m_2 - m_1^2)}{m_2 - 2m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 2$$

TWEEDIE The parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. The parameter μ is the mean of the distribution. Hence, it is initialized to the sample mean as

$$\hat{\mu} = m_1$$

Variance of a Tweedie distribution is equal to $\phi \mu^p$. Thus, the sample variance is used to initialize the value of ϕ as

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

STWEEDIE STWEEDIE is a compound Poisson-gamma mixture distribution with mean $\mu = \lambda \theta \alpha$, where α is the shape parameter of the gamma random variables in the mixture and the parameter p is determined solely by α . First, the parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. As done for initializing the parameters of the TWEEDIE distribution, the sample mean and variance are used to compute the values $\hat{\mu}$ and $\hat{\phi}$ as

$$\hat{\mu} = m_1$$

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

Based on the relationship between the parameters of TWEEDIE and STWEEDIE distributions described in the section “[Tweedie Distributions](#)” on page 865, values of θ and λ are initialized as

$$\hat{\theta} = \hat{\phi}(\hat{p} - 1)\hat{\mu}^{p-1}$$

$$\hat{\lambda} = \frac{\hat{\mu}}{\hat{\theta}\hat{\alpha}}$$

WEIBULL The parameters are initialized by using the percentile matching method. Let $q1$ and $q3$ denote the estimates of the 25th and 75th percentiles, respectively. Using the formula for the CDF of Weibull distribution, they can be written as

$$1 - \exp(-(q1/\theta)^\tau) = 0.25$$

$$1 - \exp(-(q3/\theta)^\tau) = 0.75$$

Simplifying and solving these two equations yields the following initial values,

$$\hat{\theta} = \exp\left(\frac{r \log(q1) - \log(q3)}{r - 1}\right), \quad \hat{\tau} = \frac{\log(\log(4))}{\log(q3) - \log(\hat{\theta})}$$

where $r = \log(\log(4)) / \log(\log(4/3))$. These initial values agree with those suggested in Klugman, Panjer, and Willmot (1998).

A summary of the initial values of all the parameters for all the predefined distributions is given in [Table 15.5](#). The table also provides the names of the parameters to use in the **INIT=** option in the DIST statement if you want to provide a different initial value.

Table 15.5 Parameter Initialization for Predefined Distributions

Distribution	Parameter	Name for INIT Option	Default Initial Value
BURR	θ	theta	$\sqrt{\frac{m_2 m_3}{2m_3 - 3m_1 m_2}}$
	α	alpha	$1 + \frac{m_3}{2m_3 - 3m_1 m_2}$
	γ	gamma	2
EXP	θ	theta	m_1
GAMMA	θ	theta	m_1/α
	α	alpha	$\frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
GPD	θ	theta	$m_1 m_2 / (2(m_2 - m_1^2))$
	ξ	xi	$(m_2 - 2m_1^2) / (2(m_2 - m_1^2))$
IGAUSS	θ	theta	m_1
	α	alpha	$m_1^2 / (m_2 - m_1^2)$
LOGN	μ	mu	$2 \log(m_1) - \log(m_2)/2$
	σ	sigma	$\sqrt{\log(m_2) - 2 \log(m_1)}$
PARETO	θ	theta	$m_1 m_2 / (m_2 - 2m_1^2)$
	α	alpha	$2(m_2 - m_1^2) / (m_2 - 2m_1^2)$
TWEEDIE	μ	mu	m_1
	ϕ	phi	$(m_2 - m_1^2) / m_1^p$
	p	p	$(\alpha + 2) / (\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
STWEEDIE	θ	theta	$(m_2 - m_1^2)(p - 1) / m_1$
	λ	lambda	$m_1^2 / (\alpha(m_2 - m_1^2)(p - 1))$
	p	p	$(\alpha + 2) / (\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
WEIBULL	θ	theta	$\exp\left(\frac{r \log(q1) - \log(q3)}{r-1}\right)$
	τ	tau	$\log(\log(4)) / (\log(q3) - \log(\hat{\theta}))$

Notes:

1. m_k denotes the k th raw moment.
2. $d = \log(m_1) - (\sum \log(y_i))/n$
3. $q1$ and $q3$ denote the 25th and 75th percentiles, respectively.
4. $r = \log(\log(4)) / \log(\log(4/3))$

Censoring and Truncation

One of the key features of PROC SEVSELECT is that it enables you to specify whether the severity event's magnitude is observable and if it is observable, then whether the exact value of the magnitude is known. If an event is unobservable when the magnitude is in certain intervals, then it is referred to as a truncation effect. If the exact magnitude of the event is not known, but it is known to have a value in a certain interval, then it is referred to as a censoring effect.

PROC SEVSELECT allows a severity event to be subject to any combination of the following four censoring and truncation effects:

- **Left-truncation:** An event is said to be left-truncated if it is observed only when $Y > T^l$, where Y denotes the random variable for the magnitude and T^l denotes a random variable for the truncation threshold. You can specify left-truncation using the `LEFTTRUNCATED=` option in the LOSS statement.
- **Right-truncation:** An event is said to be right-truncated if it is observed only when $Y \leq T^r$, where Y denotes the random variable for the magnitude and T^r denotes a random variable for the truncation threshold. You can specify right-truncation using the `RIGHTTRUNCATED=` option in the LOSS statement.
- **Left-censoring:** An event is said to be left-censored if it is known that the magnitude is $Y \leq C^l$, but the exact value of Y is not known. C^l is a random variable for the censoring limit. You can specify left-censoring using the `LEFTCENSORED=` option in the LOSS statement.
- **Right-censoring:** An event is said to be right-censored if it is known that the magnitude is $Y > C^r$, but the exact value of Y is not known. C^r is a random variable for the censoring limit. You can specify right-censoring using the `RIGHTCENSORED=` option in the LOSS statement.

For each effect, you can specify a different threshold or limit for each observation or specify a single threshold or limit that applies to all the observations.

If all four types of effects are present on an event, then the following relationship holds: $T^l < C^r \leq C^l \leq T^r$. PROC SEVSELECT checks these relationships and writes a warning to the SAS log if any relationship is violated.

If you specify the response variable in the LOSS statement, then PROC SEVSELECT also checks whether each observation satisfies the definitions of the specified censoring and truncation effects. If you specify left-truncation, then PROC SEVSELECT ignores observations where $Y \leq T^l$, because such observations are not observable by definition. Similarly, if you specify right-truncation, then PROC SEVSELECT ignores observations where $Y > T^r$. If you specify left-censoring, then PROC SEVSELECT treats an observation with $Y > C^l$ as uncensored and ignores the value of C^l . The observations with $Y \leq C^l$ are considered as left-censored, and the value of Y is ignored. If you specify right-censoring, then PROC SEVSELECT treats an observation with $Y \leq C^r$ as uncensored and ignores the value of C^r . The observations with $Y > C^r$ are considered as right-censored, and the value of Y is ignored. If you specify both left-censoring and right-censoring, it is referred to as interval-censoring. If $C^r < C^l$ is satisfied for an observation, then it is considered as interval-censored and the value of the response variable is ignored. If $C^r = C^l$ for an observation, then PROC SEVSELECT assumes that observation to be uncensored. If all the observations in a data table are censored in some form, then the specification of the response variable in the LOSS statement is

optional, because the actual value of the response variable is not required for the purposes of estimating a model.

Specification of censoring and truncation affects the likelihood of the data (see the section “[Likelihood Function](#)” on page 875) and how the empirical distribution function (EDF) is estimated (see the section “[Empirical Distribution Function Estimation Methods](#)” on page 883).

Probability of Observability

For left-truncated data, PROC SEVSELECT also enables you to provide additional information in the form of *probability of observability* by using the `PROBOBSERVED=` option. It is defined as the probability that the underlying severity event gets observed (and recorded) for the specified left-truncation threshold value. For example, if you specify a value of 0.75, then for every 75 observations recorded above a specified threshold, 25 more events have happened with a severity value less than or equal to the specified threshold. Although the exact severity value of those 25 events is not known, PROC SEVSELECT can use the information about the number of those events.

In particular, for each left-truncated observation, PROC SEVSELECT assumes a presence of $(1 - p)/p$ additional observations with $y_i = t_i$. These additional observations are then used for computing the likelihood (see the section “[Probability of Observability and Likelihood](#)” on page 876) and an unconditional estimate of the empirical distribution function (see the section “[EDF Estimates and Truncation](#)” on page 888).

Truncation and Conditional CDF Estimates

If you specify left-truncation without the probability of observability or if you specify right-truncation, then the EDF estimates that are computed by all methods except the STANDARD method are conditional on the truncation information. For more information, see the section “[EDF Estimates and Truncation](#)” on page 888. In such cases, PROC SEVSELECT uses conditional estimates of the CDF for computational or visual comparison to the EDF estimates.

Let $t_{\min}^l = \min_i \{t_i^l\}$ be the smallest value of the left-truncation threshold (t_i^l is the left-truncation threshold for observation i) and $t_{\max}^r = \max_i \{t_i^r\}$ be the largest value of the right-truncation threshold (t_i^r is the right-truncation threshold for observation i). If $\hat{F}(y)$ denotes the unconditional estimate of the CDF at y , then the conditional estimate $\hat{F}^c(y)$ is computed as follows:

- If you do not specify the probability of observability, then the EDF estimates are conditional on the left-truncation information. If an observation is both left-truncated and right-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\min}^l)}{\hat{F}(t_{\max}^r) - \hat{F}(t_{\min}^l)}$$

If an observation is left-truncated but not right-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\min}^l)}{1 - \hat{F}(t_{\min}^l)}$$

If an observation is right-truncated but not left-truncated, then

$$\hat{F}^c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\max}^r)}$$

- If you specify the probability of observability, then EDF estimates are not conditional on the left-truncation information. If an observation is not right-truncated, then the conditional estimate is the same as the unconditional estimate. If an observation is right-truncated, then the conditional estimate is computed as

$$\hat{F}^c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\max}^r)}$$

If you specify regression effects, then $\hat{F}(y)$, $\hat{F}(t_{\min}^l)$, and $\hat{F}(t_{\max}^r)$ are all computed from a mixture distribution, as described in the section “CDF and PDF Estimates with Regression Effects” on page 881.

Parameter Estimation Method

PROC SEVSELECT uses the maximum likelihood (ML) method to estimate the parameters of each model. A nonlinear optimization process is used to maximize the log of the likelihood function.

Likelihood Function

Let $f_{\Theta}(x)$ and $F_{\Theta}(x)$ denote the PDF and CDF, respectively, evaluated at x for a set of parameter values Θ . Let Y denote the random response variable, and let y denote its value recorded in an observation in the input data table. Let T^l and T^r denote the random variables for the left-truncation and right-truncation threshold, respectively, and let t^l and t^r denote their values for an observation, respectively. If there is no left-truncation, then $t^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t^l) = 0$. If there is no right-truncation, then $t^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t^r) = 1$. Let C^l and C^r denote the random variables for the left-censoring and right-censoring limit, respectively, and let c^l and c^r denote their values for an observation, respectively. If there is no left-censoring, then $c^l = \tau_h$; so $F(c^l) = 1$. If there is no right-censoring, then $c^r = \tau^l$; so $F(c^r) = 0$.

The set of input observations can be categorized into the following four subsets within each BY group:

- E is the set of uncensored and untruncated observations. The likelihood of an observation in E is

$$l_E = \Pr(Y = y) = f_{\Theta}(y)$$

- E_t is the set of uncensored observations that are truncated. The likelihood of an observation in E_t is

$$l_{E_t} = \Pr(Y = y | t^l < Y \leq t^r) = \frac{f_{\Theta}(y)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

- C is the set of censored observations that are not truncated. The likelihood of an observation C is

$$l_C = \Pr(c^r < Y \leq c^l) = F_{\Theta}(c^l) - F_{\Theta}(c^r)$$

- C_t is the set of censored observations that are truncated. The likelihood of an observation C_t is

$$l_{C_t} = \Pr(c^r < Y \leq c^l | t^l < Y \leq t^r) = \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

Note that $(E \cup E_t) \cap (C \cup C_t) = \emptyset$. Also, the sets E_t and C_t are empty when you do not specify truncation, and the sets C and C_t are empty when you do not specify censoring.

Given this, the likelihood of the data L is as follows:

$$L = \prod_E f_{\Theta}(y) \prod_{E_t} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)} \prod_C F_{\Theta}(c^l) - F_{\Theta}(c^r) \prod_{C_t} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r) - F_{\Theta}(t^l)}$$

The maximum likelihood procedure used by PROC SEVSELECT finds an optimal set of parameter values $\hat{\Theta}$ that maximizes $\log(L)$ subject to the boundary constraints on parameter values. For a distribution *dist*, you can specify such boundary constraints by using the *dist_LOWERBOUNDS* and *dist_UPPERBOUNDS* subroutines. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895. Some aspects of the optimization process can be controlled by using the *NLOPTIONS* statement.

Probability of Observability and Likelihood

If you specify the probability of observability for the left-truncation, then PROC SEVSELECT uses a modified likelihood function for each truncated observation. If the probability of observability is $p \in (0.0, 1.0]$, then for each left-truncated observation with truncation threshold t^l , there exist $(1-p)/p$ observations with a response variable value less than or equal to t^l . Each such observation has a probability of $\Pr(Y \leq t^l) = F_{\Theta}(t^l)$. The right-truncation and censoring information does not apply to these added observations. Thus, following the notation of the section “[Likelihood Function](#)” on page 875, the likelihood of the data is as follows:

$$L = \prod_E f_{\Theta}(y) \prod_{E_t, t^l = \tau^l} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r)} \prod_{E_t, t^l > \tau^l} \frac{f_{\Theta}(y)}{F_{\Theta}(t^r)} F_{\Theta}(t^l)^{\frac{1-p}{p}} \\ \prod_C F_{\Theta}(c^l) - F_{\Theta}(c^r) \prod_{C_t, t^l = \tau^l} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r)} \prod_{C_t, t^l > \tau^l} \frac{F_{\Theta}(c^l) - F_{\Theta}(c^r)}{F_{\Theta}(t^r)} F_{\Theta}(t^l)^{\frac{1-p}{p}}$$

Note that the likelihood of the observations that are not left-truncated (observations in sets E and C , and observations in sets E_t and C_t for which $t^l = \tau^l$) is not affected.

If you specify a custom objective function, then PROC SEVSELECT accounts for the probability of observability only while computing the empirical distribution function estimate. The parameter estimates are affected only by your custom objective function.

Estimating Covariance and Standard Errors

PROC SEVSELECT computes an estimate of the covariance matrix of the parameters by using the asymptotic theory of the maximum likelihood estimators (MLE). If N denotes the number of observations used for estimating a parameter vector θ , then the theory states that as $N \rightarrow \infty$, the distribution of $\hat{\theta}$, the estimate of θ , converges to a normal distribution with mean θ and covariance $\hat{C}$ such that $\mathbf{I}(\theta) \cdot \hat{C} \rightarrow 1$, where $\mathbf{I}(\theta) = -E[\nabla^2 \log(L(\theta))]$ is the information matrix for the likelihood of the data, $L(\theta)$. The covariance estimate is obtained by using the inverse of the information matrix.

In particular, if $\mathbf{G} = \nabla^2(-\log(L(\boldsymbol{\theta})))$ denotes the Hessian matrix of the negative of log likelihood, then the covariance estimate is computed as

$$\hat{\mathbf{C}} = \frac{N}{d} \mathbf{G}^{-1}$$

where d is a denominator that is determined by the **VARDEF=** option. If **VARDEF=N**, then $d = N$, which yields the asymptotic covariance estimate. If **VARDEF=DF**, then $d = N - k$, where k is number of parameters (the model's degrees of freedom). The **VARDEF=DF** option is the default, because it attempts to correct the potential bias introduced by the finite sample.

The standard error s_i of the parameter θ_i is computed as the square root of the i th diagonal element of the estimated covariance matrix; that is, $s_i = \sqrt{\hat{C}_{ii}}$.

If you specify a custom objective function, then the covariance matrix of the parameters is still computed by inverting the information matrix, except that the Hessian matrix $\mathbf{G}$ is computed as $\mathbf{G} = \nabla^2 \log(U(\boldsymbol{\theta}))$, where U denotes your custom objective function that is minimized by the optimizer.

Covariance and standard error estimates might not be available if the Hessian matrix is found to be singular at the end of the optimization process. This can especially happen if the optimization process stops without converging.

Parameter Initialization

PROC SEVSELECT enables you to initialize parameters of a model in different ways. A model can have two kinds of parameters: distribution parameters and regression parameters.

The distribution parameters can be initialized by using one of the following three methods:

INIT= option	You can use the INIT= option in the DIST statement.
INEST= option	You can use the INEST= data table in the PROC SEVSELECT statement.
PARMINIT subroutine	You can define a <i>dist_PARMINIT</i> subroutine in the distribution model. For more information, see the section “ Defining a Severity Distribution Model with the FCMP Procedure ” on page 895.

Note that only one of the initialization methods is used. You cannot combine them. They are used in the following order:

- The method that uses the **INIT=** option takes the highest precedence. If you use the **INIT=** option to provide an initial value for at least one parameter, then other initialization methods (**INEST=** or **PARMINIT**) are not used. If you specify initial values for some but not all the parameters by using the **INIT=** option, then the uninitialized parameters are initialized to the default value of 0.001.
If you use this option and if you specify the regression effects, then the value of the first distribution parameter must be related to the initial value for the *base* value of the scale or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 878.
- The method that uses the **INEST=** data table takes second precedence. If the **INEST=** data table contains a nonmissing value for even one distribution parameter, then the **PARMINIT** method is not used and any uninitialized parameters are initialized to the default value of 0.001.

- If none of the distribution parameters are initialized by using the INIT= option or the INEST= data table, but the distribution model defines a PARMINIT subroutine, then PROC SEVSELECT invokes that subroutine with appropriate inputs to initialize the parameters. If the PARMINIT subroutine returns missing values for some parameters, then those parameters are initialized to the default value of 0.001.
- If none of the initialization methods are used, each distribution parameter is initialized to the default value of 0.001.

For more information about regression models and initialization of regression parameters, see the section “[Estimating Regression Effects](#)” on page 878.

PARMINIT-Based Parameter Initialization Method and Distributed Data

If the input data are distributed across the computational nodes and if you use the PARMINIT subroutine to initialize the distribution parameters, then PROC SEVSELECT invokes that subroutine on each computational node with the data that are local to that node. The EDF estimates that are supplied to the PARMINIT subroutine are also computed using the local data. The initial values of the parameters that are supplied to the optimizer are the average of the local estimates that are computed on each node. This approach works well if the data are distributed randomly across nodes.

Estimating Regression Effects

The SEVSELECT procedure enables you to estimate the influence of regression (exogenous) effects while fitting a distribution if the distribution has a scale parameter or a log-transformed scale parameter.

Let x_j , $j = 1, \dots, k$, denote the k regression effects. Let β_j denote the regression parameter that corresponds to the effect x_j . If you do not specify regression effects, then the model for the response variable Y is of the form

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is the distribution of Y with parameters Θ . This model is usually referred to as the error model. The regression effects are modeled by extending the error model to the following form:

$$Y \sim \exp\left(\sum_{j=1}^k \beta_j x_j\right) \cdot \mathcal{F}(\Theta)$$

Under this model, the distribution of Y is valid and belongs to the same parametric family as $\mathcal{F}$ if and only if $\mathcal{F}$ has a scale parameter. Let θ denote the scale parameter and Ω denote the set of nonscale distribution parameters of $\mathcal{F}$. Then the model can be rewritten as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

such that θ is modeled by the regression effects as

$$\theta = \theta_0 \cdot \exp\left(\sum_{j=1}^k \beta_j x_j\right)$$

where θ_0 is the *base* value of the scale parameter. Thus, the scale regression model consists of the following parameters: θ_0 , Ω , and β_j ($j = 1, \dots, k$).

Given this form of the model, distributions without a scale parameter cannot be considered when regression effects are to be modeled. If a distribution does not have a direct scale parameter, then PROC SEVSELECT accepts it only if it has a log-transformed scale parameter—that is, if it has a parameter $p = \log(\theta)$.

Offset Variable

You can specify that an offset variable be included in the scale regression model by specifying it in the **OFFSET=** option of the SCALEMODEL statement. The offset variable is a regressor whose regression coefficient is known to be 1. If x_o denotes the offset variable, then the scale regression model becomes

$$\theta = \theta_0 \cdot \exp(x_o + \sum_{j=1}^k \beta_j x_j)$$

The regression coefficient of the offset variable is fixed at 1 and not estimated, so it is not reported in the ParameterEstimates ODS table. However, if you specify the OUTEST= data table, then the regression coefficient is added as a variable to that data table. The value of the offset variable in OUTEST= data table is equal to 1 for the estimates row (_TYPE_='EST') and is equal to the special missing value .F for the standard error (_TYPE_='STDERR') and covariance (_TYPE_='COV') rows.

An offset variable is useful to model the scale parameter per unit of some measure of exposure. For example, in the automobile insurance context, measure of exposure can be the number of car-years insured or the total number of miles driven by a fleet of cars at a rental car company. For worker's compensation insurance, if you want to model the expected loss per enterprise, then you can use the number of employees or total employee salary as the measure of exposure. For epidemiological data, measure of exposure can be the number of people who are exposed to a certain pathogen when you are modeling the loss associated with an epidemic. In general, if e denotes the value of the exposure measure and if you specify $x_o = \log(e)$ as the offset variable, then you are modeling the influence of other regression effects (x_j) on the size of the scale of the distribution *per unit of exposure*.

Another use for an offset variable is when you have a priori knowledge of the influence of some exogenous variables that cannot be included in the SCALEMODEL statement. You can model the combined influence of such variables as an offset variable in order to correct for the omitted variable bias.

Parameter Initialization for Regression Models

The regression parameters are initialized either by using the values that you specify or by the default method.

- If you provide initial values for the regression parameters, then you must provide valid, nonmissing initial values for θ_0 and β_j parameters for all j .

You can specify the initial value for θ_0 by using either the INEST= data table or the INIT= option in the DIST statement. If the distribution has a direct scale parameter (no transformation), then the initial value for the first parameter of the distribution is used as an initial value for θ_0 . If the distribution has a log-transformed scale parameter, then the initial value for the first parameter of the distribution is used as an initial value for $\log(\theta_0)$.

If you use the INEST= data table to specify the initial values for β_j , then the INEST= data table must be an unmodified version of a OUTEST= data table that you have created in a previous PROC SEVSELECT step with the same set of regression effects.

- If you do not specify valid initial values for θ_0 or β_j parameters for all j , then PROC SEVSELECT initializes those parameters by using the following method:

Let a random variable Y be distributed as $\mathcal{F}(\theta, \Omega)$, where θ is the scale parameter. By the definition of the scale parameter, a random variable $W = Y/\theta$ is distributed as $\mathcal{G}(\Omega)$ such that $\mathcal{G}(\Omega) = \mathcal{F}(1, \Omega)$. Given a random error term e that is generated from a distribution $\mathcal{G}(\Omega)$, a value y from the distribution of Y can be generated as

$$y = \theta \cdot e$$

Taking the logarithm of both sides and using the relationship of θ with the regression effects yields

$$\log(y) = \log(\theta_0) + \sum_{j=1}^k \beta_j x_j + \log(e)$$

PROC SEVSELECT makes use of the preceding relationship to initialize parameters of a regression model with distribution *dist* as follows:

1. The following linear regression problem is solved to obtain initial estimates of β_0 and β_j :

$$\log(y) = \beta_0 + \sum_{j=1}^k \beta_j x_j$$

The estimates of $\beta_j (j = 1, \dots, k)$ in the solution of this regression problem are used to initialize the respective regression parameters of the model. The estimate of β_0 is later used to initialize the value of θ_0 .

The results of this regression are also used to detect whether any regression parameters are linearly dependent on the other regression parameters. If any such parameters are found, then a warning is written to the SAS log and the corresponding parameter is eliminated from further analysis. The estimates for linearly dependent regression parameters are denoted by the special missing value .R in the OUTEST= data table and in any displayed output.

2. Let s_0 denote the initial value of the scale parameter.

If the distribution model of *dist* does not contain the *dist_PARMINIT* subroutine, then s_0 and all the nonscale distribution parameters are initialized to the default value of 0.001.

However, it is strongly recommended that each distribution's model contain the *dist_PARMINIT* subroutine. For more information, see the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895. If that subroutine is defined, then s_0 is initialized as follows: Each input value y_i of the response variable is transformed to its scale-normalized version w_i as

$$w_i = \frac{y_i}{\exp(\beta_0 + \sum_{j=1}^k \beta_j x_{ij})}$$

where x_{ij} denotes the value of j th regression effect in the i th input observation. These w_i values are used to compute the input arguments for the *dist_PARMINIT* subroutine. The values that are computed by the subroutine for nonscale parameters are used as their respective initial values. If the distribution has an untransformed scale parameter, then s_0 is set to the value of the scale parameter that is computed by the subroutine. If the distribution has a log-transformed scale parameter P , then s_0 is computed as $s_0 = \exp(l_0)$, where l_0 is the value of P computed by the subroutine.

3. The value of θ_0 is initialized as

$$\theta_0 = s_0 \cdot \exp(\beta_0)$$

Reporting Estimates of Regression Parameters

When you request estimates to be written to the output (either ODS displayed output or in the OUTEST= data table), the estimate of the base value of the first distribution parameter is reported. If the first parameter is the log-transformed scale parameter, then the estimate of $\log(\theta_0)$ is reported; otherwise, the estimate of θ_0 is reported. The transform of the first parameter of a distribution *dist* is controlled by the *dist_SCALETRANSFORM* function that is defined for it.

CDF and PDF Estimates with Regression Effects

When regression effects are estimated, the estimate of the scale parameter depends on the values of the regressors and the estimates of the regression parameters. This dependency results in a potentially different distribution for each observation. To make estimates of the cumulative distribution function (CDF) and probability density function (PDF) comparable across distributions and comparable to the empirical distribution function (EDF), PROC SEVSELECT computes and reports the CDF and PDF estimates from a representative distribution. The *representative distribution* is a mixture of a certain number of distributions, where each distribution differs only in the value of the scale parameter. You can specify the number of distributions in the mixture and how their scale values are chosen by using the *DFMIXTURE*= option in the SCALEMODEL statement.

Let N denote the number of observations that are used for estimation, K denote the number of components in the mixture distribution, s_k denote the scale parameter of the k th mixture component, and d_k denote the weight associated with k th mixture component.

Let $f(y; s_k, \hat{\Omega})$ and $F(y; s_k, \hat{\Omega})$ denote the PDF and CDF, respectively, of the k th component distribution, where $\hat{\Omega}$ denotes the set of estimates of all parameters of the distribution other than the scale parameter. Then, the PDF and CDF estimates, $f^*(y)$ and $F^*(y)$, respectively, of the mixture distribution at y are computed as

$$f^*(y) = \frac{1}{D} \sum_{k=1}^K d_k f(y; s_k, \hat{\Omega})$$

$$F^*(y) = \frac{1}{D} \sum_{k=1}^K d_k F(y; s_k, \hat{\Omega})$$

where D is the normalization factor ($D = \sum_{k=1}^K d_k$).

PROC SEVSELECT uses the $F^*(y)$ values to compute the EDF-based statistics of fit.

The scale values s_k for the K mixture components are derived from the set $\{\hat{\lambda}_i\}$ ($i = 1, \dots, N$) of N linear predictor values, where $\hat{\lambda}_i$ denotes the estimate of the linear predictor due to observation i . It is computed as

$$\hat{\lambda}_i = \log(\hat{\theta}_0) + \sum_{j=1}^k \hat{\beta}_j x_{ij}$$

where $\hat{\theta}_0$ is an estimate of the base value of the scale parameter, $\hat{\beta}_j$ are the estimates of regression coefficients, and x_{ij} is the value of j th regression effect in observation i .

Let w_i denote the weight of observation i . If you specify the WEIGHT statement, then the weight is equal to the value of the specified weight variable for the corresponding observation in the DATA= data table; otherwise, the weight is set to 1.

You can specify one of the following *method-names* in the DFMIXTURE= option in the SCALEMODEL statement to specify the method of choosing K and the corresponding s_k and d_k values:

FULL In this method, there are as many mixture components as the number of observations that are used for estimation. In other words, $K = N$, $s_k = \hat{\theta}_k$, and $d_k = w_k$ ($k = 1, \dots, N$). This is the slowest method, because it requires $O(N)$ computations to compute the mixture CDF $F^*(y_i)$ or the mixture PDF $f^*(y_i)$ of one observation. For N observations, the computational complexity in terms of number of CDF or PDF evaluations is $O(N^2)$. Even for moderately large values of N , the time that is taken to compute the mixture CDF and PDF can significantly exceed the time that is taken to estimate the model parameters. So it is recommended that you use the FULL method only for small data tables.

MEAN In this method, the mixture contains only one distribution, whose scale value is determined by the mean of the linear predictor values that are implied by all the observations. In other words, s_1 is computed as

$$s_1 = \exp\left(\frac{1}{N} \sum_{i=1}^N \hat{\lambda}_i\right)$$

The component's weight d_1 is set to 1.

This method is the fastest because it requires only one CDF or PDF evaluation per observation. The computational complexity is $O(N)$ for N observations.

If you do not specify the DFMIXTURE= option in the SCALEMODEL statement, then this is the default method.

QUANTILE In this method, a certain number of quantiles are chosen from the set of all linear predictor values. If you specify a value of q for the K= option when specifying this method, then $K = q - 1$ and s_k ($k = 1, \dots, K$) is computed as $s_k = \exp(\hat{\lambda}_k)$, where $\hat{\lambda}_k$ is the k th q -quantile from the set $\{\hat{\lambda}_i\}$ ($i = 1, \dots, N$). The weight of each of the components (d_k) is assumed to be 1 for this method.

The default value of q is 2, which implies a one-point mixture that has a distribution whose scale value is equal to the median scale value.

For this method, PROC SEVSELECT needs to sort the N linear predictor values in the set $\{\hat{\lambda}_i\}$; the sorting requires $O(N \log(N))$ computations. Then, computing the mixture estimate of one observation requires $(q - 1)$ CDF or PDF evaluations. Hence, the computational complexity of this method is $O(qN) + O(N \log(N))$ for computing a mixture CDF or PDF of N observations. For $q \ll N$, the QUANTILE method is significantly faster than the FULL method.

RANDOM In this method, a uniform random sample of observations is chosen, and the mixture contains the distributions that are implied by those observations. If you specify a value of r for the K= option when specifying this method, then the size of the sample is r . Hence, $K = r$. If l_j denotes the index of j th observation in the sample ($j = 1, \dots, r$), such that $1 \leq l_j \leq N$, then the scale of k th component distribution in the mixture is $s_k = \exp(\hat{\lambda}_{l_k})$. The weight of each of the components (d_k) is assumed to be 1 for this method.

You can also specify the seed to be used for generating the random sample by using the `SEED=` option for this method. The same sample of observations is used for all models.

Computing a mixture estimate of one observation requires r CDF or PDF evaluations. Hence, the computational complexity of this method is $O(rN)$ for computing a mixture CDF or PDF of N observations. For $r \ll N$, the `RANDOM` method is significantly faster than the `FULL` method.

Empirical Distribution Function Estimation Methods

The empirical distribution function (EDF) is a nonparametric estimate of the cumulative distribution function (CDF) of the distribution. `PROC SEVSELECT` computes EDF estimates for two purposes: to send the estimates to a distribution's `PARMINIT` subroutine in order to initialize the distribution parameters, and to compute the EDF-based statistics of fit.

To reduce the time that it takes to compute the EDF estimates, you can use the `INITSAMPLE` option to specify that only a fraction of the input data be used. If you do not specify the `INITSAMPLE` option and the data table has more than 10,000 valid observations, then a uniform random sample of at most 10,000 observations is used for EDF estimation.

In the distributed mode of execution, in which data are distributed across the grid nodes, the EDF estimates are computed on each node by using the portion of the input data that is located on that node. These local EDF estimates are an approximation of the global EDF estimates, which would have been computed by using the entire input data table. `PROC SEVSELECT` does not compute global EDF estimates. Let X denote a quantity that depends on the EDF estimates. X can be either an EDF-based initial value of a distribution parameter or an EDF-based statistic of fit. `PROC SEVSELECT` estimates X as follows: First, each grid node k computes an estimate X_k by using the local EDF estimates that are computed on that node. Then, the estimate $\hat{X}$ of X is computed as an average of all the X_k values; that is, $\hat{X} = \sum_{i=1}^K X_k$, where K denotes the total number of nodes where the data reside.

This section describes the methods that are used for computing EDF estimates.

Notation

Let there be a set of N observations, each containing a quintuplet of values $(y_i, t_i^l, t_i^r, c_i^r, c_i^l)$, $i = 1, \dots, N$, where y_i is the value of the response variable, t_i^l is the value of the left-truncation threshold, t_i^r is the value of the right-truncation threshold, c_i^r is the value of the right-censoring limit, and c_i^l is the value of the left-censoring limit.

If an observation is not left-truncated, then $t_i^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t_i^l) = 0$. If an observation is not right-truncated, then $t_i^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t_i^r) = 1$. If an observation is not right-censored, then $c_i^r = \tau^l$; so $F(c_i^r) = 0$. If an observation is not left-censored, then $c_i^l = \tau_h$; so $F(c_i^l) = 1$.

Let w_i denote the weight associated with i th observation. If you specify the `WEIGHT` statement, then w_i is the normalized value of the weight variable; otherwise, it is set to 1. The weights are normalized such that they sum up to N .

An indicator function $I[e]$ takes a value of 1 or 0 if the expression e is true or false, respectively.

Estimation Methods

If the response variable is subject to both left-censoring and right-censoring effects and if you explicitly specify the `EMPIRICALCDF=TURNBULL` option, then PROC SEVSELECT uses the Turnbull's method. This section describes methods other than Turnbull's method. For Turnbull's method, see the next section "Turnbull's EDF Estimation Method" on page 886.

The method descriptions assume that all observations are either uncensored or right-censored; that is, each observation is of the form $(y_i, t_i^l, t_i^r, \tau^l, \tau_h)$ or $(y_i, t_i^l, t_i^r, c_i^r, \tau_h)$.

If all observations are either uncensored or left-censored, then each observation is of the form $(y_i, t_i^l, t_i^r, \tau_l, c_i^l)$. It is converted to an observation $(-y_i, -t_i^r, -t_i^l, -c_i^l, \tau_h)$; that is, the signs of all the response variable values are reversed, the new left-truncation threshold is equal to the negative of the original right-truncation threshold, the new right-truncation threshold is equal to the negative of the original left-truncation threshold, and the negative of the original left-censoring limit becomes the new right-censoring limit. With this transformation, each observation is either uncensored or right-censored. The methods described for handling uncensored or right-censored data are now applicable. After the EDF estimates are computed, the observations are transformed back to the original form and EDF estimates are adjusted such $F_n(y_i) = 1 - F_n(-y_i-)$, where $F_n(-y_i-)$ denotes the EDF estimate of the value slightly less than the transformed value $-y_i$.

Further, a set of uncensored or right-censored observations can be converted to a set of observations of the form $(y_i, t_i^l, t_i^r, \delta_i)$, where δ_i is the indicator of right-censoring. $\delta_i = 0$ indicates a right-censored observation, in which case y_i is assumed to record the right-censoring limit c_i^r . $\delta_i = 1$ indicates an uncensored observation, and y_i records the exact observed value. In other words, $\delta_i = I[Y \leq C^r]$ and $y_i = \min(y_i, c_i^r)$.

Given this notation, the EDF is estimated as

$$F_n(y) = \begin{cases} 0 & \text{if } y < y^{(1)} \\ \hat{F}_n(y^{(k)}) & \text{if } y^{(k)} \leq y < y^{(k+1)}, k = 1, \dots, N-1 \\ \hat{F}_n(y^{(N)}) & \text{if } y^{(N)} \leq y \end{cases}$$

where $y^{(k)}$ denotes the k th-order statistic of the set $\{y_i\}$ and $\hat{F}_n(y^{(k)})$ is the estimate computed at that value. The definition of $\hat{F}_n$ depends on the estimation method. You can specify a particular method or let PROC SEVSELECT choose an appropriate method by using the `EMPIRICALCDF=` option in the PROC SEVSELECT statement. Each method computes $\hat{F}_n$ as follows:

NOTURNBULL This is the default method. First, censored observations, if any, are processed as follows:

- An observation that is left-censored but not right-censored is converted to an uncensored observation $(y_i^u, t_i^l, t_i^r, \tau^l, \tau_h)$, where $y_i^u = c_i^l/2$.
- An observation that is both left-censored and right-censored is converted to an uncensored observation $(y_i^u, t_i^l, t_i^r, \tau^l, \tau_h)$, where $y_i^u = (c_i^r + c_i^l)/2$.
- An observation that is right-censored but not left-censored is left unchanged.

If the processed set of observations contains any truncated or right-censored observations, the KAPLANMEIER method is used. Otherwise, the STANDARD method is used.

The observations are modified only for the purpose of computing the EDF estimates. The original censoring information is used by the parameter estimation process.

STANDARD

This method is the standard way of computing EDF. The EDF estimate at observation i is computed as follows:

$$\hat{F}_n(y_i) = \frac{1}{N} \sum_{j=1}^N w_j \cdot I[y_j \leq y_i]$$

If you do not specify any censoring or truncation information, then this method is chosen. If you explicitly specify this method, then PROC SEVSELECT ignores any censoring and truncation information that you specify in the LOSS statement.

The standard error of $\hat{F}_n(y_i)$ is computed by using the normal approximation method:

$$\hat{\sigma}_n(y_i) = \sqrt{\hat{F}_n(y_i)(1 - \hat{F}_n(y_i))/N}$$

KAPLANMEIER

The Kaplan-Meier (KM) estimator, also known as the product-limit estimator, was first introduced by Kaplan and Meier (1958) for censored data. Lynden-Bell (1971) derived a similar estimator for left-truncated data. PROC SEVSELECT uses the definition that combines both censoring and truncation information (Klein and Moeschberger 1997; Lai and Ying 1991).

The EDF estimate at observation i is computed as

$$\hat{F}_n(y_i) = 1 - \prod_{\tau \leq y_i} \left(1 - \frac{n(\tau)}{R_n(\tau)}\right)$$

where $n(\tau)$ and $R_n(\tau)$ are defined as follows:

- $n(\tau) = \sum_{k=1}^N w_k \cdot I[y_k = \tau \text{ and } \tau \leq t_k^r \text{ and } \delta_k = 1]$, which is the number of uncensored observations ($\delta_k = 1$) for which the response variable value is equal to τ and τ is observable according to the right-truncation threshold of that observation ($\tau \leq t_k^r$).
- $R_n(\tau) = \sum_{k=1}^N w_k \cdot I[y_k \geq \tau > t_k^l]$, which is the size (cardinality) of the risk set at τ . The term *risk set* has its origins in survival analysis; it contains the events that are at risk of failure at a particular time, τ . In other words, it contains the events that have survived up to time τ and might fail at or after τ . For PROC SEVSELECT, *time* is equivalent to the magnitude of the event and *failure* is equivalent to an uncensored and observable event, where observable means it satisfies the truncation thresholds.

This method is chosen when you specify at least one form of censoring or truncation.

The standard error of $\hat{F}_n(y_i)$ is computed by using Greenwood's formula (Greenwood 1926):

$$\hat{\sigma}_n(y_i) = \sqrt{(1 - \hat{F}_n(y_i))^2 \cdot \sum_{\tau \leq y_i} \left(\frac{n(\tau)}{R_n(\tau)(R_n(\tau) - n(\tau))} \right)}$$

MODIFIEDKM

The product-limit estimator used by the KAPLANMEIER method does not work well if the risk set size becomes very small. For right-censored data, the size can become

small towards the right tail. For left-truncated data, the size can become small at the left tail and can remain so for the entire range of data. This was demonstrated by Lai and Ying (1991). They proposed a modification to the estimator that ignores the effects due to small risk set sizes.

The EDF estimate at observation i is computed as

$$\hat{F}_n(y_i) = 1 - \prod_{\tau \leq y_i} \left(1 - \frac{n(\tau)}{R_n(\tau)} \cdot I[R_n(\tau) \geq cN^\alpha] \right)$$

where the definitions of $n(\tau)$ and $R_n(\tau)$ are identical to those used for the KAPLAN-MEIER method described previously.

You can specify the values of c and α by using the **C=** and **ALPHA=** options. If you do not specify a value for c , the default value used is $c = 1$. If you do not specify a value for α , the default value used is $\alpha = 0.5$.

As an alternative, you can also specify an absolute lower bound, say L , on the risk set size by using the **RSLB=** option, in which case $I[R_n(\tau) \geq cN^\alpha]$ is replaced by $I[R_n(\tau) \geq L]$ in the definition.

The standard error of $\hat{F}_n(y_i)$ is computed by using Greenwood's formula (Greenwood 1926):

$$\hat{\sigma}_n(y_i) = \sqrt{(1 - \hat{F}_n(y_i))^2 \cdot \sum_{\tau \leq y_i} \left(\frac{n(\tau)}{R_n(\tau)(R_n(\tau) - n(\tau))} \cdot I[R_n(\tau) \geq cN^\alpha] \right)}$$

Turnbull's EDF Estimation Method

If the response variable is subject to both left-censoring and right-censoring effects and if you explicitly specify the **EMPIRICALCDF=TURNBULL** option, then the SEVSELECT procedure uses a method proposed by Turnbull (1976) to compute the nonparametric estimates of the cumulative distribution function. The original Turnbull's method is modified using the suggestions made by Frydman (1994) when truncation effects are present.

Let the input data consist of N observations in the form of quintuplets of values $(y_i, t_i^l, t_i^r, c_i^r, c_i^l)$, $i = 1, \dots, N$ with notation described in the section “**Notation**” on page 883. For each observation, let $A_i = (c_i^r, c_i^l]$ be the censoring interval; that is, the response variable value is known to lie in the interval A_i , but the exact value is not known. If an observation is uncensored, then $A_i = (y_i - \epsilon, y_i]$ for any arbitrarily small value of $\epsilon > 0$. If an observation is censored, then the value y_i is ignored. Similarly, for each observation, let $B_i = (t_i^l, t_i^r]$ be the truncation interval; that is, the observation is drawn from a truncated (conditional) distribution $F(y, B_i) = P(Y \leq y | Y \in B_i)$.

Two sets, L and R , are formed using A_i and B_i as follows:

$$L = \{c_i^r, 1 \leq i \leq N\} \cup \{t_i^r, 1 \leq i \leq N\}$$

$$R = \{c_i^l, 1 \leq i \leq N\} \cup \{t_i^l, 1 \leq i \leq N\}$$

The sets L and R represent the left endpoints and right endpoints, respectively. A set of disjoint intervals $C_j = [q_j, p_j]$, $1 \leq j \leq M$ is formed such that $q_j \in L$ and $p_j \in R$ and $q_j \leq p_j$ and $p_j < q_{j+1}$. The value of M is dependent on the nature of censoring and truncation intervals in the input data. Turnbull (1976)

showed that the maximum likelihood estimate (MLE) of the EDF can increase only inside intervals C_j . In other words, the MLE estimate is constant in the interval (p_j, q_{j+1}) . The likelihood is independent of the behavior of F_n inside any of the intervals C_j . Let s_j denote the increase in F_n inside an interval C_j . Then, the EDF estimate is as follows:

$$F_n(y) = \begin{cases} 0 & \text{if } y < q_1 \\ \sum_{k=1}^j s_k & \text{if } p_j < y < q_{j+1}, 1 \leq j \leq M-1 \\ 1 & \text{if } y > p_M \end{cases}$$

PROC SEVSELECT computes the estimates $F_n(p_j+) = F_n(q_{j+1}-) = \sum_{k=1}^j s_k$ at points p_j and q_{j+1} and computes $F_n(q_1-) = 0$ at point q_1 , where $F_n(x+)$ denotes the limiting estimate at a point that is infinitesimally larger than x when approaching x from values larger than x and where $F_n(x-)$ denotes the limiting estimate at a point that is infinitesimally smaller than x when approaching x from values smaller than x .

PROC SEVSELECT uses the expectation-maximization (EM) algorithm proposed by Turnbull (1976), who referred to the algorithm as the self-consistency algorithm. By default, the algorithm runs until one of the following criteria is met:

- Relative-error criterion: The maximum relative error between the two consecutive estimates of s_j falls below a threshold ϵ . If l indicates an index of the current iteration, then this can be formally stated as

$$\arg \max_{1 \leq j \leq M} \left\{ \frac{|s_j^l - s_j^{l-1}|}{s_j^{l-1}} \right\} \leq \epsilon$$

You can control the value of ϵ by specifying the **EPS=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default value is 1.0E-8.

- Maximum-iteration criterion: The number of iterations exceeds an upper limit that you specify for the **MAXITER=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default number of maximum iterations is 500.

The self-consistent estimates obtained in this manner might not be maximum likelihood estimates. Gentleman and Geyer (1994) suggested the use of the Kuhn-Tucker conditions for the maximum likelihood problem to ensure that the estimates are MLE. If you specify the **ENSUREMLE** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement, then PROC SEVSELECT computes the Kuhn-Tucker conditions at the end of each iteration to determine whether the estimates $\{s_j\}$ are MLE. If you do not specify any truncation effects, then the Kuhn-Tucker conditions derived by Gentleman and Geyer (1994) are used. If you specify any truncation effects, then PROC SEVSELECT uses modified Kuhn-Tucker conditions that account for the truncation effects. An integral part of checking the conditions is to determine whether an estimate s_j is zero or whether an estimate of the Lagrange multiplier or the reduced gradient associated with the estimate s_j is zero. PROC SEVSELECT declares these values to be zero if they are less than or equal to a threshold δ . You can control the value of δ by specifying the **ZEROPROB=** suboption of the EDF=TURNBULL option in the PROC SEVSELECT statement. The default value is 1.0E-8. The algorithm continues until the Kuhn-Tucker conditions are satisfied or the number of iterations exceeds the upper limit. The relative-error criterion stated previously is not used when you specify the **ENSUREMLE** option.

The standard errors for Turnbull's EDF estimates are computed by using the asymptotic theory of the maximum likelihood estimators (MLE), even though the final estimates might not be MLE. Turnbull's

estimator essentially attempts to maximize the likelihood L , which depends on the parameters s_j ($j = 1, \dots, M$). Let $\mathbf{s} = \{s_j\}$ denote the set of these parameters. If $\mathbf{G}(\mathbf{s}) = \nabla^2(-\log(L(\mathbf{s})))$ denotes the Hessian matrix of the negative of log likelihood, then the variance-covariance matrix of $\mathbf{s}$ is estimated as $\hat{\mathbf{C}}(\mathbf{s}) = \mathbf{G}^{-1}(\mathbf{s})$. Given this matrix, the standard error of $F_n(y)$ is computed as

$$\sigma_n(y) = \sqrt{\sum_{k=1}^j \left(\hat{C}_{kk} + 2 \cdot \sum_{l=1}^{k-1} \hat{C}_{kl} \right)}, \text{ if } p_j < y < q_{j+1}, 1 \leq j \leq M-1$$

The standard error is undefined outside of these intervals.

EDF Estimates and Truncation

If you specify truncation, then the estimate $\hat{F}_n(y)$ that is computed by any method other than the STANDARD method is a *conditional* estimate. In other words, $\hat{F}_n(y) = \Pr(Y \leq y | \tau_G < Y \leq \tau_H)$, where G and H denote the (unknown) distribution functions of the left-truncation threshold variable T^l and the right-truncation threshold variable T^r , respectively, τ_G denotes the smallest left-truncation threshold with a nonzero cumulative probability, and τ_H denotes the largest right-truncation threshold with a nonzero cumulative probability. Formally, $\tau_G = \inf\{s : G(s) > 0\}$ and $\tau_H = \sup\{s : H(s) > 0\}$. For computational purposes, PROC SEVSELECT estimates τ_G and τ_H by $t_{\min}^l$ and $t_{\max}^r$, respectively, defined as

$$t_{\min}^l = \min\{t_k^l : 1 \leq k \leq N\}$$

$$t_{\max}^r = \max\{t_k^r : 1 \leq k \leq N\}$$

These estimates of $t_{\min}^l$ and $t_{\max}^r$ are used to compute the conditional estimates of the CDF as described in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

If you specify left-truncation *with* the probability of observability p , then PROC SEVSELECT uses the additional information provided by p to compute an estimate of the EDF that is not conditional on the left-truncation information. In particular, for each left-truncated observation i with response variable value y_i and truncation threshold t_i^l , an observation j is added with *weight* $w_j = (1 - p)/p$ and $y_j = t_j^l$. Each added observation is assumed to be uncensored and untruncated. Then, your specified EDF method is used by assuming no left-truncation. The EDF estimate that is obtained using this method is not conditional on the left-truncation information. For the KAPLANMEIER and MODIFIEDKM methods with uncensored or right-censored data, definitions of $n(\tau)$ and $R_n(\tau)$ are modified to account for the added observations. If N^a denotes the total number of observations including the added observations, then $n(\tau)$ is defined as $n(\tau) = \sum_{k=1}^{N^a} w_k I[y_k = \tau \text{ and } \tau \leq t_k^r \text{ and } \delta_k = 1]$, and $R_n(\tau)$ is defined as $R_n(\tau) = \sum_{k=1}^{N^a} w_k I[y_k \geq \tau]$. In the definition of $R_n(\tau)$, the left-truncation information is not used, because it was used along with p to add the observations.

If the original data are a combination of left- and right-censored data and if you specify the EMPIRICALCDF=TURNBULL option, then Turnbull’s method is applied to the appended set that contains no left-truncated observations.

Supplying EDF Estimates to Functions and Subroutines

The parameter initialization subroutines in distribution models and some predefined utility functions require EDF estimates. For more information, see the sections “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895 and “[Predefined Utility Functions](#)” on page 908.

PROC SEVSELECT supplies the EDF estimates to these subroutines and functions by using two arrays, x and F , the dimension of each array, and a type of the EDF estimates. The type identifies how the EDF estimates are computed and stored. They are as follows:

- Type 1 specifies that EDF estimates are computed using the STANDARD method; that is, the data that are used for estimation are neither censored nor truncated.
- Type 2 specifies that EDF estimates are computed using either the KAPLANMEIER or the MODIFIEDKM method; that is, the data that are used for estimation are subject to truncation and one type of censoring (left or right, but not both).
- Type 3 specifies that EDF estimates are computed using the TURNBULL method; that is, the data that are used for estimation are subject to both left- and right-censoring. The data might or might not be truncated.

For Types 1 and 2, the EDF estimates are stored in arrays x and F of dimension N such that the following holds,

$$F_n(y) = \begin{cases} 0 & \text{if } y < x[1] \\ F[k] & \text{if } x[k] \leq y < x[k+1], k = 1, \dots, N-1 \\ F[N] & \text{if } x[N] \leq y \end{cases}$$

where $[k]$ denotes k th element of the array ($[1]$ denotes the first element of the array).

For Type 3, the EDF estimates are stored in arrays x and F of dimension N such that the following holds:

$$F_n(y) = \begin{cases} 0 & \text{if } y < x[1] \\ \text{undefined} & \text{if } x[2k-1] \leq y < x[2k], k = 1, \dots, (N-1)/2 \\ F[2k] = F[2k+1] & \text{if } x[2k] \leq y < x[2k+1], k = 1, \dots, (N-1)/2 \\ F[N] & \text{if } x[N] \leq y \end{cases}$$

Although the behavior of EDF is theoretically undefined for the interval $[x[2k-1], x[2k])$, for computational purposes, all predefined functions and subroutines assume that the EDF increases linearly from $F[2k-1]$ to $F[2k]$ in that interval if $x[2k-1] < x[2k]$. If $x[2k-1] = x[2k]$, which can happen when the EDF is estimated from a combination of uncensored and interval-censored data, the predefined functions and subroutines assume that $F_n(x[2k-1]) = F_n(x[2k]) = F[2k]$.

Statistics of Fit

PROC SEVSELECT computes and reports various statistics of fit to indicate how well the estimated model fits the data. The statistics belong to two categories: likelihood-based statistics and EDF-based statistics. Neg2LogLike, AIC, AICC, and SBC are likelihood-based statistics, and KS, AD, and CvM are EDF-based statistics.

In the distributed mode of execution, in which data are distributed across the grid nodes, the EDF estimates are computed by using the local data. The EDF-based statistics are computed by using these local EDF estimates. The reported value of each EDF-based statistic is an average of the values of the statistic that are computed by all the grid nodes where the data reside. Also, for large data tables, in both single-machine and distributed modes of execution, the EDF estimates are computed by using a fraction of the input data that is governed by either the `INITSAMPLE` option or the default sample size. Because of this nature of computing the EDF estimates, the EDF-based statistics of fit are an approximation of the values that would have been computed if the entire input data table were used for computing the EDF estimates. So the values that are reported for EDF-based statistics should be used only for comparing different models. The reported values should not be interpreted as true estimates of the corresponding statistics.

The likelihood-based statistics are reported for the entire input data in both single-machine and distributed modes of execution.

The following subsections provide definitions of each category of statistics.

Likelihood-Based Statistics of Fit

Let $y_i, i = 1, \dots, N$, denote the response variable values. Let L be the likelihood as defined in the section “[Likelihood Function](#)” on page 875. Let p denote the number of model parameters that are estimated. Note that $p = p_d + (k - k_r)$, where p_d is the number of distribution parameters, k is the number of all regression parameters, and k_r is the number of regression parameters that are found to be linearly dependent (redundant) on other regression parameters. By default, the value of p_d includes the distribution parameters that you have defined as constant by using the `dist_CONSTANTPARM` subroutine. You can exclude them by specifying the `NOCONSTFITSTATS` option in the PROC SEVSELECT statement. Given this notation, the likelihood-based statistics are defined as follows:

Neg2LogLike The log likelihood is reported as

$$\text{Neg2LogLike} = -2 \log(L)$$

The multiplying factor -2 makes it easy to compare it to the other likelihood-based statistics. A model that has a smaller value of Neg2LogLike is deemed better.

AIC Akaike’s information criterion (AIC) is defined as

$$\text{AIC} = -2 \log(L) + 2p$$

A model that has a smaller AIC value is deemed better.

AICC The corrected Akaike’s information criterion (AICC) is defined as

$$\text{AICC} = -2 \log(L) + \frac{2Np}{N - p - 1}$$

A model that has a smaller AICC value is deemed better. It corrects the finite-sample bias that AIC has when N is small compared to p . AICC is related to AIC as

$$\text{AICC} = \text{AIC} + \frac{2p(p+1)}{N-p-1}$$

As N becomes large compared to p , AICC converges to AIC. AICC is usually recommended over AIC as a model selection criterion.

SBC

The Schwarz Bayesian information criterion (SBC) is defined as

$$\text{SBC} = -2 \log(L) + p \log(N)$$

A model that has a smaller SBC value is deemed better.

EDF-Based Statistics

This class of statistics is based on the difference between the estimate of the cumulative distribution function (CDF) and the estimate of the empirical distribution function (EDF). A model that has a smaller value of the chosen EDF-based statistic is deemed better.

Let $y_i, i = 1, \dots, N$, denote the sample of N values of the response variable. Let w_i denote the normalized weight of the i th observation. If w_i^o denotes the original, unnormalized weight of the i th observation, then $w_i = N w_i^o / (\sum_{i=1}^N w_i^o)$. Let N_u denote the number of observations with unique (nonduplicate) values of the response variable. Let $W_i = \sum_{j=1}^N w_j I[y_j = y_i]$ denote the total weight of observations with a value y_i , where I is an indicator function. Let $r_i = \sum_{j=1}^N w_j I[y_j \leq y_i]$ denote the total weight of observations with a value less than or equal to y_i . Let $W = \sum_{i=1}^{N_u} W_i$ denote the total weight of all observations. Use of normalized weights implies that $W = N$.

Let $F_n(y_i)$ denote the EDF estimate that is computed by using the method that you specify in the **EMPIRICALCDF=** option. Let $Z_i = \hat{F}(y_i)$ denote the estimate of the CDF. Let $F_n(Z_i)$ denote the EDF estimate of Z_i values that are computed using the same method that is used to compute the EDF of y_i values. Using the probability integral transformation, if $F(y)$ is the true distribution of the random variable Y , then the random variable $Z = F(y)$ is uniformly distributed between 0 and 1 (D'Agostino and Stephens 1986, Ch. 4). Thus, comparing $F_n(y_i)$ with $\hat{F}(y_i)$ is equivalent to comparing $F_n(Z_i)$ with $\hat{F}(Z_i) = Z_i$ (uniform distribution).

Note the following two points regarding which CDF estimates are used for computing the test statistics:

- If you specify regression effects, then the CDF estimates Z_i that are used for computing the EDF test statistics are from a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.
- If the EDF estimates are conditional because of the truncation information, then each unconditional estimate Z_i is converted to a conditional estimate using the method described in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

In the following, it is assumed that Z_i denotes an appropriate estimate of the CDF if you specify any truncation or regression effects. Given this, the EDF-based statistics of fit are defined as follows:

KS The Kolmogorov-Smirnov (KS) statistic computes the largest vertical distance between the CDF and the EDF. It is formally defined as follows:

$$KS = \sup_y |F_n(y) - F(y)|$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$\begin{aligned} D^+ &= \max_i \left(\frac{r_i}{W} - Z_i \right) \\ D^- &= \max_i \left(Z_i - \frac{r_{i-1}}{W} \right) \\ KS &= \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}} \end{aligned}$$

Note that r_0 is assumed to be 0.

If the method used to compute the EDF is any method other than the STANDARD method, then the following formula is used:

$$\begin{aligned} D^+ &= \max_i (F_n(Z_i) - Z_i), \text{ if } F_n(Z_i) \geq Z_i \\ D^- &= \max_i (Z_i - F_n(Z_i)), \text{ if } F_n(Z_i) < Z_i \\ KS &= \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}} \end{aligned}$$

AD The Anderson-Darling (AD) statistic is a quadratic EDF statistic that is proportional to the expected value of the weighted squared difference between the EDF and CDF. It is formally defined as follows:

$$AD = N \int_{-\infty}^{\infty} \frac{(F_n(y) - F(y))^2}{F(y)(1 - F(y))} dF(y)$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$AD = -W - \frac{1}{W} \sum_{i=1}^{N_u} W_i [(2r_i - 1) \log(Z_i) + (2W + 1 - 2r_i) \log(1 - Z_i)]$$

If the method used to compute the EDF is any method other than the STANDARD method, then the statistic can be computed by using the following two pieces of information:

- If the EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM methods, then EDF is a step function such that the estimate $F_n(z)$ is a constant equal to $F_n(Z_{i-1})$ in interval $[Z_{i-1}, Z_i]$. If the EDF estimates are computed using the TURNBULL method, then there are two types of intervals: one in which the EDF curve is constant and the other in which the EDF curve is theoretically undefined. For computational purposes, it is assumed that the EDF curve is linear for the latter type of the interval. For each method, the EDF estimate $F_n(y)$ at y can be written as

$$F_n(z) = F_n(Z_{i-1}) + S_i(z - Z_{i-1}), \text{ for } z \in [Z_{i-1}, Z_i]$$

where S_i is the slope of the line defined as

$$S_i = \frac{F_n(Z_i) - F_n(Z_{i-1})}{Z_i - Z_{i-1}}$$

For the KAPLANMEIER or MODIFIEDKM method, $S_i = 0$ in each interval.

- Using the probability integral transform $z = F(y)$, the formula simplifies to

$$AD = N \int_{-\infty}^{\infty} \frac{(F_n(z) - z)^2}{z(1-z)} dz$$

The computation formula can then be derived from the approximation,

$$\begin{aligned} AD &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(z) - z)^2}{z(1-z)} dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2}{z(1-z)} dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(P_i - Q_i z)^2}{z(1-z)} dz \end{aligned}$$

where $P_i = F_n(Z_{i-1}) - S_i Z_{i-1}$, $Q_i = 1 - S_i$, and K is the number of points at which the EDF estimate are computed. For the TURNBULL method, $K = 2k$ for some k .

Assuming $Z_0 = 0$, $Z_{K+1} = 1$, $F_n(0) = 0$, and $F_n(Z_K) = 1$ yields the computation formula,

$$\begin{aligned} AD &= -N(Z_1 + \log(1 - Z_1) + \log(Z_K) + (1 - Z_K)) \\ &\quad + N \sum_{i=2}^K [P_i^2 A_i - (Q_i - P_i)^2 B_i - Q_i^2 C_i] \end{aligned}$$

where $A_i = \log(Z_i) - \log(Z_{i-1})$, $B_i = \log(1 - Z_i) - \log(1 - Z_{i-1})$, and $C_i = Z_i - Z_{i-1}$.

If EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM method, then $P_i = F_n(Z_{i-1})$ and $Q_i = 1$, which simplifies the formula as

$$\begin{aligned} AD &= -N(1 + \log(1 - Z_1) + \log(Z_K)) \\ &\quad + N \sum_{i=2}^K [F_n(Z_{i-1})^2 A_i - (1 - F_n(Z_{i-1}))^2 B_i] \end{aligned}$$

CvM The Cramér–von Mises (CvM) statistic is a quadratic EDF statistic that is proportional to the expected value of the squared difference between the EDF and CDF. It is formally defined as follows:

$$CvM = N \int_{-\infty}^{\infty} (F_n(y) - F(y))^2 dF(y)$$

If the STANDARD method is used to compute the EDF, then the following formula is used:

$$CvM = \frac{1}{12W} + \sum_{i=1}^{N_u} W_i \left(Z_i - \frac{(2r_i - 1)}{2W} \right)^2$$

If the method used to compute the EDF is any method other than the STANDARD method, then the statistic can be computed by using the following two pieces of information:

- As described previously for the AD statistic, the EDF estimates are assumed to be piecewise linear such that the estimate $F_n(y)$ at y is

$$F_n(z) = F_n(Z_{i-1}) + S_i(z - Z_{i-1}), \text{ for } z \in [Z_{i-1}, Z_i]$$

where S_i is the slope of the line defined as

$$S_i = \frac{F_n(Z_i) - F_n(Z_{i-1})}{Z_i - Z_{i-1}}$$

For the KAPLANMEIER or MODIFIEDKM method, $S_i = 0$ in each interval.

- Using the probability integral transform $z = F(y)$, the formula simplifies to

$$\text{CvM} = N \int_{-\infty}^{\infty} (F_n(z) - z)^2 dz$$

The computation formula can then be derived from the following approximation,

$$\begin{aligned} \text{CvM} &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(z) - z)^2 dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2 dz \\ &= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (P_i - Q_i z)^2 dz \end{aligned}$$

where $P_i = F_n(Z_{i-1}) - S_i Z_{i-1}$, $Q_i = 1 - S_i$, and K is the number of points at which the EDF estimate are computed. For the TURNBULL method, $K = 2k$ for some k .

Assuming $Z_0 = 0$, $Z_{K+1} = 1$, and $F_n(0) = 0$ yields the following computation formula,

$$\text{CvM} = N \frac{Z_1^3}{3} + N \sum_{i=2}^{K+1} \left[P_i^2 A_i - P_i Q_i B_i - \frac{Q_i^2}{3} C_i \right]$$

where $A_i = Z_i - Z_{i-1}$, $B_i = Z_i^2 - Z_{i-1}^2$, and $C_i = Z_i^3 - Z_{i-1}^3$.

If EDF estimates are computed using the KAPLANMEIER or MODIFIEDKM method, then $P_i = F_n(Z_{i-1})$ and $Q_i = 1$, which simplifies the formula as

$$\text{CvM} = \frac{N}{3} + N \sum_{i=2}^{K+1} [F_n(Z_{i-1})^2 (Z_i - Z_{i-1}) - F_n(Z_{i-1})(Z_i^2 - Z_{i-1}^2)]$$

which is similar to the formula proposed by Koziol and Green (1976).

Defining a Severity Distribution Model with the FCMP Procedure

A *severity distribution model* consists of a set of functions and subroutines that are defined using the FCMP procedure. The FCMP procedure is part of Base SAS software. Each function or subroutine must be named as `<distribution-name>_<keyword>`, where *distribution-name* is the identifying short name of the distribution and *keyword* identifies one of the functions or subroutines. The total length of the name should not exceed 32. Each function or subroutine must have a specific signature, which consists of the number of arguments, sequence and types of arguments, and return value type. The summary of all the recognized function and subroutine names and their expected behavior is given in [Table 15.6](#).

Consider the following points when you define a distribution model:

- When you define a function or subroutine requiring parameter arguments, the names and order of those arguments must be the same. Arguments other than the parameter arguments can have any name, but they must satisfy the requirements on their type and order.
- When the SEVSELECT procedure invokes any function or subroutine, it provides the necessary input values according to the specified signature, and expects the function or subroutine to prepare the output and return it according to the specification of the return values in the signature.
- You can use most of the SAS programming statements and SAS functions that you can use in a DATA step for defining the FCMP functions and subroutines. However, there are a few differences in the capabilities of the DATA step and the FCMP procedure. To learn more, see the documentation of the FCMP procedure in the [Base SAS Procedures Guide](#).
- You must specify either the PDF or the LOGPDF function. Similarly, you must specify either the CDF or the LOGCDF function. All other functions are optional, except when necessary for correct definition of the distribution. It is strongly recommended that you define the PARMINIT subroutine to provide a good set of initial values for the parameters. The information that PROC SEVSELECT provides to the PARMINIT subroutine enables you to use popular initialization approaches based on the method of moments and the method of percentile matching, but you can implement any algorithm to initialize the parameters by using the values of the response variable and the estimate of its empirical distribution function.
- The LOWERBOUNDS subroutines should be defined if the lower bound on at least one distribution parameter is different from the default lower bound of 0. If you define a LOWERBOUNDS subroutine but do not set a lower bound for some parameter inside the subroutine, then that parameter is assumed to have no lower bound (or a lower bound of $-\infty$). Hence, it is recommended that you explicitly return the lower bound for each parameter when you define the LOWERBOUNDS subroutine.
- The UPPERBOUNDS subroutines should be defined if the upper bound on at least one distribution parameter is different from the default upper bound of ∞ . If you define an UPPERBOUNDS subroutine but do not set an upper bound for some parameter inside the subroutine, then that parameter is assumed to have no upper bound (or a upper bound of ∞). Hence, it is recommended that you explicitly return the upper bound for each parameter when you define the UPPERBOUNDS subroutine.
- If you want to use the distribution in a model with regression effects, then make sure that the first parameter of the distribution is the scale parameter itself or a log-transformed scale parameter. If the first parameter is a log-transformed scale parameter, then you must define the SCALETRANSFORM function.

- In general, it is not necessary to define the gradient and Hessian functions, because the SEVSELECT procedure uses an internal system to evaluate the required derivatives. The internal system typically computes the derivatives analytically. But it might not be able to do so if your function definitions use other functions that it cannot differentiate analytically. In such cases, derivatives are approximated using a finite difference method and a note is written to the SAS log to indicate the components that are differentiated using such approximations. PROC SEVSELECT does reasonably well with these finite difference approximations. But, if you know of a way to compute the derivatives of such components analytically, then you should define the gradient and Hessian functions.

In order to use your distribution with PROC SEVSELECT, you need to record the FCMP library that contains the functions and subroutines for your distribution and other FCMP libraries that contain FCMP functions or subroutines used within your distribution's functions and subroutines. Specify all those libraries in the CMPLIB= system option by using the OPTIONS global statement. For more information about the OPTIONS statement, see *SAS Global Statements: Reference*. For more information about the CMPLIB= system option, see *SAS System Options: Reference*.

Each predefined distribution mentioned in the section “[Predefined Distributions](#)” on page 863 has a distribution model associated with it. The functions and subroutines of all those models are available in the Sashelp.Svrtldist library. The order of the parameters in the signatures of the functions and subroutines is the same as listed in [Table 15.4](#). You do not need to use the CMPLIB= option in order to use the predefined distributions with PROC SEVSELECT. However, if you need to use the functions or subroutines of the predefined distributions in SAS statements other than the PROC SEVSELECT step (such as in a DATA step), then specify the Sashelp.Svrtldist library in the CMPLIB= system option by using the OPTIONS global statement prior to using them.

[Table 15.6](#) shows functions and subroutines that define a distribution model, and subsections after the table provide more detail. The functions are listed in alphabetical order of the keyword suffix.

Table 15.6 List of Functions and Subroutines That Define a Distribution Model

Name	Type	Required	Expected to Return
<i>dist_CDF</i>	Function	YES ¹	Cumulative distribution function value
<i>dist_CDFGRADIENT</i>	Subroutine	NO	Gradient of the CDF
<i>dist_CDFHESSIAN</i>	Subroutine	NO	Hessian of the CDF
<i>dist_CONSTANTPARM</i>	Subroutine	NO	Constant parameters
<i>dist_DESCRIPTION</i>	Function	NO	Description of the distribution
<i>dist_LOGCDF</i>	Function	YES ¹	Log of cumulative distribution function value
<i>dist_LOGCDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGCDF
<i>dist_LOGCDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGCDF
<i>dist_LOGPDF</i>	Function	YES ²	Log of probability density function value
<i>dist_LOGPDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGPDF
<i>dist_LOGPDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGPDF
<i>dist_LOGSDF</i>	Function	NO	Log of survival function value
<i>dist_LOGSDFGRADIENT</i>	Subroutine	NO	Gradient of the LOGSDF
<i>dist_LOGSDFHESSIAN</i>	Subroutine	NO	Hessian of the LOGSDF
<i>dist_LOWERBOUNDS</i>	Subroutine	NO	Lower bounds on parameters
<i>dist_PARMINIT</i>	Subroutine	NO	Initial values for parameters
<i>dist_PDF</i>	Function	YES ²	Probability density function value
<i>dist_PDFGRADIENT</i>	Subroutine	NO	Gradient of the PDF
<i>dist_PDFHESSIAN</i>	Subroutine	NO	Hessian of the PDF
<i>dist_QUANTILE</i>	Function	NO	Quantile for a particular CDF value
<i>dist_SCALETRANSFORM</i>	Function	NO	Type of relationship between the first distribution parameter and the scale parameter
<i>dist_SDF</i>	Function	NO	Survival function value
<i>dist_SDFGRADIENT</i>	Subroutine	NO	Gradient of the SDF
<i>dist_SDFHESSIAN</i>	Subroutine	NO	Hessian of the SDF
<i>dist_UPPERBOUNDS</i>	Subroutine	NO	Upper bounds on parameters

Notes:

1. Either the *dist_CDF* or the *dist_LOGCDF* function must be defined.
2. Either the *dist_PDF* or the *dist_LOGPDF* function must be defined.

The signature syntax and semantics of each function or subroutine are as follows:

dist_CDF

defines a function that returns the value of the cumulative distribution function (CDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* YES
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*
 - x Numeric value of the random variable at which the CDF value should be evaluated
 - $p1$ Numeric value of the first parameter
 - $p2$ Numeric value of the second parameter
 - $\vdots$
 - pm Numeric value of the m th parameter
- *Return value:* Numeric value that contains the CDF value $F(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \dots, p_m) = F\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \dots, p_m) = F\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_CDF(x, P1, P2);
  /* Code to compute CDF by using x, P1, and P2 */

  F = <computed CDF>;
  return (F);
endsub;
```

dist_CONSTANTPARM

defines a subroutine that specifies constant parameters. A parameter is *constant* if it is required for defining a distribution but is not subject to optimization in PROC SEVSELECT. Constant parameters are required to be part of the model in order to compute the PDF or the CDF of the distribution. Typically, values of these parameters are known a priori or estimated using some means other than the maximum likelihood method used by PROC SEVSELECT. You can estimate them inside the *dist_PARMINIT* subroutine. Once initialized, the parameters remain constant in the context of PROC SEVSELECT; that is, they retain their initial value. PROC SEVSELECT estimates only the nonconstant parameters.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: k , where k is the number of constant parameters
- *Sequence and type of arguments*:

constant parameter 1 Name of the first constant parameter
 :
 constant parameter k Name of the k th constant parameter

- *Return value*: None

Here is a sample structure of the subroutine for a distribution named ‘FOO’ that has P3 and P5 as its constant parameters, assuming that distribution has at least three parameters:

```
subroutine FOO_CONSTANTPARM(p5, p3);
endsub;
```

Note the following points when you specify the constant parameters:

- At least one distribution parameter must be free to be optimized; that is, if a distribution has total m parameters, then k must be strictly less than m .
- If you want to use this distribution for modeling regression effects, then the first parameter must not be a constant parameter.
- The order of arguments in the signature of this subroutine does not matter as long as each argument’s name matches the name of one of the parameters that are defined in the signature of the *dist_PDF* function.
- The constant parameters must be specified in signatures of all the functions and subroutines that accept distribution parameters as their arguments.
- You must provide a nonmissing initial value for each constant parameter by using one of the supported parameter initialization methods.

*dist_***DESCRIPTION**

defines a function that returns a description of the distribution.

- *Type*: Function
- *Required*: NO
- *Number of arguments*: None
- *Sequence and type of arguments*: Not applicable
- *Return value*: Character value containing a description of the distribution

Here is a sample structure of the function for a distribution named ‘FOO’:

```

function FOO_DESCRIPTION() $48;
  length desc $48;
  desc = "A model for a continuous distribution named foo";
  return (desc);
endsub;

```

dist_LOGcore

defines a function that returns the natural logarithm of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, or SDF.

- *Type*: Function
- *Required*: YES only if *core* is PDF or CDF and you have not defined that *core* function; otherwise, NO
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:
 - x Numeric value of the random variable at which the natural logarithm of the *core* function should be evaluated
 - p1 Numeric value of the first parameter
 - p2 Numeric value of the second parameter
 - ⋮
 - pm Numeric value of the m th parameter
- *Return value*: Numeric value that contains the natural logarithm of the *core* function

Here is a sample structure of the function for the core function PDF of a distribution named 'FOO':

```

function FOO_LOGPDF(x, P1, P2);
  /* Code to compute LOGPDF by using x, P1, and P2 */

  l = <computed LOGPDF>;
  return (l);
endsub;

```

dist_LOWERBOUNDS

defines a subroutine that returns lower bounds for the parameters of the distribution. If this subroutine is not defined for a particular distribution, then the SEVSELECT procedure assumes a lower bound of 0 for each parameter. If a lower bound of l_i is returned for a parameter p_i , then the SEVSELECT procedure assumes that $l_i < p_i$ (strict inequality). If a missing value is returned for some parameter, then the SEVSELECT procedure assumes that there is no lower bound for that parameter (equivalent to a lower bound of $-\infty$).

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: m , where m is the number of distribution parameters
- *Sequence and type of arguments*:

- | | |
|----|---|
| p1 | Output argument that returns the lower bound on the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
| p2 | Output argument that returns the lower bound on the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
| ⋮ | |
| pm | Output argument that returns the lower bound on the <i>m</i> th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |
- *Return value:* The results, lower bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_LOWERBOUNDS(p1, p2);
    outargs p1, p2;

    p1 = <lower bound for P1>;
    p2 = <lower bound for P2>;
endsub;
```

*dist_***PARMINIT**

defines a subroutine that returns the initial values for the distribution's parameters given an empirical distribution function (EDF) estimate.

- *Type:* Subroutine
- *Required:* NO
- *Number of arguments:* $m + 4$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

- | | |
|-------|---|
| dim | Input numeric value that contains the dimension of the x, nx, and F array arguments. |
| x{*} | Input numeric array of dimension <i>dim</i> that contains values of the random variables at which the EDF estimate is available. It can be assumed that x contains values in an increasing order. In other words, if $i < j$, then $x[i] < x[j]$. |
| nx{*} | Input numeric array of dimension <i>dim</i> . Each $nx[i]$ contains the number of observations in the original data that have the value $x[i]$. |
| F{*} | Input numeric array of dimension <i>dim</i> . Each $F[i]$ contains the EDF estimate for $x[i]$. This estimate is computed by the SEVSELECT procedure based on the options that you specify in the LOSS statement and the EMPIRICALCDF= option. |
| Ftype | Input numeric value that contains the type of the EDF estimate that is stored in x and F. For definitions of types, see the section " Supplying EDF Estimates to Functions and Subroutines " on page 889. |
| p1 | Output argument that returns the initial value of the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition. |

p2 Output argument that returns the initial value of the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
 ⋮

pm Output argument that returns the initial value of the m th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.

- *Return value:* The results, initial values of the parameters, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_PARMINIT(dim, x{*}, nx{*}, F{*}, Ftype, p1, p2);
    outargs p1, p2;

    /* Code to initialize values of P1 and P2 by using
       dim, x, nx, and F */

    p1 = <initial value for p1>;
    p2 = <initial value for p2>;
endsub;
```

dist_PDF

defines a function that returns the value of the probability density function (PDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* YES
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

x Numeric value of the random variable at which the PDF value should be evaluated

p1 Numeric value of the first parameter

p2 Numeric value of the second parameter

⋮

pm Numeric value of the m th parameter

- *Return value:* Numeric value that contains the PDF value $f(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \dots, p_m) = \frac{1}{p_1} f\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \dots, p_m) = \frac{1}{\exp(p_1)} f\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_PDF(x, P1, P2);
  /* Code to compute PDF by using x, P1, and P2 */

  f = <computed PDF>;
  return (f);
endsub;
```

*dist_***QUANTILE**

defines a function that returns the quantile of the distribution at the specified value of the CDF for the specified values of distribution parameters.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*
 - cdf Numeric value of the cumulative distribution function (CDF) for which the quantile should be evaluated
 - p1 Numeric value of the first parameter
 - p2 Numeric value of the second parameter
 - ⋮
 - pm Numeric value of the m th parameter
- *Return value:* Numeric value that contains the quantile $F^{-1}(\text{cdf}; p_1, p_2, \dots, p_m)$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_QUANTILE(c, P1, P2);
  /* Code to compute quantile by using c, P1, and P2 */

  Q = <computed quantile>;
  return (Q);
endsub;
```

*dist_***SCALETRANSFORM**

defines a function that returns a keyword to identify the transform that needs to be applied to the scale parameter to convert it to the first parameter of the distribution.

If you want to use this distribution for modeling regression effects, then the first parameter of this distribution must be a scale parameter. However, for some distributions, a typical or convenient parameterization might not have a scale parameter, but one of the parameters can be a simple transform of the scale parameter. As an example, consider a typical parameterization of the lognormal distribution with two parameters, location μ and shape σ , for which the PDF is defined as follows:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$$

You can reparameterize this distribution to contain a parameter θ instead of the parameter μ such that $\mu = \log(\theta)$. The parameter θ would then be a scale parameter. However, if you want to specify the distribution in terms of μ and σ (which is a more recognized form of the lognormal distribution) and still allow it as a candidate distribution for estimating regression effects, then instead of writing another distribution with parameters θ and σ , you can simply define the distribution with μ as the first parameter and specify that it is the logarithm of the scale parameter.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* None
- *Sequence and type of arguments:* Not applicable
- *Return value:* Character value that contains one of the following keywords:

LOG	specifies that the first parameter is the logarithm of the scale parameter.
IDENTITY	specifies that the first parameter is a scale parameter without any transformation.

If you do not specify this function, then the IDENTITY transform is assumed.

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SCALETRANSFORM() $8;
  length xform $8;
  xform = "IDENTITY";
  return (xform);
endsub;
```

dist_SDF

defines a function that returns the value of the survival distribution function (SDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type:* Function
- *Required:* NO
- *Number of arguments:* $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments:*

x	Numeric value of the random variable at which the SDF value should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
⋮	
pm	Numeric value of the m th parameter

- *Return value:* Numeric value that contains the SDF value $S(x; p_1, p_2, \dots, p_m)$

If you want to consider this distribution as a candidate distribution when estimating a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \dots, p_m) = S\left(\frac{x}{p_1}; 1, p_2, \dots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \dots, p_m) = S\left(\frac{x}{\exp(p_1)}; 0, p_2, \dots, p_m\right)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SDF(x, P1, P2);
  /* Code to compute SDF by using x, P1, and P2 */

  S = <computed SDF>;
  return (S);
endsub;
```

dist_UPPERBOUNDS

defines a subroutine that returns upper bounds for the parameters of the distribution. If this subroutine is not defined for a particular distribution, then the SEVSELECT procedure assumes that there is no upper bound for any of the parameters. If an upper bound of u_i is returned for a parameter p_i , then the SEVSELECT procedure assumes that $p_i < u_i$ (strict inequality). If a missing value is returned for some parameter, then the SEVSELECT procedure assumes that there is no upper bound for that parameter (equivalent to an upper bound of ∞).

- *Type:* Subroutine
- *Required:* NO
- *Number of arguments:* m , where m is the number of distribution parameters
- *Sequence and type of arguments:*

p_1	Output argument that returns the upper bound on the first parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
p_2	Output argument that returns the upper bound on the second parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
$\vdots$	
p_m	Output argument that returns the upper bound on the m th parameter. You must specify this in the OUTARGS statement inside the subroutine's definition.
- *Return value:* The results, upper bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```

subroutine FOO_UPPERBOUNDS(p1, p2);
  outargs p1, p2;

  p1 = <upper bound for P1>;
  p2 = <upper bound for P2>;
endsub;

```

*dist_core*GRADIENT

defines a subroutine that returns the gradient vector of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the gradient should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
	⋮
pm	Numeric value of the m th parameter
grad{*}	Output numeric array of size m that contains the gradient vector evaluated at the specified values. If h denotes the value of the <i>core</i> function, then the expected order of the values in the array is as follows: $\frac{\partial h}{\partial p_1} \frac{\partial h}{\partial p_2} \dots \frac{\partial h}{\partial p_m}$

- *Return value*: Numeric array that contains the gradient evaluated at x for the parameter values $(p_1, p_2, \dots, p_m)$

Here is a sample structure of the function for the core function CDF of a distribution named 'FOO':

```

subroutine FOO_CDFGRADIENT(x, P1, P2, grad{*});
  outargs grad;

  /* Code to compute gradient by using x, P1, and P2 */
  grad[1] = <partial derivative of CDF w.r.t. P1
             evaluated at x, P1, P2>;
  grad[2] = <partial derivative of CDF w.r.t. P2
             evaluated at x, P1, P2>;
endsub;

```

dist_coreHESSIAN

defines a subroutine that returns the Hessian matrix of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

<i>x</i>	Numeric value of the random variable at which the Hessian matrix should be evaluated
<i>p1</i>	Numeric value of the first parameter
<i>p2</i>	Numeric value of the second parameter
$\vdots$	
<i>pm</i>	Numeric value of the m th parameter
<i>hess{*}</i>	Output numeric array of size $m(m + 1)/2$ that contains the lower triangular portion of the Hessian matrix in a packed vector form, evaluated at the specified values. If h denotes the value of the <i>core</i> function, then the expected order of the values in the array is as follows: $\frac{\partial^2 h}{\partial p_1^2} \mid \frac{\partial^2 h}{\partial p_1 \partial p_2} \frac{\partial^2 h}{\partial p_2^2} \mid \dots \mid \frac{\partial^2 h}{\partial p_1 \partial p_m} \frac{\partial^2 h}{\partial p_2 \partial p_m} \dots \frac{\partial^2 h}{\partial p_m^2}$

- *Return value*: Numeric array that contains the lower triangular portion of the Hessian matrix evaluated at x for the parameter values $(p_1, p_2, \dots, p_m)$

Here is a sample structure of the subroutine for the core function LOGSDF of a distribution named 'FOO':

```
subroutine FOO_LOGSDFHESSIAN(x, P1, P2, hess{*});
    outargs hess;

    /* Code to compute Hessian by using x, P1, and P2 */
    hess[1] = <second order partial derivative of LOGSDF
              w.r.t. P1 evaluated at x, P1, P2>;
    hess[2] = <second order partial derivative of LOGSDF
              w.r.t. P1 and P2 evaluated at x, P1, P2>;
    hess[3] = <second order partial derivative of LOGSDF
              w.r.t. P2 evaluated at x, P1, P2>;
endsub;
```

Predefined Utility Functions

The following predefined utility functions are provided with the SEVSELECT procedure and are available in the Sashelp.SvrtDIST library:

SVRTUTIL_EDF

This function computes the empirical distribution function (EDF) estimate at the specified value of the random variable given the EDF estimate for a sample.

- *Type:* Function
- *Signature:* SVRTUTIL_EDF(y, n, x{*}, F{*}, Ftype)
- *Argument description:*

y	Value of the random variable at which the EDF estimate is desired
n	Dimension of the x and F input arrays
x{*}	Input numeric array of dimension n that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension n in which each $F[i]$ contains the EDF estimate for $x[i]$. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the x and F arrays. For definitions of types, see the section “ Supplying EDF Estimates to Functions and Subroutines ” on page 889.
- *Return value:* The EDF estimate at y

The type of the sample EDF estimate determines how the EDF estimate at y is computed. For more information, see the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 889.

SVRTUTIL_EMPLIMMOMENT

This function computes the empirical estimate of the limited moment of specified order for the specified upper limit, given the EDF estimate for a sample.

- *Type:* Function
- *Signature:* SVRTUTIL_EMPLIMMOMENT(k, u, n, x{*}, F{*}, Ftype)
- *Argument description:*

k	Order of the desired empirical limited moment
u	Upper limit on the value of the random variable to be used in the computation of the desired empirical limited moment
n	Dimension of the x and F input arrays
x{*}	Input numeric array of dimension n that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension n in which each $F[i]$ contains the EDF estimate for $x[i]$. These values must be sorted in nondecreasing order.

Ftype Type of the empirical estimate that is stored in the x and F arrays. For definitions of types, see the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 889.

- *Return value:* The desired empirical limited moment

The empirical limited moment is computed by using the empirical estimate of the CDF. If $F_n(x)$ denotes the EDF at x , then the empirical limited moment of order k with upper limit u is defined as

$$E_n[(X \wedge u)^k] = k \int_0^u (1 - F_n(x))x^{k-1} dx$$

The SVRTUTIL_EMPLIMMOMENT function uses the piecewise linear nature of $F_n(x)$ as described in the section “[Supplying EDF Estimates to Functions and Subroutines](#)” on page 889 to compute the integration.

SVRTUTIL_HILLCUTOFF

This function computes an estimate of the value where the right tail of a distribution is expected to begin. The function implements the algorithm described in Danielsson et al. 2001. The description of the algorithm uses the following notation:

n	Number of observations in the original sample
B	Number of bootstrap samples to draw
m_1	Size of the bootstrap sample in the first step of the algorithm ($m_1 < n$)
$x_{(i)}^{j,m}$	i th-order statistic of j th bootstrap sample of size m ($1 \leq i \leq m, 1 \leq j \leq B$)
$x_{(i)}$	i th-order statistic of the original sample ($1 \leq i \leq n$)

Given the input sample x and values of B and m_1 , the steps of the algorithm are as follows:

1. Take B bootstrap samples of size m_1 from the original sample.
2. Find the integer k_1 that minimizes the bootstrap estimate of the mean squared error:

$$k_1 = \arg \min_{1 \leq k < m_1} Q(m_1, k)$$

3. Take B bootstrap samples of size $m_2 = m_1^2/n$ from the original sample.
4. Find the integer k_2 that minimizes the bootstrap estimate of the mean squared error:

$$k_2 = \arg \min_{1 \leq k < m_2} Q(m_2, k)$$

5. Compute the integer k_{opt} , which is used for computing the cutoff point:

$$k_{\text{opt}} = \frac{k_1^2}{k_2} \left(\frac{\log(k_1)}{2 \log(m_1) - \log(k_1)} \right)^{2 - 2 \log(k_1) / \log(m_1)}$$

6. Set the cutoff point equal to $x_{(k_{\text{opt}}+1)}$.

The bootstrap estimate of the mean squared error is computed as

$$Q(m, k) = \frac{1}{B} \sum_{j=1}^B \text{MSE}_j(m, k)$$

The mean squared error of j th bootstrap sample is computed as

$$\text{MSE}_j(m, k) = (M_j(m, k) - 2(\gamma_j(m, k))^2)^2$$

where $M_j(m, k)$ is a control variate proposed by Danielsson et al. 2001,

$$M_j(m, k) = \frac{1}{k} \sum_{i=1}^k \left(\log(x_{(m-i+1)}^{j,m}) - \log(x_{(m-k)}^{j,m}) \right)^2$$

and $\gamma_j(m, k)$ is the Hill's estimator of the tail index (Hill 1975),

$$\gamma_j(m, k) = \frac{1}{k} \sum_{i=1}^k \log(x_{(m-i+1)}^{j,m}) - \log(x_{(m-k)}^{j,m})$$

This algorithm has two tuning parameters, B and m_1 . The number of bootstrap samples B is chosen based on the availability of computational resources. The optimal value of m_1 is chosen such that the following ratio, $R(m_1)$, is minimized:

$$R(m_1) = \frac{(Q(m_1, k_1))^2}{Q(m_2, k_2)}$$

The SVRTUTIL_HILLCUTOFF utility function implements the preceding algorithm. It uses the grid search method to compute the optimal value of m_1 .

- *Type:* Function
- *Signature:* SVRTUTIL_HILLCUTOFF(n , $x\{*\}$, b , s , $status$)
- *Argument description:*

n	Dimension of the array x
$x\{*\}$	Input numeric array of dimension n that contains the sample
b	Number of bootstrap samples used to estimate the mean squared error. If b is less than 10, then a default value of 50 is used.
s	Approximate number of steps used to search the optimal value of m_1 in the range $[n^{0.75}, n - 1]$. If s is less than or equal to 1, then a default value of 10 is used.
$status$	Output argument that contains the status of the algorithm. If the algorithm succeeds in computing a valid cutoff point, then <i>status</i> is set to 0. If the algorithm fails, then <i>status</i> is set to 1.
- *Return value:* The cutoff value where the right tail is estimated to start. If the size of the input sample is inadequate ($n \leq 5$), then a missing value is returned and *status* is set to a missing value. If the algorithm fails to estimate a valid cutoff value (*status* = 1), then the fifth-largest value in the input sample is returned.

SVRTUTIL_PERCENTILE

This function computes the specified empirical percentile given the EDF estimates.

- *Type:* Function
- *Signature:* SVRTUTIL_PERCENTILE(p, n, x{*}, F{*}, Ftype)
- *Argument description:*

p	Desired percentile. The value must be in the interval (0,1). The function returns the 100p th percentile.
n	Dimension of the <i>x</i> and <i>F</i> input arrays
x{*}	Input numeric array of dimension <i>n</i> that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension <i>n</i> in which each F[<i>i</i>] contains the EDF estimate for x[<i>i</i>]. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the <i>x</i> and <i>F</i> arrays. For definitions of types, see the section “ Supplying EDF Estimates to Functions and Subroutines ” on page 889.

- *Return value:* The 100pth percentile of the input sample

The method used to compute the percentile depends on the type of the EDF estimate (Ftype argument).

Ftype = 1 Smoothed empirical estimates are computed using the method described in Klugman, Panjer, and Willmot (1998). Let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x . Define $g = \lfloor p(n+1) \rfloor$ and $h = p(n+1) - g$. Then the empirical percentile $\hat{\pi}_p$ is defined as

$$\hat{\pi}_p = (1-h)x[g] + hx[g+1]$$

This method does not work if $p < 1/(n+1)$ or $p > n/(n+1)$. If $p < 1/(n+1)$, then the function returns $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $p > n/(n+1)$, then $\hat{\pi}_p = x[n]$.

Ftype = 2 If $p < F[1]$, then $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $|p - F[i]| < \epsilon$ for some value of i and $i < n$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = \frac{x[i] + x[i+1]}{2}$$

where ϵ is a machine-precision constant as returned by the SAS function CONSTANT(‘MACEPS’). If $F[i-1] < p < F[i]$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = x[i]$$

If $p \geq F[n]$, then $\hat{\pi}_p = x[n]$.

Ftype = 3 If $p < F[1]$, then $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $|p - F[i]| < \epsilon$ for some value of i and $i < n$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = \frac{x[i] + x[i+1]}{2}$$

where ϵ is a machine-precision constant as returned by the SAS function `CONSTANT('MACEPS')`. If $F[i - 1] < p < F[i]$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = x[i - 1] + (p - F[i - 1]) \frac{x[i] - x[i - 1]}{F[i] - F[i - 1]}$$

If $p \geq F[n]$, then $\hat{\pi}_p = x[n]$.

SVRTUTIL_RAWMOMENTS

This subroutine computes the raw moments of a sample.

- *Type:* Subroutine
- *Signature:* `SVRTUTIL_RAWMOMENTS(n, x{*}, nx{*}, nRaw, raw{*})`
- *Argument description:*

<code>n</code>	Dimension of the <code>x</code> and <code>nx</code> input arrays
<code>x{*}</code>	Input numeric array of dimension <code>n</code> that contains distinct values of the random variable that are observed in the sample
<code>nx{*}</code>	Input numeric array of dimension <code>n</code> in which each <code>nx[i]</code> contains the number of observations in the sample that have the value <code>x[i]</code>
<code>nRaw</code>	Desired number of raw moments. The output array <code>raw</code> contains the first <code>nRaw</code> raw moments.
<code>raw{*}</code>	Output array of raw moments. The <code>k</code> th element in the array (<code>raw{k}</code>) contains the <code>k</code> th raw moment, where $1 \leq k \leq nRaw$.
- *Return value:* Numeric array `raw` that contains the first `nRaw` raw moments. The array contains missing values if the sample has no observations (that is, if all the values in the `nx` array add up to zero).

SVRTUTIL_SORT

This function sorts an array of numeric values in an ascending or descending order.

- *Type:* Subroutine
- *Signature:* `SVRTUTIL_SORT(n, x{*}, flag)`
- *Argument description:*

<code>n</code>	Dimension of the input array <code>x</code>
<code>x{*}</code>	Numeric array that contains the values to be sorted at input. The subroutine uses the same array to return the sorted values.
<code>flag</code>	A numeric value that controls the sort order. If <code>flag</code> is 0, then the values are sorted in an ascending order. If <code>flag</code> has any value other than 0, then the values are sorted in descending order.
- *Return value:* Numeric array `x`, which is sorted in place (that is, the sorted array is stored in the same storage area occupied by the input array `x`)

You can use the following predefined functions when you use the FCMP procedure to define functions and subroutines. They are summarized here for your information. For more information, see the FCMP procedure documentation in *Base SAS Procedures Guide*.

INVCDF

This function computes the quantile from any continuous probability distribution by numerically inverting the CDF of that distribution. You need to specify the CDF function of the distribution, the values of its parameters, and the cumulative probability to compute the quantile.

LIMMOMENT

This function computes the limited moment of order k with upper limit u for any continuous probability distribution. The limited moment is defined as

$$\begin{aligned} E[(X \wedge u)^k] &= \int_0^u x^k f(x) dx + \int_u^\infty u^k f(x) dx \\ &= \int_0^u x^k f(x) dx + u^k (1 - F(u)) \end{aligned}$$

where $f(x)$ and $F(x)$ denote the PDF and the CDF of the distribution, respectively. The LIMMOMENT function uses the following alternate definition, which can be derived using integration-by-parts:

$$E[(X \wedge u)^k] = k \int_0^u (1 - F(x)) x^{k-1} dx$$

You need to specify the CDF function of the distribution, the values of its parameters, and the values of k and u to compute the limited moment.

Scoring Functions

Scoring refers to the act of evaluating a distribution function, such as LOGPDF, SDF, or QUANTILE, on an observation by using the fitted parameter estimates of that distribution. You can do scoring in a DATA step by using the **OUTEST=** data table that you create using PROC SEVSELECT. However, that approach requires some cumbersome programming. To simplify the scoring process, you can use PROC SEVSELECT to create scoring functions for each fitted distribution.

As an example, assume that you fit the Pareto distribution by using PROC SEVSELECT and that it converges. Further assume that you want to use the fitted distribution to evaluate the probability of observing a loss value greater than some set of regulatory limits $\{L\}$ that are encoded in a data table. You can simplify this scoring process as follows. First, in the PROC SEVSELECT step that fits your distributions, you create the scoring functions library by specifying the **OUTSCORELIB** statement as illustrated in the following steps:

```
proc sevselect data=mycas.input;
  loss lossclaim;
  dist pareto;
  outscorelib outlib=sasuser.fitdist;
run;
```

Upon successful completion, if the Pareto distribution model has converged, then the library Sasuser.Fitdist contains the *SEV_SDF* scoring function in addition to other scoring functions, such as *SEV_PDF*, *SEV_LOGPDF*, and so on. Further, PROC SEVSELECT also sets the CMPLIB system option to include the Sasuser.Fitdist library. If the set of limits $\{L\}$ is recorded in the variable Limit in the scoring data set Work.Limits, then you can submit the following DATA step to compute the probability of seeing a loss greater than each limit:

```

data prob;
  set work.limits;
  exceedance_probability = sev_sdf(limit);
run;

```

Without using scoring functions, you can still perform this scoring task, but the DATA step that you need to accomplish it becomes more complicated and less flexible. For example, you would need to read the parameter estimates from some output created by PROC SEVSELECT. To do that, you would need to know the parameter names, which are different for different distributions; this in turn would require you to write a specific DATA step for each distribution or to write a SAS macro. Scoring functions enable you to accomplish that task much more easily.

If you fit multiple distributions, then you can specify the COMMONPACKAGE option in the OUTSCORELIB statement as follows:

```

proc sevselect data=mycas.input;
  loss lossclaim;
  dist exp pareto weibull;
  outscorelib outlib=sasuser.fitdist commonpackage;
run;

```

The preceding step creates scoring functions such as *SEV_SDF_Exp*, *SEV_SDF_Pareto*, and *SEV_SDF_Weibull*. You can use them to compare the probabilities of exceeding the limit for different distributions by using the following DATA step:

```

data prob;
  set work.limits;
  exceedance_exp = sev_sdf_exp(limit);
  exceedance_pareto = sev_sdf_pareto(limit);
  exceedance_weibull = sev_sdf_weibull(limit);
run;

```

Formal Description

PROC SEVSELECT creates a scoring function for each distribution function. A distribution function is defined as any function named *dist_suffix*, where *dist* is the name of a distribution that you specify in the DIST statement and the function's signature is identical to the signature of the required distribution function, such as *dist_CDF* or *dist_LOGCDF*. For example, for the function 'FOO_BAR' to be a distribution function, you must specify the distribution 'FOO' in the DIST statement, and you must define 'FOO_BAR' in the following manner if the distribution 'FOO' has parameters named 'P1' and 'P2':

```

function FOO_BAR(y, P1, P2);
  /* Code to compute BAR by using y, P1, and P2 */
  R = <computed BAR>;
  return (R);
endsub;

```

For more information about the signature that defines a distribution function, see the description of the *dist_CDF* function in the section “[Defining a Severity Distribution Model with the FCMP Procedure](#)” on page 895.

The name and package of the scoring function of a distribution function depend on whether you specify the COMMONPACKAGE option in the OUTSCORELIB statement.

When you do not specify the `COMMONPACKAGE` option, the scoring function that corresponds to the distribution function *dist_suffix* is named *SEV_suffix*, where *SEV_* is the standard prefix of all scoring functions. The scoring function is created in a package named *dist*. Each scoring function accepts only one argument, the value of the loss variable, and returns the same value as the value that is returned by the corresponding distribution function for the final estimates of the distribution's parameters. For example, for the preceding 'FOO_BAR' distribution function, the scoring function named 'SEV_BAR' is created in the package named 'FOO' and 'SEV_BAR' has the following signature:

```
function SEV_BAR(y);
  /* returns value of FOO_BAR for the supplied value
    of y and fitted values of P1, P2 */
endsub;
```

If you specify the `COMMONPACKAGE` option in the `OUTSCORELIB` statement, then the scoring function that corresponds to the distribution function *dist_suffix* is named *SEV_suffix_dist*, where *SEV_* is the standard prefix of all scoring functions. The scoring function is created in a package named *sevfit*. For example, for the preceding 'FOO_BAR' distribution function, if you specify the `COMMONPACKAGE` option, the scoring function named 'SEV_BAR_FOO' is created in the *sevfit* package and 'SEV_BAR_FOO' has the following signature:

```
function SEV_BAR_FOO(y);
  /* returns value of FOO_BAR for the supplied value
    of y and fitted values of P1, P2 */
endsub;
```

Scoring Functions for the Scale Regression Model

If you use the `SCALEMODEL` statement to specify a scale regression model, then `PROC SEVSELECT` generates the scoring functions when you specify only singleton continuous effects. If you specify interaction or classification effects, then scoring functions are not generated.

For a scale regression model, the estimate of the scale parameter or the log-transformed scale parameter of the distribution depends on the values of the regressors. So `PROC SEVSELECT` creates a scoring function that has the following signature, where *x{*}* represents the array of regressors:

```
function SEV_BAR(y, x{*});
  /* returns value of FOO_BAR for the supplied value of x and fitted values of P1, P2 */
endsub;
```

As an illustration of using this form, assume that you submit the following `PROC SEVSELECT` step to create the scoring library `Sasuser.Scalescore`:

```
proc sevselect data=mycas.input;
  loss lossclaim;
  scalemodel x1-x3;
  dist pareto;
  outscorelib outlib=sasuser.scalescore;
run;
```

Your scoring data set must contain all the regressors that you specify in the `SCALEMODEL` statement. You can submit the following `DATA` step to score observations by using the scale regression model:

```
data prob;
  array regvals{*} x1-x3;
  set work.limits;
  exceedance_probability = sev_sdf(limit, regvals);
run;
```

PROC SEVSELECT creates two utility functions—SEV_NUMREG and SEV_REGNAME—in the OUTLIB= library, which return the number of regressors and name of a given regressor, respectively. They are described in detail in the next section. These utility functions are useful when you do not have easy access to the regressor names in the SCALEMODEL statement. You can use the utility functions as follows:

```
data prob;
  array regvals{10} _temporary_;
  set work.limits;
  do i = 1 to sev_numreg();
    regvals(i) = input(vvaluex(sev_regname(i)), best12.);
  end;
  exceedance_probability = sev_sdf(limit, regvals);
run;
```

The dimension of the regressor values array that you supply to the scoring function must be equal to $K + L$, where K is the number of regressors that you specify in the SCALEMODEL statement irrespective of whether PROC SEVSELECT deems any of those regressors to be redundant. L is 1 if you specify an OFFSET= variable in the SCALEMODEL statement, and 0 otherwise.

Utility Functions and Subroutines in the OUTLIB= Library

In addition to creating the scoring functions for all distribution functions, PROC SEVSELECT creates the following utility functions and subroutines in the OUTLIB= library.

SEV_NUMPARM | SEV_NUMPARM_dist

is a function that returns the number of distribution parameters and has the following signature:

- *Type:* Function
- *Number of arguments:* 0
- *Sequence and type of arguments:* Not applicable
- *Return value:* Numeric value that contains the number of distribution parameters

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a function named SEV_NUMPARM is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_NUMPARM();
  n = <number of distribution parameters>;
  return (n);
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the function named SEV_NUMPARM_dist is created in the *sevfit* package. SEV_NUMPARM_dist has the same structure as the SEV_NUMPARM function that is described previously.

SEV_PARMEST | SEV_PARMEST_dist

is a subroutine that returns the estimate and standard error of a specified distribution parameter and has the following signature:

- *Type:* Subroutine
- *Number of arguments:* 3
- *Sequence and type of arguments:*
 - index* specifies the numeric value of the index of the distribution parameter for which you want the information. The value of *index* must be in the interval $[1, m]$, where m is the number of parameters in the distribution to which this subroutine belongs.
 - est* specifies the output argument that returns the estimate of the requested parameter.
 - stderr* specifies the output argument that returns the standard error of the requested parameter.
- *Return value:* Estimate and standard error of the requested distribution parameter that are returned in the output arguments *est* and *stderr*, respectively

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a subroutine named SEV_PARMEST is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the subroutine:

```
subroutine SEV_PARMEST(index, est, stderr);
  outargs est, stderr;
  est = <value of the estimate for the distribution parameter
        at position 'index'>;
  stderr = <value of the standard error for distribution parameter
           at position 'index'>;
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the subroutine named SEV_PARMEST_dist is created in the *sevfit* package. SEV_PARMEST_dist has the same structure as the SEV_PARMEST subroutine that is described previously.

If you use the SCALEMODEL statement to specify a scale regression model, and if you specify only singleton continuous effects, then for *index*=1, the returned estimates are of θ_0 (the base value of the scale parameter) or $\log(\theta_0)$ if the distribution has a log-scale parameter. For more information about θ_0 , see the section “[Estimating Regression Effects](#)” on page 878.

SEV_PARMNAME | SEV_PARMNAME_dist

is a function that returns the name of a specified distribution parameter and has the following signature:

- *Type:* Function
- *Number of arguments:* 1
- *Sequence and type of arguments:*
 - index* specifies the numeric value of the index of the distribution parameter for which you want the information. The value of *index* must be in the interval $[1, m]$, where m is the number of parameters in the distribution to which this function belongs.

- *Return value:* Character value that contains the name of the distribution parameter that appears at the position *index* in the distribution's definition

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a function named SEV_PARMNAME is created in the package of each distribution.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_PARMNAME(index) $32;
    name = <name of the distribution parameter at position 'index'>;
    return (name);
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, a function named SEV_PARMNAME_*dist* is created in the *sevfit* package. SEV_PARMNAME_*dist* has the same structure as the SEV_PARMNAME function that is described previously.

If you use the SCALEMODEL statement to specify a scale regression model, and if you specify only singleton continuous effects, then the following helper functions and subroutines are also created in the OUTLIB= library.

SEV_NUMREG

is a function that returns the number of regressors and has the following signature:

- *Type:* Function
- *Number of arguments:* 0
- *Sequence and type of arguments:* Not applicable
- *Return value:* Numeric value that contains the number of regressors that you specify in the SCALEMODEL statement. If you specify an OFFSET= option variable in the SCALEMODEL statement, then the returned value is equal to 1 plus the number of regressors that you specify in the SCALEMODEL statement.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_NUMREG();
    m = <number of regressors>;
    if (<offset variable is specified>) then m = m + 1;
    return (m);
endsub;
```

This function does not depend on any distribution, so it is always created in the *sevfit* package.

SEV_REGEST | SEV_REGEST_*dist*

is a subroutine that returns the estimate and standard error of a specified regression parameter and has the following signature:

- *Type:* Subroutine
- *Number of arguments:* 3
- *Sequence and type of arguments:*

index specifies the numeric value of the index of the regression parameter for which you want the information. The value of *index* must be in the interval $[1, K]$, where K is the number of regressors that is returned by the SEV_NUMREG function. If you specify an OFFSET= option variable in the SCALEMODEL statement, then an *index* value of K corresponds to the offset variable.

est specifies the output argument that returns the estimate of the requested regression parameter.

stderr specifies the output argument that returns the standard error of the requested regression parameter.

- *Return value:* Estimate and standard error of the requested regression parameter that are returned in the output arguments *est* and *stderr*, respectively

If you do not specify the COMMONPACKAGE option in the OUTSCORELIB statement, then a subroutine named SEV_REGEST is created in the package of each distribution. Here is a sample structure of the code that PROC SEVSELECT uses to define the subroutine:

```
subroutine SEV_REGEST(index, est, stderr);
  outargs est, stderr;
  est = <value of the estimate for the regression parameter
        at position 'index'>;
  stderr = <value of the standard error for regression parameter
           at position 'index'>;
endsub;
```

If you specify the COMMONPACKAGE option in the OUTSCORELIB statement, then for each distribution *dist*, the subroutine named SEV_REGEST_*dist* is created in the *sevfit* package. SEV_REGEST_*dist* has the same structure as the SEV_REGEST subroutine that is described previously.

If the regressor that corresponds to the specified *index* value is a redundant regressor, the returned values of both *est* and *stderr* are equal to the special missing value of .R. If you specify an OFFSET= option variable in the SCALEMODEL statement and if the *index* value corresponds to the offset variable—that is, it is equal to the value that the SEV_NUMREG function returns—then the returned value of *est* is equal to 1 and the returned value of *stderr* is equal to the special missing value of .F.

SEV_REGNAME

is a function that returns the name of a specified regressor and has the following signature:

- *Type:* Function
- *Number of arguments:* 1
- *Sequence and type of arguments:*

index specifies the numeric value of the index of the regressor for which you want the name. The value of *index* must be in the interval $[1, K]$, where K is the number of regressors that is returned by the SEV_NUMREG function. If you specify an OFFSET= option variable in the SCALEMODEL statement, then an *index* value of K corresponds to the offset variable.

- *Return value:* Character value that contains the name of the regressor that appears at the position *index* in the SCALEMODEL statement. If you specify an OFFSET= option variable in the SCALEMODEL statement, then for an *index* value of *K*, the returned value contains the name of the offset variable.

Here is a sample structure of the code that PROC SEVSELECT uses to define the function:

```
function SEV_REGNAME(index) $32;
    name = <name of regressor at position 'index'>;
    return (name);
endsub;
```

This function does not depend on any distribution, so it is always created in the *sevfit* package.

Custom Objective Functions

You can use a series of programming statements that use variables in the DATA= data table to assign a value to an objective function symbol. You must specify the objective function symbol by using the **OBJECTIVE=** option in the PROC SEVSELECT statement.

The objective function can be programmed such that it applies to any distribution that is used in the model. For that purpose, PROC SEVSELECT recognizes the following *keyword* functions in the programming statements:

<code>_PDF_(x)</code>	evaluates the probability density function (PDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_CDF_(x)</code>	evaluates the cumulative distribution function (CDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_SDF_(x)</code>	evaluates the survival distribution function (SDF) of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGPDF_(x)</code>	evaluates the natural logarithm of the PDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGCDF_(x)</code>	evaluates the natural logarithm of the CDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_LOGSDF_(x)</code>	evaluates the natural logarithm of the SDF of a distribution at the current value of a data table variable <i>x</i> .
<code>_EDF_(x)</code>	evaluates the empirical distribution function (EDF) estimate at the current value of a data table variable <i>x</i> . Internally, PROC SEVSELECT computes the estimate by using the SVRTUTIL_EDF function as described in the section “ Predefined Utility Functions ” on page 908. The EDF estimate that the SVRTUTIL_EDF function requires is computed by using a subset of observations in the current BY group or a subset of the input data table if you do not specify the BY statement. The subset of observations is randomly chosen on each worker node by using the values that you specify in the INITSAMPLE option.

EMPLIMMOMENT(k, u)

evaluates the empirical limited moment of order k at the current value of a data table variable u that represents the upper limit of the limited moment. The order k can also be a data table variable. Internally, PROC SEVSELECT computes the moment by using the SVRTUTIL_EMPLIMMOMENT function as described in the section “[Predefined Utility Functions](#)” on page 908. The EDF estimate that the SVRTUTIL_EMPLIMMOMENT function requires is computed by using a subset of observations in the current BY group or a subset of the input data table if you do not specify the BY statement. The subset of observations is randomly chosen on each worker node by using the values that you specify in the [INITSAMPLE](#) option.

LIMMOMENT(k, u)

evaluates the limited moment of order k at the current value of a data table variable u that represents the upper limit of the limited moment. The order k can be a data table variable or a constant. Internally, for each candidate distribution, PROC SEVSELECT computes the moment by using the LIMMOMENT function as described in the section “[Predefined Utility Functions](#)” on page 908.

All the preceding functions are right-hand-side functions. They act as placeholders for distribution-specific functions, with the exception of `_EDF_` and `_EMPLIMMOMENT_` functions.

As an example, suppose that the data table `mycas.test` contains a response variable Y and a left-truncation threshold variable T . The following statements use the values in this data table to fit a model with distribution D such that the parameters of the model minimize the value of the objective function symbol `MYOBJ`:

```
options cmplib=(work.mydist);
proc sevselect data=mycas.test objective=myobj;
  loss y / lt=t;

  myobj = -_LOGPDF_(y);
  if (not(missing(t))) then
    myobj = myobj + log(1-_CDF_(t));

  dist d;
run;
```

You designate the symbol `MYOBJ` as an objective function symbol by using the [OBJECTIVE=](#) option in the PROC SEVSELECT statement. The response variable Y and left-truncation variable T are specified in the LOSS statement. The distribution D is specified in the DIST statement. The remaining statements constitute a program that computes the value of the `MYOBJ` symbol.

Let the distribution D have the parameters $P1$ and $P2$. To estimate the model for this distribution, PROC SEVSELECT internally converts the generic program to the following program specific to distribution D :

```
myobj = -D_LOGPDF(y, p1, p2);
if (not(missing(t))) then
  myobj = myobj + log(1-D_CDF(t, p1, p2));
```

Note that the generic keyword functions `_LOGPDF_` and `_CDF_` have been replaced by the distribution-specific functions `D_LOGPDF` and `D_CDF`, respectively, and the argument list of each is expanded to include the distribution parameters. You must have defined the `D_LOGPDF` and `D_CDF` functions previously; they are assumed to be available in the library `Work.Mydist` that you specify in the `CMPLIB=` option.

The program is executed for each observation in the data table `mycas.Test` to compute the value of `MYOBJ` by using the values of variables `Y` and `T` in that observation and internally computed values of the model parameters `P1` and `P2`. The values of `MYOBJ` are then added over all the observations of the data table or over all the observations of the current `BY` group if you specify the `BY` statement. The resulting aggregate value is the value of the objective function, and it is supplied to the optimizer. If the optimizer requires derivatives of the objective function, then `PROC SEVSELECT` automatically differentiates `MYOBJ` with respect to the parameters `P1` and `P2`. The optimizer iterates through various combinations of the values of `P1` and `P2`; in each iteration, it computes a new value of the objective function and the needed derivatives of it, until it finds a combination that minimizes the objective function.

Note the following points when you define your own program to compute the custom objective function:

- The value of the objective function is always minimized by `PROC SEVSELECT`. If you want to maximize the value of a certain objective, then add a statement that assigns the negated value of the maximization objective to the objective function symbol that you specify in the `OBJECTIVE=` option. Minimization of the negated objective is equivalent to maximization of the original objective.
- The contributions of individual observations are always added together to compute the overall objective function in a given iteration of the optimizer. If you specify the `WEIGHT` statement, then the contribution of each observation is weighted by multiplying it by the normalized value of the weight variable for that observation.
- If you are fitting multiple distributions in one `PROC SEVSELECT` step and you use any of the keyword functions in your program, then it is recommended that you not explicitly use the parameters of any of the specified distributions in your programming statements.
- If you use a specific keyword function in your programming statements, then the corresponding distribution functions must be defined in a library that you specify in the `CMPLIB=` system option or in `Sashelp.Svrtlist`, the predefined functions library. In the preceding example, it is assumed that the functions `D_LOGPDF` and `D_CDF` are defined in the library `Work.Mydist`, which you specify in the `CMPLIB=` option.
- You can use most `DATA` step statements and functions in your program. The `DATA` step file and the data set I/O statements (for example, `INPUT`, `FILE`, `SET`, and `MERGE`) are not available. However, some `PUT` statement functionality is supported. For more information, see the section “`PROC FCMP` and `DATA` Step Differences” in *Base SAS Procedures Guide*. In addition to the differences that are listed in that section, the following differences exist:
 - You can use only numeric-valued variables in `PROC SEVSELECT` programming statements. This restriction also implies that you cannot use SAS functions or call routines that require character-valued arguments, unless you pass those arguments as constant (literal) strings or characters.
 - You cannot use functions that create lagged versions of a variable in `PROC SEVSELECT` programming statements. If you need lagged versions, then you can use a `DATA` step before the `PROC SEVSELECT` step to add those versions to the input data table.
- When you are writing your programming statements, avoid defining variables that begin with an underscore (`_`), because they might conflict with internal variables that `PROC SEVSELECT` creates.

Custom Objective Functions and Regression Effects

If you specify regression effects by using the SCALEMODEL statement, then PROC SEVSELECT automatically adds a statement before your programming statements to compute the value of the scale parameter or the log-transformed scale parameter of the distribution by using the values of the regression variables and internally created regression parameters. For example, if your specification of the SCALEMODEL statement results in three regression effects, x_1 , x_2 , and x_3 , then for a model that contains the distribution D with scale parameter S, PROC SEVSELECT adds to the beginning of your program a statement that is equivalent to the following statement:

$$S = _SEVTHETA0 * \exp(_SEVBETA1 * x_1 + _SEVBETA2 * x_2 + _SEVBETA3 * x_3);$$

If a model contains a distribution D1 with a log-transformed scale parameter M, PROC SEVSELECT adds to the beginning of your program a statement that is equivalent to the following statement:

$$M = _SEVTHETA0 + _SEVBETA1 * x_1 + _SEVBETA2 * x_2 + _SEVBETA3 * x_3;$$

The internal symbols $_SEVTHETA0$, $_SEVBETA1$, $_SEVBETA2$, and $_SEVBETA3$ represent the regression parameters that are associated with the intercept and the regression effects x_1 , x_2 , and x_3 , respectively.

Because the names of the internal regression parameter symbols start with the prefix $_SEV$, if you use a variable in your program whose name begins with $_SEV$, then PROC SEVSELECT writes an error message to the SAS log and stops processing.

Input Data Tables

PROC SEVSELECT accepts DATA= and INEST= data tables as input data tables. This section details the information that they are expected to contain.

DATA= Data Table

The DATA= data table is expected to contain the values of the analysis variables that you specify in the CLASS, EFFECT, LOSS, and SCALEMODEL statements.

If you specify the BY statement, then the DATA= data table must contain all the BY variables that you specify in the BY statement. However, it is not required to be sorted by the BY variables.

INEST= Data Table

The INEST= data table is expected to contain the initial values of the parameters for the parameter estimation process.

If you specify the BY statement, then the INEST= data table must contain all the BY variables that you specify in the BY statement. The data table is not required to be sorted by BY variables, and it is not required to contain all the BY groups present in the DATA= data table. For the BY groups that are not present in the INEST= data table, the default parameter initialization method is used.

The INEST= data table must be created by using the OUTEST= option in a PROC SEVSELECT step that uses the same BY and SCALEMODEL statements as the PROC SEVSELECT step where you specify the INEST= option.

Output Data Tables

PROC SEVSELECT produces the OUTEST=, OUTMODELINFO=, and OUTSTAT= data tables when requested by their respective options in the PROC SEVSELECT statement. It also produces the OUT= data table when you specify the OUTPUT statement. The data tables and their contents are described in the following sections.

OUT= Data Table

The OUT= data table that you specify in the OUTPUT statement records the estimates of the scoring functions and quantiles that you specify in the OUTPUT statement.

For each distribution that you specify in the DIST statement, the OUT= data table contains one variable for each scoring function that you specify in the FUNCTIONS= option and one variable for each quantile that you specify in the QUANTILES= option. The prefix of the variable's name is <distribution-name>_, whereas the suffix of the variable's name is determined by the information that you specify in the respective option or by the default method that PROC SEVSELECT uses. For more information about variable names, see the description of the OUTPUT statement.

The OUT= data table also contains the variables that you specify in the COPYVARS= option. If you specify the BY statement and if you want PROC SEVSELECT to copy the BY variables from the DATA= data table to the OUT= data table, then you must specify them in the COPYVARS= option.

The number of observations in the OUT= data table depends on the options that you specify in the OUTPUT statement and whether or not you specify the SCALEMODEL statement.

If either of the following conditions is met, then the number of observations in the OUT= data table is equal to the number of observations in the DATA= data table:

- You specify the SCALEMODEL statement.
- You specify the FUNCTIONS= option in the OUTPUT statement such that at least one scoring function does not have a constant, nonmissing argument.

If neither of the preceding conditions is met, then the number of observations in the OUT= data table is equal to the number of BY groups, which is equal to 1 if you do not specify the BY statement.

OUTEST= Data Table

The OUTEST= data table records the estimates of the model parameters. It also contains estimates of their standard errors and optionally their covariance structure. If you specify BY variables, then the data are organized in BY groups and the data table contains variables that you specify in the BY statement.

If you do not specify the COVOUT option, then the data table contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_TYPE_</code>	type of the estimates reported in this observation. It can take one of the following two values:

EST point estimates of model parameters
 STDERR standard error estimates of model parameters

`_STATUS_` status of the reported estimates. The possible values are listed in the section “[_STATUS_ Variable Values](#)” on page 928.

`<Parameter 1> ... <Parameter M>`

M variables, named after the parameters of all candidate distributions, that contain estimates of the respective parameters. M is the cardinality of the union of parameter name sets from all candidate distributions. In an observation, estimates are populated only for parameters that correspond to the distribution that is indicated by the `_MODEL_` variable. If `_TYPE_` is EST, then the estimates are missing if the model does not converge. If `_TYPE_` is STDERR, then the estimates are missing if covariance estimates cannot be obtained.

If you specify regression effects, then the estimate that is reported for the first parameter of each distribution is the estimate of the base value of the scale or log-transformed scale parameter. For more information, see the section “[Estimating Regression Effects](#)” on page 878.

`<Regression Parameter 1> ... <Regression Parameter K>`

If your effect specification in the [SCALEMODEL](#) statement results in K regression parameters and if you do not specify the [SELECTOUT](#) option, then the OUTEST= data table contains K regression parameter variables.

The name of each variable is formed by using the name of the effect and the levels of the CLASS variables that the effect might contain. If the effect name or level names are too long, then the variable name is constructed by using a part of the effect name and parts of the levels of the CLASS variables. The label of the variable is more descriptive than the name of the variable. The variables contain different types of estimates for their respective regression parameters.

If `_TYPE_` is EST, then a variable contains the estimate of the regression coefficient (β_j) of the respective regression parameter, unless one of the following conditions applies:

- If the model does not converge, then it contains an ordinary missing value.
- If the regression parameter is linearly dependent on other regression parameters for a particular BY group and
 - if you specify the [ZEROEST](#) option, then it contains a value of 0.
 - if you do not specify the [ZEROEST](#) option, then it contains the special missing value .R.
- If you specify the SELECTION statement and if the regression parameter is not part of the selected model for a particular BY group, then
 - if you specify the [ZEROEST](#) option, then it contains a value of 0.
 - if you do not specify the [ZEROEST](#) option, then it contains the special missing value .N.

If `_TYPE_` is STDERR, then a variable contains the estimate of the standard error of the respective regression parameter, unless one of the following conditions applies:

- If covariance estimates cannot be obtained, then it contains an ordinary missing value.
- If the regression parameter is linearly dependent on other regression parameters for a particular BY group, then it contains the special missing value .R.
- If you specify the SELECTION statement and if the regression parameter is not part of the selected model for a particular BY group, then it contains the special missing value .N.

<Offset Variable>

If you specify an OFFSET= variable in the SCALEMODEL statement, then the OUTEST= data table contains a variable that is named after the offset variable. If _TYPE_ is EST, then the value of this variable is 1. If _TYPE_ is STDERR, then the value of this variable is the special missing value .F.

If you specify the COVOUT option in the PROC SEVSELECT statement, then the OUTEST= data table contains additional observations that contain the estimates of the covariance structure. In addition to the variables listed and described previously, the data table contains the following variables that are either new or have a modified description:

TYPE	type of the estimates reported in this observation. For observations that contain rows of the covariance structure, the value is COV.
STATUS	status of the reported estimates. For observations that contain rows of the covariance structure, the status is 0 if covariance estimation was successful. If estimation fails, the status is 1 and a single observation is reported with _TYPE_=COV and missing values for all the parameter variables.
NAME	name of the parameter for the row of covariance matrix that is reported in the current observation.

If _TYPE_ is COV, then a variable contains the estimate of the covariance between the respective parameter and the parameter that is named in the _NAME_ variable, unless one of the following conditions applies:

- If covariance estimates cannot be obtained, then it contains an ordinary missing value.
- If the variable corresponds to a regression parameter and
 - if the regression parameter is linearly dependent on other regression parameters for a particular BY group, then it contains the special missing value .R.
 - if you specify the SELECTION statement and if the regression parameter is not in the selected model for a particular BY group, then it contains the special missing value .N.

If you specify the **SELECTION** statement and the **SELECTOUT** option, then the OUTEST= data table contains only the regression parameters that correspond to the final selected effects. For the selected parameters, the values are written according to the rules stated previously for _TYPE_ values of EST, STDERR, and COV.

If you specify the BY and CLASS statements and if the names and values of some CLASS variables are long, then it is possible for a regression parameter with the same name to have different labels in different BY groups. In such cases, the OUTEST= data table contains rows with _TYPE_='COV' and an additional

variable called `_LABEL_`. You can use the values of `_NAME_` and `_LABEL_` variables in such rows to obtain the label for multilabel regression parameters in each BY group. When the `OUTEST=` data table does not contain the `_LABEL_` variable, then the label of a regression parameter is the label of the variable that contains the estimate of that regression parameter.

OUTMODELINFO= Data Table

The `OUTMODELINFO=` data table records the information about each candidate distribution that you specify in the `DIST` statement. It contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_DEPVAR_</code>	name of the loss variable.
<code>_DESCRIPTION_</code>	descriptive name of the model. This has a nonmissing value only if the <code>DESCRIPTION</code> function has been defined for this model.
<code>_VALID_</code>	validity of the distribution definition. This has a value of 1 for valid definitions and a value of 0 for invalid definitions. If the definition is invalid, then <code>PROC SEVSELECT</code> writes the reason for invalidity to the SAS log.
<code>_PARAMNAME1 ... _PARAMNAMEM</code>	M variables that contain names of parameters of the distribution model, where M is the maximum number of parameters across all the specified distribution models. For a given distribution with m parameters, values of the variables <code>_PARAMNAMEj</code> ($j > m$) are missing.

OUTSTAT= Data Table

The `OUTSTAT=` data table records statistics of fit and model selection information. If you specify BY variables, then the data are organized in BY groups and the data table contains variables that you specify in the BY statement. The data table contains the following variables:

<code>_MODEL_</code>	identifying name of the distribution model. The observation contains information about this distribution.
<code>_NMODELPARAM_</code>	number of parameters in the distribution.
<code>_NESTPARAM_</code>	number of estimated parameters. This includes the regression parameters, if you specify any regression effects.
<code>_NOBS_</code>	number of nonmissing observations that are used for parameter estimation.
<code>_STATUS_</code>	status of the parameter estimation process for this model. The possible values are listed in the section “ _STATUS_ Variable Values ” on page 928.
<code>_SELECTED_</code>	indicator of the best distribution model. If the value is 1, then this model is the best model for the current BY group according to the specified model selection criterion.
<code><Custom Objective></code>	value of the custom objective function that is obtained at the end of the parameter estimation process. <code>PROC SEVSELECT</code> creates this variable only if you specify the <code>OBJECTIVE=</code> option.

Neg2LogLike	value of the log likelihood, multiplied by -2 , that is obtained at the end of the parameter estimation process.
AIC	value of Akaike's information criterion (AIC) that is obtained at the end of the parameter estimation process.
AICC	value of the corrected Akaike's information criterion (AICC) that is obtained at the end of the parameter estimation process.
BIC	value of the Schwarz Bayesian information criterion (SBC) that is obtained at the end of the parameter estimation process.
KS	value of the Kolmogorov-Smirnov (KS) statistic that is obtained at the end of the parameter estimation process.
AD	value of the Anderson-Darling (AD) statistic that is obtained at the end of the parameter estimation process.
CVM	value of the Cramér-von Mises (CvM) statistic that is obtained at the end of the parameter estimation process.

The values of the variables that correspond to the fit statistics and the `_SELECTED_` variable are missing if the parameter estimation process does not converge for the model in the `_MODEL_` variable.

`_STATUS_` Variable Values

The `_STATUS_` variable in the `OUTEST=` and `OUTSTAT=` data tables contains a value that indicates the status of the parameter estimation process for the respective distribution model. The variable can take the following values in the `OUTEST=` data table for `_TYPE_=EST` observations and in the `OUTSTAT=` data table:

- 0 The parameter estimation process converged for this model.
- 301 The parameter estimation process might not have converged for this model because there is no improvement in the objective function value. This might indicate that the initial values of the parameters are optimal, or you can try different convergence criteria in the `NLOPTIONS` statement.
- 302 The parameter estimation process might not have converged for this model because the number of iterations exceeded the maximum allowed value. You can try setting a larger value for the `MAXITER=` options in the `NLOPTIONS` statement.
- 303 The parameter estimation process might not have converged for this model because the number of objective function evaluations exceeded the maximum allowed value. You can try setting a larger value for the `MAXFUNC=` options in the `NLOPTIONS` statement.
- 304 The parameter estimation process might not have converged for this model because the time taken by the process exceeded the maximum allowed value. You can try setting a larger value for the `MAXTIME=` option in the `NLOPTIONS` statement.
- 400 The parameter estimation process did not converge for this model.

The `_STATUS_` variable can take the following values in the `OUTEST=` data table for `_TYPE_=STDERR` and `_TYPE_=COV` observations:

- 0 The covariance and standard error estimates are available and valid.
- 1 The covariance and standard error estimates are not available, because the process of computing covariance estimates failed.

Displayed Output

The SEVSELECT procedure optionally produces displayed output by using the Output Delivery System (ODS). All output is controlled by the PRINT= option in the PROC SEVSELECT statement. [Table 15.7](#) relates the ODS tables to PRINT= options.

Table 15.7 ODS Tables Produced in PROC SEVSELECT

ODS Table Name	Description	Option
AllFitStatistics	Statistics of fit for all the distribution models	PRINT=ALLFITSTATS
BSplineDetails	Basis details for B-spline effects	PRINT=ALL and EFFECT SPLINE statement
ClassLevels	Level information for classification variables	Default for CLASS statement
CollectionLevelInfo	Level details for collection effects	PRINT=ALL and EFFECT COLLECTION statement
ConvergenceStatus	Convergence status of parameter estimation process	PRINT=CONVSTATUS
CorrB	Correlations of parameter estimates	PRINT=CORRB
CovB	Covariances of parameter estimates	PRINT=COVB
DescStats	Descriptive statistics for the response variable	PRINT=DESCSTATS
Distribution	Information about number of parameters in a distribution's model	Default
DistributionInfo	Detailed information about all distributions	PRINT=DISTINFO
EntryCandidates	Details about candidates for entry into the model	PRINT=ALL and SELECTION statement
EstimationDetails	Details of the estimation process for all the distribution models	PRINT=ESTIMATIONDETAILS
FitStatistics	Statistics of fit	PRINT=STATISTICS
InitialValues	Initial parameter values and bounds	PRINT=INITIALVALUES
IterationHistory	Optimization iteration history	PRINT=NLOHISTORY
MMLevelInfo	Level information for multimember effects	PRINT=ALL and EFFECT MULTIMEMBER statement
ModelSelection	Model selection summary	PRINT=SELECTION

Table 15.7 *continued*

ODS Table Name	Description	Option
OptimizationSummary	Optimization summary	PRINT=NLOSUMMARY
ParameterEstimates	Final parameter estimates	PRINT=ESTIMATES
PolyDetails	Number of variables and columns, polynomial degree, and standardization method	PRINT=ALL and EFFECT POLY statement
PolyScaling	Centering and scaling details	PRINT=ALL and EFFECT POLY statement
RegDescStats	Descriptive statistics for the regression effects that do not contain a CLASS variable	PRINT=DESCSTATS
RemovalCandidates	Details about candidates for removal from the model	PRINT=ALL and SELECTION statement
SelectedEffects	List of effects selected for the model	Default for SELECTION statement
SelectionEstimationDetails	Details of the estimation process for all selection steps of a distribution's model	PRINT=ESTIMATIONDETAILS and SELECTION statement
SelectionInfo	Information about the settings for model selection	Default for SELECTION statement
SelectionReason	Reason why the particular model was selected	Default for SELECTION statement
SelectionSummary	Summary information about model selection steps	Default for SELECTION statement
SplineKnots	Knot and boundary knot values	PRINT=ALL and EFFECT SPLINE statement
StopReason	Reason for termination of model selection	Default for SELECTION statement
TPFSplineDetails	Truncated power function (TPF) spline basis details	PRINT=ALL and EFFECT SPLINE statement
Timing	Timing information for various computational stages of the procedure	PRINT=ALL

If you do not specify the PRINT= option, then by default PROC SEVSELECT produces the ModelSelection, ConvergenceStatus, OptimizationSummary, FitStatistics, and ParameterEstimates ODS tables.

If you specify the CLASS statement and if you do not specify the **NOCLPRINT** option, then by default PROC SEVSELECT produces the ClassLevels table.

If you specify the SELECTION statement, then by default PROC SEVSELECT produces the SelectedEffects, SelectionInfo, SelectionReason, SelectionSummary, and StopReason ODS tables.

The following list presents further details of the contents of some of the tables:

AllFitStatistics (PRINT=ALLFITSTATS)

displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge. If no models converge, then this table is not produced. If the table contains more than one model, then the best model according to each statistic is indicated by an asterisk (*) in that statistic's column.

CorrB (PRINT=CORRB)

displays the estimated correlation matrix of parameters of the fitted model. You can control the display of this matrix by specifying the SHAPE= and NOUNITDIAG suboptions of the PRINT([SYMMETRIC=](#)) option. The estimates are not displayed for models whose parameter estimation process does not converge.

CovB (PRINT=COVB)

displays the estimated variance-covariance matrix of parameters of the fitted model. You can control the display of this matrix by specifying the SHAPE= suboption of the PRINT([SYMMETRIC=](#)) option. The estimates are not displayed for models whose parameter estimation process does not converge.

DistributionInfo (PRINT=DISTINFO)

displays the information about all the candidate distribution. It includes the name, the description, the number of distribution parameters, and whether the distribution is valid for the specified modeling task.

FitStatistics (PRINT=STATISTICS)

displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

ModelSelection (PRINT=SELECTION)

displays the model selection table. The table shows the convergence status of each candidate model and the value of the selection criterion along with an indication of the selected model.

ParameterEstimates (PRINT=ESTIMATES)

displays the estimates of parameters of the fitted model. The estimates are not displayed for models whose parameter estimation process does not converge.

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in the “Statistical Graphics Using ODS” chapter in *SAS/STAT User's Guide*.

Before you create graphs, ODS Graphics must be enabled (for example, by using the ODS GRAPHICS ON statement). For more information about enabling and disabling ODS Graphics, see the section “Enabling and Disabling ODS Graphics” in that chapter.

The overall appearance of graphs is controlled by ODS styles. Styles and other aspects of using ODS Graphics are discussed in the section “A Primer on ODS Statistical Graphics” in that chapter.

This section describes how the SEVSELECT procedure uses ODS to create graphics.

ODS Graph Names

PROC SEVSELECT assigns a name to each graph that it creates by using ODS. You can use these names to selectively reference the graphs. The names are listed in [Table 15.8](#).

Table 15.8 ODS Graphics Produced by PROC SEVSELECT

ODS Graph Name	Plot Description	PLOTS= Option
Comparative Plots		
CDFPlot	Comparative CDF plot	CDF
PDFPlot	Comparative PDF plot	PDF
ConditionalPDFPlot	Comparative conditional PDF plot	CONDITIONALPDF
Per Distribution Plots		
CDFDistPlot	CDF plot per distribution	CDFPERDIST
PDFDistPlot	PDF plot per distribution	PDFPERDIST
ConditionalPDFDistPlot	Conditional PDF plot per distribution	CONDITIONALPDFPERDIST
PPPlot	P-P plot of CDF and EDF	PP
QQPlot	Q-Q plot	QQ

Comparative CDF Plot

The comparative CDF plot helps you visually compare the cumulative distribution function (CDF) estimates of all the candidate distribution models to the empirical distribution function (EDF) estimate. The plot does not contain CDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the CDF or EDF estimates.

If you specify truncation, then conditional CDF estimates are plotted. Otherwise, unconditional CDF estimates are plotted. The conditional estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

If you specify regression effects, then the plotted CDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

Comparative PDF Plot

The comparative PDF plot helps you visually compare the probability density function (PDF) estimates of all the candidate distribution models. The plot does not contain PDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the PDF estimates.

If you specify the HISTOGRAM option, then the plot also contains the histogram of response variable values. If you specify the KERNEL option, then the plot also contains the kernel density estimate of the response variable values.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

Comparative Conditional PDF Plot

The comparative conditional PDF plot helps you visually compare the conditional probability density function (PDF) estimates of all the candidate distribution models in up to three ranges of values of the response variable. The plot does not contain conditional PDF estimates for models whose parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the conditional PDF estimates.

For a region that has values in the interval $(L, R]$, PROC SEVSELECT computes the conditional PDF estimate, $\hat{f}^c$, for a distribution as $\hat{f}^c(y) = \hat{f}(y)/(\hat{F}(R) - \hat{F}(L))$, where $\hat{f}$ and $\hat{F}$ denote the estimated unconditional PDF and CDF, respectively, of the distribution. You can control the regions and the values of L and R for each region by specifying the **LEFTQ=**, **RIGHTQ=**, and **SHOWREGION=** suboptions of the **PLOT=CONDITIONALPDF** option.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values in each region.

If you specify regression effects, then the plotted conditional PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

CDF Plot per Distribution

The CDF plot per distribution shows the CDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The plot also contains estimates of the EDF. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the CDF or EDF estimates.

This plot shows the lower and upper pointwise confidence limits for the EDF estimates. For an EDF estimate F_n with standard error σ_n , they are computed as $\max(0, F_n - z_{(1-\alpha/2)}\sigma_n)$ and $\min(1, F_n + z_{(1-\alpha/2)}\sigma_n)$, respectively, where z_p is the p th quantile from the standard normal distribution and α denotes the confidence level that you specify in the **EDFALPHA=** option (the default is $\alpha = 0.05$).

If you specify truncation, then conditional CDF estimates are plotted. Otherwise, unconditional CDF estimates are plotted. The conditional estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

If you specify regression effects, then the plotted CDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

PDF Plot per Distribution

The PDF plot per distribution shows the PDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the PDF estimates.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values. If you specify the **KERNEL** option, then the plot also contains the kernel density estimate of the response variable values.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

Conditional PDF Plot per Distribution

The conditional PDF plot per distribution shows the conditional PDF estimates of each candidate distribution model unless that model's parameter estimation process does not converge. The horizontal axis represents the values of the response variable. The vertical axis represents the values of the conditional PDF estimates.

For a region that has values in the interval $(L, R]$, PROC SEVSELECT computes the conditional PDF estimate, $\hat{f}^c$, for a distribution as $\hat{f}^c(y) = \hat{f}(y)/(\hat{F}(R) - \hat{F}(L))$, where $\hat{f}$ and $\hat{F}$ denote the estimated unconditional PDF and CDF, respectively, of the distribution. You can control the regions and the values of L and R for each region by specifying the **LEFTQ=**, **RIGHTQ=**, and **SHOWREGION=** suboptions of the **PLOT=CONDITIONALPDFPERDIST** option.

If you specify the **HISTOGRAM** option, then the plot also contains the histogram of response variable values in each region.

If you specify regression effects, then the plotted PDF estimates are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

P-P Plot of CDF and EDF

The P-P plot of CDF and EDF is the probability-probability plot that compares the CDF estimates of a distribution to the EDF estimates. A plot is not prepared for models whose parameter estimation process does not converge. The horizontal axis represents the CDF estimates of a candidate distribution, and the vertical axis represents the EDF estimates.

This plot can be interpreted as displaying the data that are used for computing the EDF-based statistics of fit for the given candidate distribution. As described in the section “[EDF-Based Statistics](#)” on page 891, these statistics are computed by comparing the EDF, denoted by $F_n(y)$, to the CDF, denoted by $F(y)$, at each of the response variable values y . Using the probability inverse transform $z = F(y)$, this is equivalent to comparing the EDF of z , denoted by $F_n(z)$, to the CDF of z , denoted by $F(z)$ (D’Agostino and Stephens 1986, Ch. 4). Because the CDF of z is a uniform distribution ($F(z) = z$), the EDF-based statistics can be computed by comparing the estimate of the EDF of z to the estimate of z itself, which is the estimate of the CDF. The horizontal axis of the plot represents $\hat{z} = \hat{F}(y)$. The vertical axis represents the estimated EDF of z , $\hat{F}_n(z)$. The plot contains a scatter plot of $(\hat{z}, \hat{F}_n(z))$ points and a reference line, $F_n(z) = z$, that represents the expected uniform distribution of z . Points that are scattered closer to the reference line indicate a better fit than points that are scattered farther away from the reference line.

If you specify truncation, then the EDF estimates are conditional, as described in the section “[EDF Estimates and Truncation](#)” on page 888. So conditional estimates of CDF are displayed; these estimates are computed by using the method that is described in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

If you specify regression effects, then the displayed CDF estimates, both unconditional and conditional, are those of a mixture distribution. For more information, see the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

Q-Q Plot

The Q-Q plot is a quantile-quantile scatter plot that compares the empirical quantiles to the quantiles from a candidate distribution. A plot is not prepared for models whose parameter estimation process does not converge. The horizontal axis represents the quantiles from a candidate distribution, and the vertical axis represents the empirical quantiles.

Each point in the plot corresponds to a specific value of the EDF estimate, F_n . The Y coordinate is the value of the response variable for which F_n is computed. The X coordinate is computed by using one of the two following methods for a candidate distribution named *dist*:

- If you have defined the *dist_QUANTILE* function that satisfies the requirements listed in the section “[dist_QUANTILE](#)” on page 903, then that function is invoked by using F_n and estimated distribution parameters as arguments. The QUANTILE function is defined in the SasHELP.SVRDIST library for all the predefined distributions.
- If the *dist_QUANTILE* function is not defined, then PROC SEVSELECT numerically inverts the *dist_CDF* function at the CDF value of F_n for the estimated distribution parameters. If the *dist_CDF* function is not defined, then the *exp(dist_LOGCDF)* function is inverted. If the inversion fails, the corresponding point is not plotted in the Q-Q plot.

If you specify truncation, then the EDF estimates are conditional, as described in the section “[EDF Estimates and Truncation](#)” on page 888. The CDF inversion process, whether done numerically or by evaluating the *dist_QUANTILE* function, needs to accept an unconditional CDF value. So the F_n value is first transformed to an unconditional estimate F_n^u as

$$F_n^u = F_n \cdot (\hat{F}(t_{\max}^r) - \hat{F}(t_{\min}^l)) + \hat{F}(t_{\min}^l)$$

where $\hat{F}(t_{\max}^r)$ and $\hat{F}(t_{\min}^l)$ are as defined in the section “[Truncation and Conditional CDF Estimates](#)” on page 874.

If you specify regression effects, then the value of the first distribution parameter is determined by using the method that is equivalent to specifying the DFMIXTURE=MEAN option in the SCALEMODEL statement. This is described in the section “[CDF and PDF Estimates with Regression Effects](#)” on page 881.

Examples: SEVSELECT Procedure

Example 15.1: Defining a Model for the Gaussian Distribution with a Scale Parameter

If you want to estimate the influence of regression effects, then the model needs to be parameterized to have a scale parameter. Although this might not be always possible, it is possible for certain distributions. This example illustrates it for the Gaussian distribution. To obtain a parameterization with scale parameter, you replace the location parameter μ with another parameter, $\alpha = \mu/\sigma$, and define the PDF (f) and the CDF (F) as follows:

$$f(x; \sigma, \alpha) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left(-\frac{1}{2} \left(\frac{x}{\sigma} - \alpha \right)^2 \right)$$

$$F(x; \sigma, \alpha) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{1}{\sqrt{2}} \left(\frac{x}{\sigma} - \alpha \right) \right) \right)$$

Then, you can verify that σ is the scale parameter, because both of the following equalities are true:

$$f(x; \sigma, \alpha) = \frac{1}{\sigma} f\left(\frac{x}{\sigma}; 1, \alpha\right)$$

$$F(x; \sigma, \alpha) = F\left(\frac{x}{\sigma}; 1, \alpha\right)$$

NOTE: The Gaussian distribution is not a commonly used severity distribution. It is used in this example primarily to illustrate the concept of parameterizing a distribution such that it has a scale parameter. Although the distribution has a support over the entire real line, you can fit the distribution with PROC SEVSELECT only if the input sample contains nonnegative values.

The following statements use the alternate parameterization to define a new model named NORMAL_S. The definition is stored in the library Work.Sevexmpl.

```
/*----- Define normal distribution with scale parameter -----*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
  function normal_s_pdf(x, Sigma, Alpha);
    /* Sigma : Scale & Standard Deviation */
    /* Alpha : Scaled mean */
    return ( exp(-(x/Sigma - Alpha)**2/2) /
              (Sigma * sqrt(2*constant('PI')))) );
  endsub;

  function normal_s_cdf(x, Sigma, Alpha);
    /* Sigma : Scale & Standard Deviation */
    /* Alpha : Scaled mean */
    z = x/Sigma - Alpha;
    return (0.5 + 0.5*erf(z/sqrt(2)));
  endsub;

  subroutine normal_s_parinit(dim, x[*], nx[*], F[*], Ftype, Sigma, Alpha);
```

```

    outargs Sigma, Alpha;
    array m[2] / nosymbols;
    /* Compute estimates by using method of moments */
    call svrtutil_rawmoments(dim, x, nx, 2, m);
    Sigma = sqrt(m[2] - m[1]**2);
    Alpha = m[1]/Sigma;
endsub;

subroutine normal_s_lowerbounds(Sigma, Alpha);
    outargs Sigma, Alpha;
    Alpha = .; /* Alpha has no lower bound */
    Sigma = 0; /* Sigma > 0 */
endsub;
quit;

```

An important point to note is that the scale parameter *Sigma* is the first distribution parameter (after the 'x' argument) listed in the signatures of NORMAL_S_PDF and NORMAL_S_CDF functions. *Sigma* is also the first distribution parameter listed in the signatures of other subroutines. This is required by PROC SEVSELECT, so that it can identify which is the scale parameter. When you specify regression effects, PROC SEVSELECT checks whether the first parameter of each candidate distribution is a scale parameter (or a log-transformed scale parameter if [dist_SCALETRANSFORM](#) subroutine is defined for the distribution with LOG as the transform). If it is not, then an appropriate message is written the SAS log and that distribution is not fitted.

Let the following DATA step statements simulate a sample from the normal distribution where the parameter σ is affected by the regressors as follows:

$$\sigma = \exp(1 + 0.5 X_1 + 0.75 X_3 - 2 X_4 + X_5)$$

The sample is simulated such that the regressor X_2 is linearly dependent on regressors X_1 and X_3 .

```

/*--- Simulate a normal distribution's sample affected by regressors ---*/
data testnorm_reg(keep=y x1-x5 Sigma);
    array x{*} x1-x5;
    array b{6} _TEMPORARY_ (1 0.5 . 0.75 -2 1);
    call streaminit(34567);
    label y='Normal Response Influenced by Regressors';

    do n = 1 to 100;
        /* simulate regressors */
        do i = 1 to dim(x);
            x(i) = rand('UNIFORM');
        end;
        /* make x2 linearly dependent on x1 */
        x(2) = 5 * x(1);

        /* compute log of the scale parameter */
        logSigma = b(1);
        do i = 1 to dim(x);
            if (i ne 2) then
                logSigma = logSigma + b(i+1) * x(i);
        end;

        Sigma = exp(logSigma);
    end;

```

```

        y = rand('NORMAL', 25, Sigma);
        output;
    end;
run;

```

The following DATA step loads the data set `Work.Testnorm_reg` into a data table in your CAS session that is associated with the `mycas` CAS engine libref. The DATA step assumes that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```

data mycas.testnorm_reg;
    set testnorm_reg;
run;

```

The following statements use PROC SEVSELECT to fit the NORMAL_S distribution model along with some of the predefined distributions to the simulated sample:

```

/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit models with PROC SEVSELECT -----*/
proc sevselect data=mycas.testnorm_reg print=all;
    loss y;
    scalemodel x1-x5;
    dist Normal_s burr logn pareto weibull;
run;

```

The “Model Selection” table in [Output 15.1.1](#) indicates that all the models, except the Burr distribution model, have converged. Also, only three models, Normal_s, Burr, and Weibull, seem to have a good fit for the data. The table that compares all the fit statistics indicates that Normal_s model is the best according to the likelihood-based statistics; however, the Burr model is the best according to the EDF-based statistics.

Output 15.1.1 Summary of Results for Fitting the Normal Distribution with Regressors

The SEVSELECT Procedure

Model Selection							
		-2 Log					
Distribution	Converged	Likelihood		Selected			
Normal_s	Yes	603.95786		Yes			
Burr	Maybe	612.81685		No			
Logn	Yes	749.20125		No			
Pareto	Yes	841.07022		No			
Weibull	Yes	612.77496		No			

All Fit Statistics							
		-2 Log					
Distribution	Likelihood	AIC	AICC	SBC	KS	AD	CvM
Normal_s	603.95786*	615.95786*	616.86108*	631.58888*	1.52388	4.00152	0.70769
Burr	612.81685	626.81685	628.03424	645.05304	1.50448*	3.90072*	0.63399*
Logn	749.20125	761.20125	762.10448	776.83227	2.88110	16.20558	3.04825
Pareto	841.07022	853.07022	853.97345	868.70124	4.83810	31.60568	6.84046
Weibull	612.77496	624.77496	625.67819	640.40598	1.50490	3.90559	0.63458

Asterisk (*) denotes the best model in the column.

This prompts you to further evaluate why the model with Burr distribution has not converged. The initial values, convergence status, and the optimization summary for the Burr distribution are shown in [Output 15.1.2](#). The initial values table indicates that the regressor X2 is redundant, which is expected. More importantly, the convergence status indicates that it requires more than 50 iterations. PROC SEVSELECT enables you to change several settings of the optimizer by using the [NLOPTIONS](#) statement. In this case, you can increase the limit of 50 on the iterations, change the convergence criterion, or change the technique to something other than the default trust-region technique.

Output 15.1.2 Details of the Fitted Burr Distribution Model

The SEVSELECT Procedure

Burr Distribution

Model Information	
Distribution	Burr
Description	Burr Distribution (Type XII Family)
Distribution Parameters	3
Regression Parameters	4

Initial Parameter Values and Bounds			
Parameter	Initial Value	Lower Bound	Upper Bound
Theta	25.75198	1.05367E-8	Infy
Alpha	2.00000	1.05367E-8	Infy
Gamma	2.00000	1.05367E-8	Infy
x1	0.07345	-709.78271	709.78271
x2	Redundant	-709.78271	709.78271
x3	-0.14056	-709.78271	709.78271
x4	0.27064	-709.78271	709.78271
x5	-0.23230	-709.78271	709.78271

Convergence Status
Needs more than 50 iterations.

Optimization Summary	
Optimization Technique	Trust Region
Iterations	50
Function Calls	137
Log Likelihood	-306.4084238

The following PROC SEVSELECT step uses the NLOPTIONS statement to change the convergence criterion and the limits on the iterations and function evaluations, exclude the lognormal and Pareto distributions that have been confirmed previously to fit the data poorly, and exclude the redundant regressor X2 from the model:

```
/*--- Refit and compare models with higher limit on iterations ---*/
proc sevselect data=mycas.testnorm_reg print=all;
  loss y;
  scalemodel x1 x3-x5;
  dist Normal_s burr weibull;
  nloptions absfconv=2.0e-5 maxiter=100 maxfunc=500;
run;
```

The results shown in [Output 15.1.3](#) indicate that the Burr distribution has now converged. The NORMAL_S distribution is still the best distribution according to the likelihood-based criteria.

Output 15.1.3 Summary of Results after Changing Maximum Number of Iterations

The SEVSELECT Procedure

Model Selection							
Distribution	Converged	-2 Log Likelihood		Selected			
Normal_s	Yes	603.95786	Yes				
Burr	Yes	612.79276	No				
Weibull	Yes	612.77496	No				

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Normal_s	603.95786*	615.95786*	616.86108*	631.58888*	1.52388	4.00152	0.70769
Burr	612.79276	626.79276	628.01015	645.02895	1.50472*	3.90351*	0.63433*
Weibull	612.77496	624.77496	625.67819	640.40598	1.50490	3.90559	0.63458

Asterisk (*) denotes the best model in the column.

Example 15.2: Defining a Model for Mixed-Tail Distributions

In some applications, a few severity values tend to be extreme as compared to the typical values. The extreme values represent the worst-case scenarios and cannot be discarded as outliers. Instead, their distribution must be modeled to prepare for their occurrences. In such cases, it is often useful to fit one distribution to the non-extreme values and another distribution to the extreme values. The *mixed-tail* distribution mixes two distributions: one for the *body* region, which contains the non-extreme values, and another for the *tail* region, which contains the extreme values. The tail distribution is usually a generalized Pareto distribution (GPD), because it is usually good for modeling the conditional excess severity above a threshold. The body distribution can be any distribution. The following definitions are used in describing a generic formulation of a mixed-tail distribution:

$g(x)$	PDF of the body distribution
$G(x)$	CDF of the body distribution
$h(x)$	PDF of the tail distribution
$H(x)$	CDF of the tail distribution
θ	scale parameter for the body distribution
Ω	set of nonscale parameters for the body distribution
ξ	shape parameter for the GPD tail distribution
x_r	normalized value of the response variable at which the tail starts
p_n	mixing probability

Given these notations, the PDF $f(x)$ and the CDF $F(x)$ of the mixed-tail distribution are defined as

$$f(x) = \begin{cases} \frac{p_n}{G(x_b)} g(x) & \text{if } x \leq x_b \\ (1 - p_n) h(x - x_b) & \text{if } x > x_b \end{cases}$$

$$F(x) = \begin{cases} \frac{p_n}{G(x_b)} G(x) & \text{if } x \leq x_b \\ p_n + (1 - p_n) H(x - x_b) & \text{if } x > x_b \end{cases}$$

where $x_b = \theta x_r$ is the value of the response variable at which the tail starts.

These definitions indicate the following:

- The body distribution is conditional on $X \leq x_b$, where X denotes the random response variable.
- The tail distribution is the generalized Pareto distribution of the $(X - x_b)$ values.
- The probability that a response variable value belongs to the body is p_n . Consequently the probability that the value belongs to the tail is $(1 - p_n)$.

The parameters of this distribution are θ , Ω , ξ , x_r , and p_n . The scale of the GPD tail distribution θ_t is computed as

$$\theta_t = \frac{G(x_b; \theta, \Omega) (1 - p_n)}{g(x_b; \theta, \Omega) p_n} = \theta \frac{G(x_r; \theta = 1, \Omega) (1 - p_n)}{g(x_r; \theta = 1, \Omega) p_n}$$

The parameter x_r is usually initialized using a tail index estimation algorithm. One such algorithm is Hill's algorithm (Danielsson et al. 2001), which is implemented by the predefined utility function SVRTUTIL_HILLCUTOFF available to you in the Sashelp.Svrtldist library. The algorithm and the utility function are described in detail in the section “[Predefined Utility Functions](#)” on page 908. The function computes an estimate of x_b , which can be used to compute an initial estimate of x_r as $x_r = x_b / \hat{\theta}$, where $\hat{\theta}$ is the estimate of the scale parameter of the body distribution.

The parameter p_n is usually determined by the domain expert based on the fraction of losses that are expected to belong to the tail.

The following SAS statements define the LOGNGPD distribution model for a mixed-tail distribution with the lognormal distribution as the body distribution and GPD as the tail distribution:

```
/*----- Define lognormal Body-GPD tail mixed distribution -----*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
  function LOGNGPD_DESCRIPTION() $256;
    length desc $256;
    desc1 = "Lognormal Body-GPD Tail Distribution";
    desc2 = " (Mu, Sigma, Xi, and Xr are free parameters;";
    desc3 = " Pn is fixed at 0.8)";
    desc = desc1 || desc2 || desc3;
    return(desc);
  endsub;

  function LOGNGPD_SCALETRANSFORM() $3;
    length xform $3;
    xform = "LOG";
    return (xform);
```

```

endsub;

subroutine LOGNGPD_CONSTANTPARM(Pn);
endsub;

function LOGNGPD_PDF(x, Mu, Sigma, Xi, Xr, Pn);
  cutoff = exp(Mu) * Xr;
  p = CDF('LOGN', cutoff, Mu, Sigma);
  if (x < cutoff + constant('MACEPS')) then do;
    return ((Pn/p)*PDF('LOGN', x, Mu, Sigma));
  end;
  else do;
    gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
    h = (1+Xi*(x-cutoff)/gpd_scale)**(-1-(1/Xi))/gpd_scale;
    return ((1-Pn)*h);
  end;
endsub;

function LOGNGPD_CDF(x, Mu, Sigma, Xi, Xr, Pn);
  cutoff = exp(Mu) * Xr;
  p = CDF('LOGN', cutoff, Mu, Sigma);
  if (x < cutoff + constant('MACEPS')) then do;
    return ((Pn/p)*CDF('LOGN', x, Mu, Sigma));
  end;
  else do;
    gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
    H = 1 - (1 + Xi*((x-cutoff)/gpd_scale))**(-1/Xi);
    return (Pn + (1-Pn)*H);
  end;
endsub;

subroutine LOGNGPD_PARMINIT(dim, x[*], nx[*], F[*], Ftype,
                          Mu, Sigma, Xi, Xr, Pn);
  outargs Mu, Sigma, Xi, Xr, Pn;
  array xe[1] / nosymbols;
  array nxe[1] / nosymbols;

  eps = constant('MACEPS');

  Pn = 0.8; /* Set mixing probability */
  _status_ = .;
  call streaminit(56789);
  Xb = svrtutil_hillcutoff(dim, x, 100, 25, _status_);
  if (missing(_status_) or _status_ = 1) then
    Xb = svrtutil_percentile(Pn, dim, x, F, Ftype);

  /* Initialize lognormal parameters */
  call logn_parminit(dim, x, nx, F, Ftype, Mu, Sigma);
  if (not(missing(Mu))) then
    Xr = Xb/exp(Mu);
  else
    Xr = .;

  /* Prepare arrays for excess values */

```



```

i = 1;
do while (i <= dim and x[i] < Xb+eps);
    i = i + 1;
end;
dime = dim-i+1;
if (dime > 0) then do;
    call dynamic_array(xe, dime);
    call dynamic_array(nxe, dime);
    j = 1;
    do while(i <= dim);
        xe[j] = x[i] - Xb;
        nxe[j] = nx[i];
        i = i + 1;
        j = j + 1;
    end;

    /* Initialize GPD's shape parameter using excess values */
    call gpd_parminit(dime, xe, nxe, F, Ftype, theta_gpd, Xi);
end;
else do;
    Xi = .;
end;
endsub;

subroutine LOGNGPD_LOWERBOUNDS (Mu, Sigma, Xi, Xr, Pn);
    outargs Mu, Sigma, Xi, Xr, Pn;

    Mu    = .; /* Mu has no lower bound */
    Sigma = 0; /* Sigma > 0 */
    Xi    = 0; /* Xi > 0 */
    Xr    = 0; /* Xr > 0 */
endsub;
quit;

```

Note the following points about the LOGNGPD definition:

- In this example, the parameter p_n is not estimated with the maximum likelihood method used by PROC SEVSELECT, so you need to specify it as a *constant* parameter by defining the [dist_CONSTANTPARM](#) subroutine. The signature of the LOGNGPD_CONSTANTPARM subroutine lists only the constant parameter P_n .
- The LOGNGPD_PARMINIT subroutine initializes the parameter x_r by first using the SVRTUTIL_HILLCUTOFF utility function to compute an estimate of the cutoff point $\hat{x}_b$ and then computing $x_r = \hat{x}_b / e^{\hat{\mu}}$. If SVRTUTIL_HILLCUTOFF fails to compute a valid estimate, then the SVRTUTIL_PERCENTILE utility function is used to set $\hat{x}_b$ to the p_n th percentile of the data. The parameter p_n is fixed to 0.8.
- The Sashelp.SvrtDIST library is specified with the LIBRARY= option in the PROC FCMP statement to enable the LOGNGPD_PARMINIT subroutine to use the predefined utility functions (SVRTUTIL_HILLCUTOFF and SVRTUTIL_PERCENTILE) and parameter initialization subroutines (LOGN_PARMINIT and GPD_PARMINIT).
- The LOGNGPD_LOWERBOUNDS subroutine defines the lower bounds for all parameters. This subroutine is required because the parameter Mu has a non-default lower bound. The bounds for *Sigma*

and X_i must be specified. If they are not specified, they are returned as missing values, which PROC SEVSELECT interprets as having no lower bound. You do not need to specify any bounds for the constant parameter P_n , because it is not subject to optimization.

The following DATA step statements simulate a sample from a mixed-tail distribution with a lognormal body and GPD tail. The parameter p_n is fixed to 0.8, the same value used in the LOGNGPD_PARMINIT subroutine defined previously.

```
/*----- Simulate a sample for the mixed-tail distribution -----*/
data testmixdist(keep=y label='Lognormal Body-GPD Tail Sample');
  call streaminit(45678);
  label y='Response Variable';
  N = 1000;
  Mu = 1.5;
  Sigma = 0.25;
  Xi = 0.7;
  Pn = 0.8;

  /* Generate data for the lognormal body */
  Nbody = N*Pn;
  do i=1 to Nbody;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
  end;

  /* Generate data for the GPD tail */
  cutoff = quantile('LOGNORMAL', Pn, Mu, Sigma);
  gpd_scale = (1-Pn) / pdf('LOGNORMAL', cutoff, Mu, Sigma);
  do i=Nbody+1 to N;
    y = cutoff + ((1-rand('UNIFORM'))**(-Xi) - 1)*gpd_scale/Xi;
    output;
  end;
run;
```

The following DATA step loads the data set Work.Testmixdist into a data table in your CAS session that is associated with the mycas CAS engine libref. The DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.testmixdist;
  set testmixdist;
run;
```

The following statements use PROC SEVSELECT to fit the LOGNGPD distribution model to the simulated sample. They also fit three other predefined distributions (BURR, LOGN, and GPD). The final parameter estimates are written to the mycas.Parmest data table.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit LOGNGPD model with PROC SEVSELECT -----*/
proc sevselect data=mycas.testmixdist print=all outest=mycas.parmest;
  loss y;
  dist logngpd burr logn gpd;
run;
```

Some of the results that PROC SEVSELECT produces are shown in [Output 15.2.1](#) and [Output 15.2.2](#). The “Model Selection” table in [Output 15.2.1](#) indicates that all models converged. The “All Fit Statistics” table in [Output 15.2.1](#) shows that the model with LOGNGPD distribution has the best fit according to all the fit statistics. The Burr distribution model is the closest contender to the LOGNGPD model, but the GPD distribution model fits the data very poorly.

Output 15.2.1 Summary of Fitting Mixed-Tail Distribution

The SEVSELECT Procedure

Model Selection							
Distribution	Converged	-2 Log Likelihood		Selected			
Logngpd	Yes	3640		Yes			
Burr	Yes	3687		No			
Logn	Yes	3862		No			
Gpd	Yes	5344		No			

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logngpd	3640*	3650*	3650*	3674*	0.87957*	0.73810*	0.12350*
Burr	3687	3693	3693	3708	0.96199	1.12542	0.17778
Logn	3862	3866	3866	3875	1.38088	2.78456	0.36306
Gpd	5344	5348	5348	5358	7.15101	72.88461	14.86748

Asterisk (*) denotes the best model in the column.

The detailed results for the LOGNGPD distribution are shown in [Output 15.2.2](#). The initial values table shows the fixed value of the P_n parameter that the LOGNGPD_PARMINIT subroutine sets. The table uses the bounds columns to indicate that it is a constant parameter. The last table in the figure shows the final parameter estimates. The estimates of all free parameters are significantly different from 0. As expected, the final estimate of the constant parameter P_n has not changed from its initial value.

Output 15.2.2 Detailed Results for the LOGNGPD Distribution

The SEVSELECT Procedure

Logngpd Distribution

Model Information	
Distribution	Logngpd
Description	Lognormal Body-GPD Tail Distribution (Mu, Sigma, Xi, and Xr are free parameters; Pn is fixed at 0.8)
Distribution Parameters	5

Output 15.2.2 *continued*

Initial Parameter Values and Bounds					
Parameter	Initial Value	Lower Bound	Upper Bound		
Mu	1.32964	-Infy	Infy		
Sigma	0.71508	1.05367E-8	Infy		
Xi	0.29858	1.05367E-8	Infy		
Xr	1.39446	1.05367E-8	Infy		
Pn	0.80000	Constant	Constant		

Convergence Status					
Convergence criterion (GCONV=1E-8) satisfied.					

Optimization Summary					
Optimization Technique		Trust Region			
Iterations		24			
Function Calls		69			
Log Likelihood		-1819.773514			

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	1.61424	0.02518	64.10	<.0001
Sigma	1	0.31740	0.01544	20.56	<.0001
Xi	1	0.53941	0.09017	5.98	<.0001
Xr	1	1.19757	0.03974	30.14	<.0001
Pn	1	0.80000	Constant	.	.

The following SAS statements use the parameter estimates to compute the value where the tail region is estimated to start ($x_b = e^{\hat{\mu}} \hat{x}_r$) and the scale of the GPD tail distribution ($\theta_t = \frac{G(x_b)}{g(x_b)} \frac{(1-p_n)}{p_n}$):

```

/*----- Compute tail cutoff and tail distribution's scale -----*/
data xb_thetat(keep=x_b theta_t);
  set mycas.parmest(where=(_MODEL_='Logngpd' and _TYPE_='EST'));
  x_b = exp(Mu) * Xr;
  theta_t = (CDF('LOGN', x_b, Mu, Sigma) / PDF('LOGN', x_b, Mu, Sigma)) *
            ((1-Pn) / Pn);
run;

proc print data=xb_thetat noobs;
run;

```

Output 15.2.3 Start of the Tail and Scale of the GPD Tail Distribution

x_b	theta_t
6.01665	1.00545

The computed values of x_b and θ_t are shown as `x_b` and `theta_t` in [Output 15.2.3](#). Equipped with this additional derived information, you can now interpret the results of fitting the mixed-tail distribution as follows:

- The tail starts at $y \approx 6.02$. Optimizing the scale-normalized relative cutoff (x_r) in addition to optimizing the scale of the body region ($\theta = e^\mu$) gives you more flexibility in optimizing the absolute cutoff (x_b). If X_r is declared as a constant parameter, then x_b is optimized by virtue of optimizing the scale of the body region ($\theta = e^\mu$), and you must rely on Hill's tail index estimator to yield an initial estimate of x_b that is close to an optimal estimate. By keeping X_r as a free parameter, you account for the possibility that Hill's estimator can yield a suboptimal estimate.
- The values $y \leq 6.02$ follow the lognormal distribution with parameters $\mu \approx 1.614$ and $\sigma \approx 0.317$. These parameter estimates are reasonably close to the parameters of the body distribution that is used for simulating the sample.
- If X_t denotes the loss random variable for the tail defined as $X_t = X - x_b$, where X is the original loss variable, then for this example, $\Pr[X_t = X - 6.02 | X_t > 0]$ follows the GPD density function with scale $\theta_t \approx 1.005$ and shape $\xi \approx 0.539$.

Example 15.3: Fitting a Scaled Tweedie Model with Regressors

The Tweedie distribution is often used in the insurance industry to explain the influence of regression effects on the distribution of losses. PROC SEVSELECT provides a predefined scaled Tweedie distribution (STWEEDIE) that enables you to model the influence of regression effects on the scale parameter. The scale regression model has its own advantages such as the ability to easily account for inflation effects. This example illustrates how that model can be used to evaluate the influence of regression effects on the *mean* of the Tweedie distribution, which is useful in problems such rate-making and pure premium modeling.

Assume a Tweedie process, whose mean μ is affected by k regression effects x_j , $j = 1, \dots, k$, as follows,

$$\mu = \mu_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right)$$

where μ_0 represents the base value of the mean (you can think of μ_0 as $\exp(\beta_0)$, where β_0 is the intercept). This model for the mean is identical to the popular generalized linear model for the mean with a logarithmic link function.

More interestingly, it parallels the model used by PROC SEVSELECT for the scale parameter θ ,

$$\theta = \theta_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right)$$

where θ_0 represents the base value of the scale parameter. As described in the section “[Tweedie Distributions](#)” on page 865, for the parameter range $p \in (1, 2)$, the mean of the Tweedie distribution is given by

$$\mu = \theta \lambda \frac{2-p}{p-1}$$

where λ is the Poisson mean parameter of the scaled Tweedie distribution. This relationship enables you to use the scale regression model to infer the influence of regression effects on the mean of the distribution.

Let the data set `Work.Test_Sevtw` contain a sample generated from a Tweedie distribution with dispersion parameter $\phi = 0.5$, index parameter $p = 1.75$, and the mean parameter that is affected by three regression variables x_1 , x_2 , and x_3 as follows:

$$\mu = 5 \exp(0.25 x_1 - x_2 + 3 x_3)$$

Thus, the population values of regression parameters are $\mu_0 = 5$, $\beta_1 = 0.25$, $\beta_2 = -1$, and $\beta_3 = 3$.

You following DATA step generates a sample of the Tweedie distribution:

```
/*--- Simulate a Tweedie sample that is affected by regressors ---*/
%let samplesize=300;
data test_sevtw(keep=y x1-x3
                label='A Tweedie Sample Affected by Regressors');
  array x{*} x1-x3;
  array b{4} _TEMPORARY_ (5 0.25 -1 3);
  call streaminit(45678);
  label y='Response Influenced by Regressors';

  Phi = 0.5;
  P = 1.75;
  do n = 1 to &samplesize;
    Mu = 0;
    do i = 1 to dim(x);
      x(i) = rand('UNIFORM');
      Mu = Mu + b(i+1) * x(i);
    end;
    Mu = b(1) * exp(Mu); /* b(1) is base value of mean */

    cdf = rand('UNIFORM');
    y = quantile('TWEEDIE', cdf, P, Mu, Phi);
    output;
  end;
run;
```

The following DATA step loads the data set `Work.Test_sevtw` into a data table in your CAS session that is associated with the `mycas` CAS engine libref. The DATA step assumes that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.test_sevtw;
  set test_sevtw;
run;
```

The following PROC SEVSELECT step uses the sample in `mycas.Test_Sevtw` data table to estimate the parameters of the scale regression model for the predefined scaled Tweedie distribution (STWEEDIE) with the dual quasi-Newton (QUANEW) optimization technique:

```
/*--- Fit the scale parameter version of the Tweedie distribution ---*/
proc sevselect data=mycas.test_sevtw outest=mycas.estw covout print=all;
  loss y;
  scalemodel x1-x3;

  dist stweedie;
  nloptions tech=quanew;
run;
```

The dual quasi-Newton technique is used because it requires only the first-order derivatives of the objective function, and it is harder to compute reasonably accurate estimates of the second-order derivatives of Tweedie distribution's PDF with respect to the parameters.

Some of the key results that PROC SEVSELECT prepares are shown in [Output 15.3.1](#) and [Output 15.3.2](#). The model information and the convergence results are shown in [Output 15.3.1](#).

Output 15.3.1 Convergence Results for the STWEEDIE Model with Regressors

The SEVSELECT Procedure

Stweedie Distribution

Model Information	
Distribution	Stweedie
Description	Tweedie Distribution with Scale Parameter
Distribution Parameters	3
Regression Parameters	3

Convergence Status	
Convergence criterion (FCONV=2.220446E-16) satisfied.	

Optimization Summary	
Optimization Technique	Dual Quasi-Newton
Iterations	40
Function Calls	262
Log Likelihood	-1044.321589

The final parameter estimates of the STWEEDIE regression model are shown in [Output 15.3.2](#). The estimate that is reported for the parameter Theta is the estimate of the base value θ_0 . The estimates of regression coefficients β_1 , β_2 , and β_3 are indicated by the rows of x1, x2, and x3, respectively.

Output 15.3.2 Parameter Estimates for the STWEEDIE Model with Regressors

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	0.82650	0.25312	3.27	0.0012
Lambda	1	16.40991	12.10345	1.36	0.1762
P	1	1.75194	0.18994	9.22	<.0001
x1	1	0.27926	0.09872	2.83	0.0050
x2	1	-0.76749	0.10308	-7.45	<.0001
x3	1	3.03182	0.10137	29.91	<.0001

If your goal is to explain the influence of regression effects on the scale parameter, then the output displayed in [Output 15.3.2](#) is sufficient. But, if you want to compute the influence of regression effects on the mean of the distribution, then you need to do some postprocessing. Using the relationship between μ and θ , μ can be

written in terms of the parameters of the STWEEDIE model as

$$\mu = \theta_0 \exp \left(\sum_{j=1}^k \beta_j x_j \right) \lambda \frac{2-p}{p-1}$$

This shows that the parameters β_j are identical for the mean and the scale model, and the base value μ_0 of the mean model is

$$\mu_0 = \theta_0 \lambda \frac{2-p}{p-1}$$

The estimate of μ_0 and the standard error associated with it can be computed by using the property of the functions of maximum likelihood estimators (MLE). If $g(\Omega)$ represents a totally differentiable function of parameters Ω , then the MLE of g has an asymptotic normal distribution with mean $g(\hat{\Omega})$ and covariance $C = (\partial g)' \Sigma (\partial g)$, where $\hat{\Omega}$ is the MLE of Ω , Σ is the estimate of covariance matrix of Ω , and ∂g is the gradient vector of g with respect to Ω evaluated at $\hat{\Omega}$. For μ_0 , the function is $g(\Omega) = \theta_0 \lambda (2-p)/(p-1)$. The gradient vector is

$$\begin{aligned} \partial g &= \left(\frac{\partial g}{\partial \theta_0} \quad \frac{\partial g}{\partial \lambda} \quad \frac{\partial g}{\partial p} \quad \frac{\partial g}{\partial \beta_1} \cdots \frac{\partial g}{\partial \beta_k} \right) \\ &= \left(\frac{\mu_0}{\theta_0} \quad \frac{\mu_0}{\lambda} \quad \frac{-\mu_0}{(p-1)(2-p)} \quad 0 \dots 0 \right) \end{aligned}$$

You can write a DATA step that implements these computations by using the parameter and covariance estimates that the PROC SEVSELECT step prepares. The following statements show one such implementation:

```

/*--- Bring estimates data to the client ---*/
data estw;
    set mycas.estw;
run;

/*--- Compute the estimate, standard error, and p-value of the mean ---*/
data Mu0(keep=Parameter Estimate Stderr T probT);
    Parameter='Mu0';
    label Estimate='Estimate' Stderr='Standard;Error'
           T='t Value' probT='Approx;Pr > |t|';

    array cov{3,3} _temporary_;
    array parms{3} _temporary_;
    array parmrow{3} theta lambda p;
    set estw(where=(_MODEL_='Stweedie' and
                    (_type_='COV' or _type_='EST'))) end=eof;

    n = _n_-1;
    if (n = 0) then do;
        do i=1 to 3;
            parms[i] = parmrow[i];
        end;
    end;
    else do;
        do i=1 to n;

```



```

        cov[n,i] = parmrow[i];
    end;
end;
if (eof or n = 3) then do;
    mu0 = parms[1]*parms[2]*(2-parms[3])/(parms[3]-1);
    Estimate = mu0;

    a = mu0/parms[1]; /* dMu0_T0 */
    b = mu0/parms[2]; /* dMu0_L */
    c = -mu0/((parms[3]-1)*(2-parms[3])); /* dMu0_P */

    varMu0 = a * (a*cov[1,1]+b*cov[2,1]+c*cov[3,1]) +
             b * (a*cov[2,1]+b*cov[2,2]+c*cov[3,2]) +
             c * (a*cov[3,1]+b*cov[3,2]+c*cov[3,3]);

    Stderr = sqrt(varMu0);

    df = &samplesize-6;
    T = mu0/Stderr;
    probt = (1-probt(T, df))*2;
    output;
    stop;
end;
run;

proc print data=Mu0 noobs s='';
run;

```

The estimates of μ_0 that the preceding DATA step prepares are shown in [Output 15.3.3](#). These estimates and the estimates of β_j as shown in [Output 15.3.2](#) are reasonably close (that is, within one or two standard errors) to the parameters of the population from which the sample in mycas.Test_Sevtw data table was drawn.

Output 15.3.3 Estimate of the Base Value Mu0 of the Mean Parameter

Parameter	Estimate	Standard Error	t Value	Approx Pr > t
Mu0	4.47420	0.42226	10.5958	0

Another outcome of using the scaled Tweedie distribution to model the influence of regression effects is that the regression effects also influence the variance V of the Tweedie distribution. The variance is related to the mean as $V = \phi\mu^p$, where ϕ is the dispersion parameter. Using the relationship between the parameters TWEEDIE and STWEEDIE distributions as described in the section “[Tweedie Distributions](#)” on page 865, the regression model for the dispersion parameter is

$$\begin{aligned}
 \log(\phi) &= (2-p)\log(\mu) - \log(\lambda(2-p)) \\
 &= ((2-p)\log(\mu_0) - \log(\lambda(2-p))) + (2-p) \sum_{j=1}^k \beta_j x_j
 \end{aligned}$$

Subsequently, the regression model for the variance is

$$\begin{aligned}\log(V) &= 2\log(\mu) - \log(\lambda(2 - p)) \\ &= (2\log(\mu_0) - \log(\lambda(2 - p))) + 2 \sum_{j=1}^k \beta_j x_j\end{aligned}$$

In summary, PROC SEVSELECT enables you to estimate regression effects on various parameters and statistics of the Tweedie model.

Example 15.4: Fitting Distributions to Interval-Censored Data

In some applications, the data available for modeling might not be exact. A commonly encountered scenario is the use of grouped data from an external agency, which for several reasons, including privacy, does not provide information about individual loss events. The losses are grouped into disjoint bins, and you know only the range and number of values in each bin. Each group is essentially interval-censored, because you know that a loss magnitude is in certain interval, but you do not know the exact magnitude. This example illustrates how you can use PROC SEVSELECT to model such data.

The following DATA step generates sample grouped data for dental insurance claims, which is taken from Klugman, Panjer, and Willmot (1998):

```
/* Grouped dental insurance claims data (Klugman, Panjer, and Willmot 1998) */
data gdental;
    input lowerbd upperbd count @@;
    datalines;
0 25 30 25 50 31 50 100 57 100 150 42 150 250 65 250 500 84
500 1000 45 1000 1500 10 1500 2500 11 2500 4000 3
;
run;
```

To analyze small data sets such as this, you might want to restrict the number of workers that are used by PROC SEVSELECT by submitting the following statements:

```
cas mysess terminate;
cas mysess sessopts=(nworkers=1);
libname mycas cas sessref=mysess;
```

The first CAS statement terminates the current CAS session named mysess. It also clears the mycas libref that was associated with the session. The second CAS statement creates another session that uses only one worker of the server. The LIBNAME statement associates the mycas libref with the new session. For more information about the CAS statement and the CAS LIBNAME engine, see [SAS Cloud Analytic Services: User's Guide](#).

If you want to load a SAS data set into a data table on the CAS server, then it is recommended that you restrict the number of workers before you load the data set. If you restrict the number of workers after you load the data, then PROC SEVSELECT needs to redistribute the data table so that it resides on the restricted number of workers; this can cause unnecessary deterioration of performance.

The following DATA step loads the data set Work.Gdental into a data table in your CAS session that is associated with the mycas CAS engine libref. The DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.gdental;
  set gdental;
run;
```

The following PROC SEVSELECT step fits all the predefined distributions to the data in the mycas.Gdental data table:

```
/* Fit all predefined distributions */
proc sevselect data=mycas.gdental edf=turnbull print=all criterion=aicc;
  loss / rc=lowerbd lc=upperbd;
  weight count;
  dist _predef_;
run;
```

The EDF= option in the PROC SEVSELECT statement specifies that the Turnbull’s method be used for EDF estimation. The LOSS statement specifies the left and right boundaries of each group as the right-censoring and left-censoring limits, respectively. The variable count records the number of losses in each group and is specified in the WEIGHT statement. Note that no response variable is specified in the LOSS statement, which is allowed as long as each observation in the input data table is censored.

Some of the key results that PROC SEVSELECT produces are shown in [Output 15.4.1](#). According to the “Model Selection” table in [Output 15.4.1](#), all distribution models have converged. The “All Fit Statistics” table in [Output 15.4.1](#) indicates that the exponential distribution (EXP) has the best fit for data according to a majority of the likelihood-based statistics and that the Burr distribution (BURR) has the best fit according to all the EDF-based statistics.

Output 15.4.1 Statistics of Fit for Interval-Censored Data

The SEVSELECT Procedure

Model Selection			
Distribution	Converged	AICC	Selected
Burr	Yes	51.41112	No
Exp	Yes	44.64768	Yes
Gamma	Yes	47.63969	No
lgauss	Yes	48.05874	No
Logn	Yes	47.34027	No
Pareto	Yes	47.16908	No
Gpd	Yes	47.16908	No
Weibull	Yes	47.47700	No

Output 15.4.1 continued

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Burr	41.41112*	47.41112	51.41112	48.31888	0.08974*	0.00103*	0.0000816*
Exp	42.14768	44.14768*	44.64768*	44.45026*	0.26412	0.09936	0.01866
Gamma	41.92541	45.92541	47.63969	46.53058	0.19569	0.04608	0.00759
Igauss	42.34445	46.34445	48.05874	46.94962	0.34514	0.12301	0.02562
Logn	41.62598	45.62598	47.34027	46.23115	0.16853	0.01884	0.00333
Pareto	41.45480	45.45480	47.16908	46.05997	0.11423	0.00739	0.0009084
Gpd	41.45480	45.45480	47.16908	46.05997	0.11423	0.00739	0.0009084
Weibull	41.76272	45.76272	47.47700	46.36789	0.17238	0.03293	0.00472

Asterisk (*) denotes the best model in the column.

When the best distributions that are chosen by the likelihood-based and EDF-based statistics are different, you need to decide which fit statistic best represents your objective. In this example, if your objective is to minimize the distance between EDF and CDF values, then you should choose the Burr distribution. On the other hand, if your objective is to maximize the likelihood of the observed data while minimizing the model complexity, then you should choose the exponential distribution. Note that the exponential distribution has worse (lower) raw likelihood than the Burr distribution, but it has better AIC, AICC, and SBC statistics than the Burr distribution because the exponential distribution has only one parameter compared to the three parameters of the Burr distribution. Further, the small sample size of 10 helps accentuate the role of model complexity in the AIC, AICC, and SBC statistics. If the sample size would have been larger, the exponential distribution might not have won according to the likelihood-based statistics.

Example 15.5: Defining a Finite Mixture Model That Has a Scale Parameter

A finite mixture model is a stochastic model that postulates that the probability distribution of the data generation process is a mixture of a finite number of probability distributions. For example, when an insurance company analyzes loss data from multiple policies that are underwritten in different geographic regions, some regions might behave similarly, but the distribution that governs some regions might be different from the distribution that governs other regions. Further, it might not be known which regions behave similarly. Also, the larger amounts of losses might follow a different stochastic process from the stochastic process that governs the smaller amounts of losses. It helps to model all policies together in order to pool the data together and exploit any commonalities among the regions, and the use of a finite mixture model can help capture the differences in distributions across regions and ranges of loss amounts.

Formally, if f_i and F_i denote the PDF and CDF, respectively, of component distribution i and p_i represents the mixing probability that is associated with component i , then the PDF and CDF of the finite mixture of K distribution components are

$$f(x; \Theta, \mathbf{p}) = \sum_{i=1}^K p_i f_i(x; \Theta_i)$$

$$F(x; \Theta, \mathbf{p}) = \sum_{i=1}^K p_i F_i(x; \Theta_i)$$

where Θ_i denotes the parameters of component distribution i and Θ denotes the parameters of the mixture distribution, which is a union of all the Θ_i parameters. $\mathbf{p}$ denotes the set of mixing probabilities. All mixing probabilities must add up to 1 ($\sum_{i=1}^K p_i = 1$).

You can define the finite mixture of a specific number of components and specific distributions for each of the components by defining the FCMP functions for the PDF and CDF. However, in general, it is not possible to fit a scale regression model by using any finite mixture distribution unless you take special care to ensure that the mixture distribution has a scale parameter. This example provides a formulation of a two-component finite mixture model that has a scale parameter.

To start with, each component distribution must have either a scale parameter or a log-transformed scale parameter. Let θ_1 and θ_2 denote the scale parameters of the first and second components, respectively. Let $p_1 = p$ be the mixing probability, which makes $p_2 = 1 - p$ by using the constraint on $\mathbf{p}$. The PDF of the mixture of these two distributions can be written as

$$f(x; \theta_1, \theta_2, \Phi, p) = \frac{p}{\theta_1} f_1\left(\frac{x}{\theta_1}; \Phi_1\right) + \frac{1-p}{\theta_2} f_2\left(\frac{x}{\theta_2}; \Phi_2\right)$$

where Φ_1 and Φ_2 denote the sets of nonscale parameters of the first and second components, respectively, and Φ denotes a union of Φ_1 and Φ_2 . For the mixture to have the scale parameter θ , the PDF must be of the form

$$f(x; \theta, \Phi', p) = \frac{1}{\theta} \left(p f_1\left(\frac{x}{\theta}; \Phi'_1\right) + (1-p) f_2\left(\frac{x}{\theta}; \Phi'_2\right) \right)$$

where Φ' , Φ'_1 , and Φ'_2 denote the modified sets of nonscale parameters. One simple way to achieve this is to make $\theta_1 = \theta_2 = \theta$ and $\Phi' = \Phi$; that is, you simply equate the scale parameters of both components and keep the set of nonscale parameters unchanged. However, forcing the scale parameters to be equal in both components is restrictive, because the mixture cannot model potential differences in the scales of the two components. A better approach is to tie the scale parameters of the two components by a ratio such that $\theta_1 = \theta$ and $\theta_2 = \rho\theta$. If the ratio parameter ρ is estimated along with the other parameters, then the mixture distribution becomes flexible enough to model the variations across the scale parameters of individual components.

To summarize, the PDF and CDF are of the following form for the two-component mixture that has a scale parameter:

$$\begin{aligned} f(x; \theta, \rho, \Phi, p) &= \frac{1}{\theta} \left(p f_1\left(\frac{x}{\theta}; \Phi_1\right) + (1-p) f_2\left(\frac{x}{\theta}; \rho, \Phi_2\right) \right) \\ F(x; \theta, \rho, \Phi, p) &= p F_1\left(\frac{x}{\theta}; \Phi_1\right) + (1-p) F_2\left(\frac{x}{\theta}; \rho, \Phi_2\right) \end{aligned}$$

This can be generalized to a mixture of K components by introducing the $K - 1$ ratio parameters ρ_i that relate the scale parameters of each of the K components to the scale parameter θ of the mixture distribution as follows:

$$\begin{aligned} \theta_1 &= \theta \\ \theta_i &= \rho_i \theta; i \in [2, K] \end{aligned}$$

In order to illustrate this approach, define a mixture of two lognormal distributions by using the following PDF function:

$$f(x; \mu, \sigma_1, p_2, \rho_2, \sigma_2) = \frac{(1 - p_2)}{\sigma_1 x \sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu)^2}{2\sigma_1^2}\right) + \frac{p_2}{\sigma_2 x \sqrt{2\pi}} \exp\left(\frac{-(\log(x) - \mu - \log(\rho_2))^2}{2\sigma_2^2}\right)$$

You can verify that μ serves as the log of the scale parameter θ ($\mu = \log(\theta)$).

The following PROC FCMP steps encode this formulation in a distribution named SLOGNMIX2 for use with PROC SEVSELECT:

```

/*- Define mixture of 2 lognormal distributions with a log-scale parameter -*/
proc fcmp library=sashelp.svrtldist outlib=work.sevexmpl.models;
  function slognmix2_description() $128;
    return ("Mixture of two lognormals with a log-scale parameter Mu");
  endsub;

  function slognmix2_scaletransform() $8;
    return ("LOG");
  endsub;

  function slognmix2_pdf(x, Mu, Sigma1, p2, Rho2, Sigma2);
    Mu1 = Mu;
    Mu2 = Mu + log(Rho2);
    pdf1 = logn_pdf(x, Mu1, Sigma1);
    pdf2 = logn_pdf(x, Mu2, Sigma2);
    return ((1-p2)*pdf1 + p2*pdf2);
  endsub;

  function slognmix2_cdf(x, Mu, Sigma1, p2, Rho2, Sigma2);
    Mu1 = Mu;
    Mu2 = Mu + log(Rho2);
    cdf1 = logn_cdf(x, Mu1, Sigma1);
    cdf2 = logn_cdf(x, Mu2, Sigma2);
    return ((1-p2)*cdf1 + p2*cdf2);
  endsub;

  subroutine slognmix2_parminit(dim, x[*], nx[*], F[*], Ftype,
                                Mu, Sigma1, p2, Rho2, Sigma2);
    outargs Mu, Sigma1, p2, Rho2, Sigma2;
    array m[1] / nosymbols;
    p2 = 0.5;
    Rho2 = 0.5;
    median = svrtutil_percentile(0.5, dim, x, F, Ftype);
    Mu = log(2*median/1.5);
    call svrtutil_rawmoments(dim, x, nx, 1, m);
    lm1 = log(m[1]);

    /* Search Rho2 that makes log(sample mean) > Mu */
    do while (lm1 <= Mu and Rho2 < 1);
      Rho2 = Rho2 + 0.01;
    end;
  endsub;
endproc;

```

```

    Mu = log(2*median/(1+Rho2));
end;
if (Rho2 >= 1) then
    /* If Mu cannot be decreased enough to make it less
       than log(sample mean), then revert to Rho2=0.5.
       That will set Sigma1 and possibly Sigma2 to missing.
       PROC SEVSELECT replaces missing initial values with 0.001. */
    Mu = log(2*median/1.5);

    Sigma1 = sqrt(2.0*(log(m[1])-Mu));
    Sigma2 = sqrt(2.0*(log(m[1])-Mu-log(Rho2)));
endsub;

subroutine slognmix2_lowerbounds(Mu, Sigma1, p2, Rho2, Sigma2);
    outargs Mu, Sigma1, p2, Rho2, Sigma2;
    Mu = .;      /* Mu has no lower bound */
    Sigma1 = 0; /* Sigma1 > 0 */
    p2 = 0;      /* p2 > 0 */
    Rho2 = 0;    /* Rho2 > 0 */
    Sigma2 = 0; /* Sigma2 > 0 */
endsub;

subroutine slognmix2_upperbounds(Mu, Sigma1, p2, Rho2, Sigma2);
    outargs Mu, Sigma1, p2, Rho2, Sigma2;
    Mu = .;      /* Mu has no upper bound */
    Sigma1 = .; /* Sigma1 has no upper bound */
    p2 = 1;      /* p2 < 1 */
    Rho2 = 1;    /* Rho2 < 1 */
    Sigma2 = .; /* Sigma2 has no upper bound */
endsub;
quit;

```

As shown in previous examples, an important aspect of defining a distribution for use with PROC SEVSELECT is the definition of the PARMINIT subroutine that initializes the parameters. For mixture distributions, in general, the parameter initialization is a nontrivial task. For a two-component mixture, some simplifying assumptions make the problem easier to handle. For the initialization of SLOGNMIX2, the initial values of p_1 and p_2 are fixed at 0.5, and the following two simplifying assumptions are made:

- The median of the mixture is the average of the medians of the two components:

$$F^{-1}(0.5) = (\exp(\mu_1) + \exp(\mu_2))/2 = \exp(\mu)(1 + \rho_2)/2$$

Solution of this equation yields the value of μ in terms of ρ_2 and the sample median.

- Each component has the same mean, which implies the following:

$$\exp(\mu + \sigma_1^2/2) = \exp(\mu + \log(\rho_2) + \sigma_2^2/2)$$

If X_i represents the random variable of component distribution i and X represents the random variable of the mixture distribution, then the following equation holds for the raw moment of any order k :

$$E[X^k] = \sum_{i=1}^K p_i E[X_i^k]$$

This, in conjunction with the assumption on component means, leads to the equations

$$\log(m_1) = \mu + \frac{\sigma_1^2}{2}$$

$$\log(m_1) = \mu + \log(\rho_2) + \frac{\sigma_2^2}{2}$$

where m_1 denotes the first raw moment of the sample. Solving these equations leads to the following values of σ_1 and σ_2 :

$$\sigma_1^2 = 2(\log(m_1) - \mu)$$

$$\sigma_2^2 = 2(\log(m_1) - \mu - \log(\rho_2))$$

Note that σ_1 has a valid value only if $\log(m_1) > \mu$. Among the many possible methods of ensuring this condition, the SLOGNMIX2_PARMINT subroutine uses the method of doing a linear search over ρ_2 .

Even when the preceding assumptions are not true for a particular problem, they produce reasonable initial values to help guide the nonlinear optimizer to an acceptable optimum if the mixture of two lognormal distributions is indeed a good fit for your input data. This is illustrated by the results of the following steps that fit the SLOGNMIX2 distribution to simulated data, which have different means for the two components (12.18 and 22.76, respectively), and the median of the sample (15.95) is not equal to the average of the medians of the two components (7.39 and 20.09, respectively):

```

/*----- Simulate a lognormal mixture sample -----*/
data testlognmix(keep=y);
  call streaminit(12345);
  Mu1 = 2;
  Sigma1 = 1;
  i = 0;
  do j=1 to 2000;
    y = exp(Mu1) * rand('LOGNORMAL')**Sigma1;
    output;
  end;
  Mu2 = 3;
  Sigma2 = 0.5;
  do j=1 to 3000;
    y = exp(Mu2) * rand('LOGNORMAL')**Sigma2;
    output;
  end;
run;

/*----- Load data into the CAS server -----*/
data mycas.testlognmix;
  set testlognmix;
run;

/*-- Fit and compare scale regression models with 2-component --*/
/*-- lognormal mixture and the standard lognormal distribution --*/
options cmplib=(work.sevexmpl);

proc sevselect data=mycas.testlognmix print=all;
```



```

loss y;
dist slognmix2 logn;
run;

```

The comparison of the fit statistics of SLOGNMIX2 and LOGN, as shown in [Output 15.5.1](#), confirms that the two-component mixture is certainly a better fit to these data than the single lognormal distribution.

Output 15.5.1 Comparison of Fitting One versus Two Lognormal Components to Mixture Data

The SEVSELECT Procedure

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Slognmix2	38343*	38353*	38353*	38386*	0.94405*	1.30448*	0.20102*
Logn	39073	39077	39077	39090	3.57373	23.45814	4.09737

Asterisk (*) denotes the best model in the column.

The detailed results for the SLOGNMIX2 distribution are shown in [Output 15.5.2](#). According to the “Initial Parameter Values and Bounds” table, the initial value of ρ_2 is not 0.5, indicating that a linear search was conducted to ensure $\log(m_1) > \mu$.

Output 15.5.2 Detailed Estimation Results for the SLOGNMIX2 Distribution

The SEVSELECT Procedure

Slognmix2 Distribution

Model Information	
Distribution	Slognmix2
Description	Mixture of two lognormals with a log-scale parameter Mu
Distribution Parameters	5

Initial Parameter Values and Bounds			
Parameter	Initial Value	Lower Bound	Upper Bound
Mu	2.92251	-Infy	Infy
Sigma1	0.06338	1.05367E-8	Infy
P2	0.50000	1.05367E-8	1.00000
Rho2	0.72333	1.05367E-8	1.00000
Sigma2	0.80670	1.05367E-8	Infy

Convergence Status
Convergence criterion (GCONV=1E-8) satisfied.

Optimization Summary	
Optimization Technique	Trust Region
Iterations	9
Function Calls	24
Log Likelihood	-19171.54807

Output 15.5.2 continued

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	3.00922	0.01554	193.68	<.0001
Sigma1	1	0.49516	0.01451	34.13	<.0001
P2	1	0.40619	0.02600	15.62	<.0001
Rho2	1	0.37212	0.02038	18.26	<.0001
Sigma2	1	1.00019	0.02124	47.09	<.0001

By using the relationship that $\mu_2 = \mu + \log(\rho_2)$, you can see that the final parameter estimates are reasonably close to the true parameter values that were used to simulate the input sample.

Example 15.6: Scale Regression with Rich Regression Effects

This example illustrates the use of regression effects that include CLASS variables and interaction effects.

Consider that you, as an actuary at an automobile insurance company, want to evaluate the effect of certain external factors on the distribution of the severity of the losses that your policyholders incur. Such analysis can help you determine the relative differences in premiums that you should charge to policyholders who have different characteristics. Assume that when you collect and record the information about each claim, you also collect and record some key characteristics of the policyholder and the vehicle that is involved in the claim. This example focuses on the following five factors: type of car, safety rating of the car, gender of the policyholder, education level of the policyholder, and annual household income of the policyholder (which can be thought of as a proxy for the luxury level of the car). Let these regressors be recorded in the variables CarType (1: sedan, 2: sport utility vehicle), CarSafety (scaled to be between 0 and 1, the safest being 1), Gender (1: female, 2: male), Education (1: high school graduate, 2: college graduate, 3: advanced degree holder), and Income (scaled by a factor of 1/100,000), respectively. Let the historical data about the severity of each loss be recorded in the LossAmount variable of the mycas.Losses data table in the CAS libref. Let the data table also contain two additional variables, Deductible and Limit, that record the deductible and ground-up loss limit provisions, respectively, of the insurance policy that the policyholder has. The limit on ground-up loss is usually derived from the payment limit that a typical insurance policy states. Deductible serves as the left-truncation variable, and Limit serves as the right-censoring variable.

The following SAS statements simulate an example of the mycas.Losses data table:

```
proc format casfmtlib='myfmtlib';
    value genderFmt 1='Female'
                  2='Male';
run;

data losses(keep=gender carType education carSafety income
               lossAmount deductible limit);
    call streaminit(12345);
    array sx{8} _temporary_;
    array sbeta{9} _TEMPORARY_ (5 0.6 0.4 -0.75 -0.3 0.4 0.7 -0.5 -0.3);

    length carType $8 education $16;
    format gender genderFmt.;
```

```

sigma = 0.5;
do lossEventId=1 to 6000;
  /* Simulate policyholder and vehicle attributes */
  do i=1 to dim(sx);
    sx(i) = 0;
  end;

  if (rand('UNIFORM') < 0.5) then do;
    gender = 1; * female;
    sx(2) = 1;
  end;
  else do;
    gender = 2; * male;
  end;

  if (rand('UNIFORM') < 0.7) then do;
    carInt = 1;
    carType = 'Sedan';
  end;
  else do;
    carInt = 2;
    carType = 'SUV';
    sx(1) = 1;
  end;

  educationLevel = rand('UNIFORM');
  if (educationLevel < 0.5) then do;
    eduInt = 1;
    education = 'High School';
  end;
  else if (educationLevel < 0.85) then do;
    eduInt = 2;
    education = 'College';
    if (carInt=1) then
      sx(8) = 1;
    else
      sx(6) = 1;
  end;
  else do;
    eduInt = 3;
    education = 'AdvancedDegree';
    if (carInt=1) then
      sx(7) = 1;
    else
      sx(5) = 1;
  end;

  carSafety = rand('UNIFORM'); /* scaled to be between 0 & 1 */
  sx(3) = carSafety;

  income = MAX(15000,int(rand('NORMAL', eduInt*30000, 50000)))/100000;
  sx(4) = income;

```

```

/* Simulate lognormal severity */
Mu = sbeta(1);
do i=1 to dim(sx);
    Mu = Mu + sx(i) * sbeta(i+1);
end;
lossAmount = exp(Mu) * rand('LOGNORMAL')**Sigma;
loglossAmount = log(lossAmount);

deductible = lossAmount * rand('UNIFORM');
if (rand('UNIFORM') < 0.25) then
    limit = lossAmount;

output;
end;
run;

data mycas.losses;
    set losses;
run;

```

The variables CarType, Education, and Gender each contain a known, finite set of discrete values. By specifying such variables as classification variables, you can separately identify the effect of each level of the variable on the severity distribution. For example, you might be interested in finding out how the magnitude of loss for a sport utility vehicle (SUV) differs from that for a sedan. This is an example of a main effect. You might also want to evaluate how the distribution of losses that are incurred by a policyholder with a college degree who drives a SUV differs from that of a policyholder with an advanced degree who drives a sedan. This is an example of an interaction effect. You can include various such types of effects in the scale regression model. For more information about the effect types, see the section “[Specification and Parameterization of Model Effects](#)” on page 49 in Chapter 3, “[Shared Concepts](#).”

Analyzing such a rich set of regression effects can help you make more accurate predictions about the losses that a new applicant with certain characteristics might incur when he or she requests insurance for a specific vehicle, which can further help you with ratemaking decisions.

The following PROC SEVSELECT step fits the scale regression model with a lognormal distribution to data in the mycas.Losses data table, and stores the model and parameter estimate information in the mycas.Est data table on the CAS server:

```

/* Fit scale regression model with different types of regression effects */
proc sevselect data=mycas.losses outest=mycas.est print=all;
    loss lossAmount / lt=deductible rc=limit;
    class carType gender education;
    scalemodel carType gender carSafety income education*carType
               income*gender carSafety*income;
    dist logn;
run;

```

The SCALEMODEL statement in the preceding PROC SEVSELECT step includes two main effects (carType and gender), two singleton continuous effects (carSafety and income), one interaction effect (education*carType), one continuous-by-class effect (income*gender), and one polynomial continuous effect (carSafety*income).

When you specify a CLASS statement, it is recommended that you observe the “Class Level Information” table. For this example, the table is shown in [Output 15.6.1](#). Note that if you specify BY-group processing,

then the class level information might change from one BY group to the next, potentially resulting in a different parameterization for each BY group.

Output 15.6.1 Class Level Information Table

The SEVSELECT Procedure

Class Level Information		
Class	Levels	Values
carType	2	SUV Sedan
gender	2	Female Male
education	3	AdvancedDegree College High School

The regression modeling results for the lognormal distribution are shown in [Output 15.6.2](#). The “Initial Parameter Values and Bounds” table is important especially because the preceding PROC SEVSELECT step uses the default GLM parameterization, which is a singular parameterization—that is, it results in some redundant parameters. As shown in the table, the redundant parameters correspond to the last level of each classification variable; this correspondence is a defining characteristic of a GLM parameterization. An alternative would be to use the reference parameterization by specifying the PARAM=REFERENCE option in the CLASS statement, which does not generate redundant parameters for effects that contain CLASS variables and enables you to specify a reference level for each CLASS variable.

Output 15.6.2 Initial Values for the Scale Regression Model with Class and Interaction Effects

Initial Parameter Values and Bounds			
Parameter	Initial Value	Lower Bound	Upper Bound
Mu	4.85228	-709.78271	709.78271
Sigma	0.52348	1.05367E-8	Inf
carType SUV	0.54686	-709.78271	709.78271
carType Sedan	Redundant	-709.78271	709.78271
gender Female	0.34893	-709.78271	709.78271
gender Male	Redundant	-709.78271	709.78271
carSafety	-0.63504	-709.78271	709.78271
income	-0.24031	-709.78271	709.78271
carType SUV * education AdvancedDegree	0.32719	-709.78271	709.78271
carType SUV * education College	0.68899	-709.78271	709.78271
carType SUV * education High School	Redundant	-709.78271	709.78271
carType Sedan * education AdvancedDegree	-0.44650	-709.78271	709.78271
carType Sedan * education College	-0.26834	-709.78271	709.78271
carType Sedan * education High School	Redundant	-709.78271	709.78271
income * gender Female	0.00843	-709.78271	709.78271
income * gender Male	Redundant	-709.78271	709.78271
carSafety * income	-0.04744	-709.78271	709.78271

The convergence and optimization summary information in [Output 15.6.3](#) indicates that the scale regression model for the lognormal distribution has converged with the default optimization technique in five iterations.

Output 15.6.3 Optimization Summary for the Scale Regression Model with Class and Interaction Effects

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	

Optimization Summary	
Optimization Technique	Trust Region
Iterations	5
Function Calls	14
Log Likelihood	-19716.25785

The “Parameter Estimates” table in [Output 15.6.4](#) shows the distribution parameter estimates and estimates for various regression effects. You can use the estimates for effects that contain CLASS variables to infer the relative influence of various CLASS variable levels. For example, on average, the magnitude of losses that are incurred by the female drivers is $\exp(0.41714) \approx 1.52$ times greater than that of male drivers, and an SUV driver with an advanced degree incurs a loss that is on average $\exp(0.42454) / \exp(-0.35208) \approx 2.174$ times greater than the loss that a college-educated sedan driver incurs. Neither the continuous-by-class effect `income*gender` nor the polynomial continuous effect `carSafety*income` is significant in this example.

Output 15.6.4 Parameter Estimates for the Scale Regression with Class and Interaction Effects

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	5.12332	0.03792	135.11	<.0001
Sigma	1	0.56941	0.00744	76.56	<.0001
carType SUV	1	0.64686	0.02989	21.64	<.0001
carType Sedan	0	0	.	.	.
gender Female	1	0.41714	0.03213	12.98	<.0001
gender Male	0	0	.	.	.
carSafety	1	-0.88078	0.05532	-15.92	<.0001
income	1	-0.38045	0.05128	-7.42	<.0001
carType SUV * education AdvancedDegree	1	0.42454	0.05107	8.31	<.0001
carType SUV * education College	1	0.73228	0.03738	19.59	<.0001
carType SUV * education High School	0	0	.	.	.
carType Sedan * education AdvancedDegree	1	-0.56685	0.03525	-16.08	<.0001
carType Sedan * education College	1	-0.35208	0.02618	-13.45	<.0001
carType Sedan * education High School	0	0	.	.	.
income * gender Female	1	0.01288	0.04394	0.29	0.7694
income * gender Male	0	0	.	.	.
carSafety * income	1	0.06499	0.07547	0.86	0.3892

If you want to update the model when new claims data arrive, then you can potentially speed up the estimation process by specifying the `OUTEST=` data table that is created by the preceding PROC SEVSELECT step as an `INEST=` data table in a new PROC SEVSELECT step. To illustrate, the following PROC SEVSELECT step refits the model on the same input data as the preceding PROC SEVSELECT, but it uses the `mycas.Est` data table that is created by that step as an `INEST=` data table:

```

/* Refit scale regression model on new data */
proc sevselect data=mycas.losses inest=mycas.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carSafety income education*carType
             income*gender carSafety*income;
  dist logn;
run;

```

Because the INEST= data table is used to initialize the distribution and regression parameters, the optimization occurs in very few iterations, as shown in [Output 15.6.5](#).

Output 15.6.5 Optimization Summary for Refitting the Scale Regression Model with the INEST= Option
The SEVSELECT Procedure

Logn Distribution

Convergence Status	
Convergence criterion (ABSGCONV=0.00001) satisfied.	

Optimization Summary	
Optimization Technique	Trust Region
Iterations	0
Function Calls	4
Log Likelihood	-19716.25785

Example 15.7: Scale Regression Model Selection

This example extends “[Example 15.6: Scale Regression with Rich Regression Effects](#)” to illustrate the model selection features of the SEVSELECT procedure. There are two phases of model selection when you fit a scale regression model. First, you want to select an optimal set of regression effects for each distribution’s scale regression model. The SELECTION statement provides several methods to do that. Second, you want to identify the best severity distribution by comparing the final selected scale regression models for each distribution. PROC SEVSELECT displays tables to help you do that.

The following SEVSELECT step uses the same data table as in [Example 15.6](#) to find the best set of scale regression effects for three severity distributions—lognormal (Logn), Burr, and Weibull:

```

proc sevselect data=mycas.losses outest=mycas.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carSafety income education*carType
             income*gender carSafety*income;
  selection;
  dist logn burr weibull;
  output out=mycas.score copyvars=(carType gender education carSafety income)
         functions=(mean) quantiles=(points=0.5 0.975 names=(median var));
run;

```

The SELECTION statement without any options uses the stepwise selection method by default. PROC SEVSELECT reports the settings of the selection process in the “Selection Information” table, as shown in [Output 15.7.1](#).

Output 15.7.1 Selection Method Settings

The SEVSELECT Procedure

Logn Distribution

Selection Information	
Selection Method	Stepwise
Select Criterion	SBC
Stop Criterion	SBC
Effect Hierarchy Enforced	None
Stop Horizon	3

For each severity distribution, PROC SEVSELECT reports the summary of the selection process, as shown in [Output 15.7.2](#) for the lognormal distribution. The “Selection Summary” table shows the order in which regression effects enter or leave the model. In this case, no effect leaves the model after entering the model. In the first step, PROC SEVSELECT fits seven models, each with an intercept and one of the seven eligible effects in the SCALEMODEL statement. The model that contains the intercept and carType*education effect has the lowest value of the Schwarz Bayesian criterion (SBC) statistic in the first step, and hence it enters the model. In the second step, PROC SEVSELECT compares the model that does not contain the existing effect, carType*education, with models that contain that effect and one of the remaining six effects. The model that contains the intercept, carType*education, and carSafety effects is the best in the second step according to the SBC statistic. The process of removing an effect from the model or adding an effect that is not in the model continues until the stop criterion does not improve or the stop horizon condition is met. As the results in [Output 15.7.2](#) show, the optimal value of the SBC criterion is achieved in the fourth step. The final scale regression model for the lognormal distribution contains the following effects: intercept, gender, carSafety, carType*education, and income*gender.

Output 15.7.2 Stepwise Selection Summary for the Lognormal Distribution

The SEVSELECT Procedure

Logn Distribution Selection Details

Selection Summary			
Step	Effect Entered	Number Effects In	SBC
0	Intercept	1	35975.2456
1	carType*education	2	34121.0008
2	carSafety	3	33709.8419
3	income*gender	4	33298.5871
4	gender	5	33159.7903*
* Optimal Value Of Criterion			

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

Output 15.7.2 *continued*

The model at step 4 is selected.

Selected Effects: Intercept gender carSafety carType*education income*gender

The selection summary information for the Weibull distribution is shown in [Output 15.7.3](#). It shows that the best model in the sixth step is obtained by removing the carSafety*income effect, which was added in the second step. The model in the sixth step also has the lowest value of the SBC criterion; hence, it is the best model and contains the following effects: intercept, gender, carSafety, income, and carType*education.

Output 15.7.3 Stepwise Selection Summary for the Weibull Distribution**The SEVSELECT Procedure****Weibull Distribution
Selection Details**

Selection Summary				
Step	Effect Entered	Effect Removed	Number Effects In	SBC
0	Intercept		1	36381.5042
1	carType*education		2	34477.5474
2	carSafety*income		3	34092.5186
3	gender		4	33684.7598
4	carSafety		5	33573.1854
5	income		6	33517.6460
6		carSafety*income	5	33509.5021*
* Optimal Value Of Criterion				

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

The model at step 6 is selected.

Selected Effects: Intercept gender carSafety income carType*education

PROC SEVSELECT reports the optimization summary, fit statistics, and parameter estimates for the final selected model of each severity distribution. The results for the lognormal distribution are shown in [Output 15.7.4](#). The “Parameter Estimates” table shows that all regression parameters in the final model are statistically significant. Comparison of these parameter estimates with those in [Output 15.6.4](#), which were obtained without using the SELECTION statement, shows that the statistically insignificant effect, carSafety*income, is not present in the final selected model. The comparison also shows that the stepwise selection process prefers income*gender effect over the income effect and, because of the exclusion of the income effect, the income*gender effect has become statistically significant.

Output 15.7.4 Selected Scale Regression Model for the Lognormal Distribution**Convergence Status**

Convergence criterion (GCONV=1E-8) satisfied.

Output 15.7.4 *continued*

Optimization Summary					
Optimization Technique		Trust Region			
Iterations		5			
Function Calls		14			
Log Likelihood		-16533.05056			

Fit Statistics		
-2 Log Likelihood		33066
Akaike's Information Criterion		33088
Corrected Akaike's Information Criterion		33088
Schwarz's Bayesian Information Criterion		33160
Kolmogorov-Smirnov Statistic		6.48511
Anderson-Darling Statistic		217.45448
Cramer-von Mises Statistic		18.37380

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Mu	1	5.09549	0.03326	153.18	<.0001
Sigma	1	0.56924	0.00811	70.18	<.0001
gender Female	1	0.43162	0.03516	12.28	<.0001
gender Male	0	0	.	.	.
carSafety	1	-0.83597	0.03574	-23.39	<.0001
carType SUV * education AdvancedDegree	1	1.02458	0.05115	20.03	<.0001
carType SUV * education College	1	1.40166	0.03516	39.87	<.0001
carType SUV * education High School	1	0.64207	0.03273	19.61	<.0001
carType Sedan * education AdvancedDegree	1	-0.54784	0.03836	-14.28	<.0001
carType Sedan * education College	1	-0.34621	0.02861	-12.10	<.0001
carType Sedan * education High School	0	0	.	.	.
income * gender Female	1	-0.34466	0.03490	-9.88	<.0001
income * gender Male	1	-0.36093	0.03588	-10.06	<.0001

After the effect selection process chooses the best scale regression model for each severity distribution, the next phase of model selection is to compare the final models across all severity distributions and identify the best severity distribution. You can do that by using the “All Fit Statistics” table, which helps you compare the final selected models according to all the fit statistics. For this example, [Output 15.7.5](#) compares the final models of the three severity distributions—lognormal, Burr, and Weibull.

Output 15.7.5 Comparison of the Selected Scale Regression Models for Different Severity Distributions

All Fit Statistics							
Distribution	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logn	33066*	33088*	33088*	33160*	6.48511	217.45448	18.37380
Burr	33100	33124	33125	33203	6.66587	173.98022*	18.46358
Weibull	33424	33444	33444	33510	6.06631*	253.53062	12.26544*

Asterisk (*) denotes the best model in the column.

The lognormal distribution's final selected model is the best according to all the likelihood-based statistics, the Burr distribution's final selected model is the best according to the Anderson-Darling (AD) statistic, and the Weibull distribution's final selected model is the best according to the Kolmogorov-Smirnov (KS) and Cramér–von Mises (CvM) statistics. You can choose the fit statistic that is most suitable for your application and use it to identify the best severity distribution. Alternatively, if your goal is to estimate some loss distribution statistic, such as the value-at-risk (VaR), for a particular scenario, then instead of choosing one severity distribution, you might want to compute and compare the desired statistic for the final scale regression model of each severity distribution that wins according to at least one fit statistic. The **OUTPUT** statement is useful for such evaluation, because it helps you *score* each observation of the input data table that you specify in the **DATA=** option. In the context of PROC SEVSELECT, scoring an observation is equivalent to computing particular statistics for the severity distribution of the losses. The previous PROC SEVSELECT step specifies the following OUTPUT statement to write the scores for each observation to the mycas.Score output table:

```
output out=mycas.score copyvars=(carType gender education carSafety income)
      functions=(mean) quantiles=(cdf=0.5 0.975 names=(median var));
```

When you specify the scale regression model, the scale parameter of each severity distribution depends on the values of the regression effects in the final selected model of that distribution. For each observation in the input table, PROC SEVSELECT computes the estimate of the scale parameter and uses it to evaluate the scoring and quantile functions that you specify in the **FUNCTIONS=** and **QUANTILES=** options, respectively. The OUTPUT statement of this example specifies that three statistics be computed for the final model of each severity distribution: the *mean* scoring function and two quantile functions, $Q_{\text{dist}}(0.5, \hat{\Theta})$ and $Q_{\text{dist}}(0.975, \hat{\Theta})$, where $Q_{\text{dist}}(p, \Theta)$ denotes the value of the quantile function for a distribution *dist* that is evaluated at the CDF value of p for the estimated distribution parameters $\hat{\Theta}$. The variable that contains the estimate of the mean function is named *mean*. The variables that contain the quantile function estimate for $p=0.5$ and $p=0.975$ are named *median* and *var*, respectively; *var* denotes the value-at-risk (VaR). In order to compute a function F in the **FUNCTIONS=** option, for each severity distribution *dist*, the *dist_F* function must be available in the function library search path that you specify in the **CMPLIB=** system option. Also, in order to compute the quantile functions faster, it is recommended that the *dist_QUANTILE* function be available in the **CMPLIB=** search path. If PROC SEVSELECT does not find a quantile function's definition, it computes the quantile function by numerically inverting the cumulative distribution function (CDF). For each predefined distribution, such as the lognormal, Burr, and Weibull distributions of this example, the *dist_MEAN* and *dist_QUANTILE* functions are already defined and available to you in the function library, Sashelp.Svrtldist, that PROC SEVSELECT includes by default.

Output 15.7.6, Output 15.7.7, and Output 15.7.8 compare the estimates of the mean, median, and value-at-risk (*var*), respectively, of the three severity distributions for the first 10 observations that are fetched from the mycas.Score table. **NOTE:** The estimates are all available in the same table but are shown here separately for better comparison among different severity distributions. These results illustrate the following:

- The **OUT=** data table contains the variables that you specify in the **COPYVARS=** option. This helps you associate the scoring statistics with the values of the regression effects that decide the severity distribution's scale parameter.
- The estimates of a particular statistic vary by observation, because the scale parameter value depends on the values of the regression effects. You can interpret the estimate of the mean as the average loss that a particular policyholder will incur if he or she has the characteristics that are quantified in the values of the regression effects. For example, the fifth observation in Output 15.7.6 shows that a male

policyholder with a college degree who earns 0.70732 on a normalized scale and drives a sport utility vehicle (SUV) that has a normalized car safety rating of 0.8446 is expected to incur an average loss of 298 units if you use the lognormal distribution's selected scale regression model for estimation, or an average loss of 283 units if you use the Weibull distribution's selected scale regression model for estimation.

Output 15.7.6 Estimates of the Mean for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_MEAN	Burr_MEAN	Weibull_MEAN
1	Male	College	1.00896	SUV	0.85747	264.595	279.531	249.048
2	Male	AdvancedDegree	0.51942	Sedan	0.01410	90.959	93.358	88.674
3	Female	College	0.38431	Sedan	0.36374	135.151	137.595	133.730
4	Male	College	0.67508	Sedan	0.14417	94.361	97.079	90.119
5	Male	College	0.70732	SUV	0.84460	298.219	312.766	282.880
6	Male	High School	0.15000	Sedan	0.27925	144.016	148.594	137.050
7	Male	High School	0.47704	SUV	0.15692	269.407	278.323	259.974
8	Male	AdvancedDegree	0.77170	Sedan	0.61586	50.215	51.930	49.286
9	Female	AdvancedDegree	0.60157	Sedan	0.69361	77.799	79.648	78.331
10	Male	AdvancedDegree	0.90709	Sedan	0.00579	79.634	82.498	76.822

- The estimates of the median, which is a robust estimate of the central tendency, vary less across different severity distributions. So you might want to use the median loss to predict the average loss that a policyholder will incur.

Output 15.7.7 Estimates of the Median for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_MEDIAN	Burr_MEDIAN	Weibull_MEDIAN
1	Male	College	1.00896	SUV	0.85747	225.018	228.080	227.398
2	Male	AdvancedDegree	0.51942	Sedan	0.01410	77.354	76.174	80.966
3	Female	College	0.38431	Sedan	0.36374	114.936	112.269	122.104
4	Male	College	0.67508	Sedan	0.14417	80.247	79.211	82.285
5	Male	College	0.70732	SUV	0.84460	253.613	255.198	258.289
6	Male	High School	0.15000	Sedan	0.27925	122.475	121.244	125.136
7	Male	High School	0.47704	SUV	0.15692	229.111	227.094	237.373
8	Male	AdvancedDegree	0.77170	Sedan	0.61586	42.704	42.372	45.001
9	Female	AdvancedDegree	0.60157	Sedan	0.69361	66.162	64.988	71.522
10	Male	AdvancedDegree	0.90709	Sedan	0.00579	67.723	67.313	70.143

- The estimates of the value-at-risk (var), which depend on the heaviness of the tail of the distribution, vary significantly across severity distributions. For this example, the fitted Burr and Weibull distributions have the heaviest and lightest tails, respectively, among the three severity distributions. The VaR statistic is typically used to assess the worst-case loss that a policyholder might incur. You can interpret it as follows for the ninth observation in [Output 15.7.8](#): for a female policyholder with an advanced degree who earns 0.60157 and drives a sedan with a car safety rating of 0.69361, the probability that she will incur a loss greater than 228 units is 2.5% if you use the Burr distribution's scale regression model for estimation. You can add a CDF value of 0.995 to the CDF= suboption of the QUANTILES= option to get another value-at-risk estimate, which estimates a more extreme loss that a policyholder might incur with a probability of only 0.5%, but you might need to be prepared to cover that loss depending on the policy provisions.

Output 15.7.8 Estimates of the Value-at-Risk (VaR) for Final Models of All Severity Distributions

Obs	gender	education	income	carType	carSafety	Logn_VAR	Burr_VAR	Weibull_VAR
1	Male	College	1.00896	SUV	0.85747	686.684	801.100	585.528
2	Male	AdvancedDegree	0.51942	Sedan	0.01410	236.059	267.552	208.479
3	Female	College	0.38431	Sedan	0.36374	350.747	394.330	314.407
4	Male	College	0.67508	Sedan	0.14417	244.888	278.217	211.876
5	Male	College	0.70732	SUV	0.84460	773.946	896.348	665.069
6	Male	High School	0.15000	Sedan	0.27925	373.755	425.851	322.214
7	Male	High School	0.47704	SUV	0.15692	699.172	797.637	611.215
8	Male	AdvancedDegree	0.77170	Sedan	0.61586	130.319	148.824	115.874
9	Female	AdvancedDegree	0.60157	Sedan	0.69361	201.905	228.260	184.162
10	Male	AdvancedDegree	0.90709	Sedan	0.00579	206.668	236.428	180.612

PROC SEVSELECT always includes the intercept in all scale regression models, so the selection process never attempts to fit a model without an intercept. In addition to the intercept, you can force other regression effects to be included in each selected model by specifying the **INCLUDE=** option in the **SCALEMODEL** statement. The following SEVSELECT step illustrates this by forcing the **carSafety** effect into every selected model:

```
proc sevselect data=mycas.losses outest=mycas.est print=all;
  loss lossAmount / lt=deductible rc=limit;
  class carType gender education;
  scalemodel carType gender carType*gender carSafety income education
             income*gender carSafety*income / include=(carSafety);
  selection method=forward(stop=aicc) hierarchy=single stophorizon=2 details=all;
  dist logn weibull;
run;
```

The various options in the **SELECTION** statement do the following:

- The **METHOD=FORWARD** option specifies that the forward elimination method of selection be used.
- The **HIERARCHY=SINGLE** option instructs PROC SEVSELECT not to add an interaction effect to the model until all the main effects in the interaction are already in the model.
- The **STOPHORIZON=2** option specifies that the **STOP=** criterion, which is the corrected Akaike's information criterion (AICC), must worsen for two steps in order for a local extremum to be detected.
- The **DETAILS=ALL** option displays details of all the steps of the selection process.

The "Selection Information" table in [Output 15.7.9](#) summarizes the selection settings.

Output 15.7.9 Selection Method Settings for Forward Selection**The SEVSELECT Procedure****Logn Distribution**

Selection Information	
Selection Method	Forward
Select Criterion	SBC
Stop Criterion	AICC
Effect Hierarchy Enforced	Single
Stop Horizon	2

The selection summary for the lognormal distribution, shown in [Output 15.7.10](#), illustrates the following:

- The model at step 0 includes both the intercept and carSafety effects. The carSafety effect always stays in the model.
- Because of the HIERARCHY=SINGLE option, the carType*gender interaction effect does not enter the model until the model contains both main effects (carType and gender) that constitute it. Similarly, the carSafety*income interaction effect between two continuous effects also does not enter the model before both carSafety and income are in the model.
- The stop criterion keeps improving for several steps, so even the smaller stop horizon of 2 does not affect the selection process for the lognormal distribution. The selection stops when all effects are in the model.
- The final model is chosen by the optimal value of the STOP= criterion, because the CHOOSE= option is not specified. For the lognormal distribution, it is the model at step 6, which contains all regression effects except the income*gender effect. If you had specified the CHOOSE=SBC option, then the final model would be the model in step 4 that excludes carType*gender, carSafety*income, and income*gender effects, because it has the smallest value for the SBC criterion.

Output 15.7.10 Forward Selection Summary for the Lognormal Distribution**The SEVSELECT Procedure****Logn Distribution
Selection Details**

Selection Summary				
Step	Effect Entered	Number Effects In	AICC	SBC
0	Intercept	1		
	carSafety	2	35766.6044	35786.1512
1	carType	3	34230.4771	34256.5378
2	gender	4	33895.0222	33927.5962
3	income	5	33656.7655	33695.8519
4	education	6	33594.6989	33646.8076*
5	carType*gender	7	33591.2653	33649.8839
6	carSafety*income	8	33590.2351*	33655.3629
7	income*gender	9	33591.8039	33663.4401
* Optimal Value Of Criterion				

Selection stopped because all effects are in the model.

The model at step 6 is selected.

Selected Effects: Intercept carType gender carType*gender carSafety income education carSafety*income

The selection summary for the Weibull distribution is shown in [Output 15.7.11](#). It indicates that the selection stopped at the local minimum of the AICC statistic at step 5 because of the shorter stop horizon.

Output 15.7.11 Forward Selection Summary for the Weibull Distribution**The SEVSELECT Procedure****Weibull Distribution
Selection Details**

Selection Summary				
Step	Effect Entered	Number Effects In	AICC	SBC
0	Intercept	1		
	carSafety	2	36202.4284	36221.9751
1	carType	3	34560.1129	34586.1736
2	gender	4	34200.9756	34233.5496
3	income	5	33935.3440	33974.4303
4	education	6	33857.1156	33909.2243*
5	carType*gender	7	33856.5340*	33915.1526
6	carSafety*income	8	33857.1143	33922.2421
* Optimal Value Of Criterion				

Selection stopped at a local minimum of the AICC criterion.

Output 15.7.11 continued

The model at step 5 is selected.

Selected Effects: Intercept carType gender carType*gender carSafety income education

As the results in [Output 15.7.10](#) and [Output 15.7.11](#) show, although the SELECT= criterion is reported in the “Selection Summary” table, it does not decide the final selected model. Instead, the SELECT= criterion is used to choose the best effect to add or remove in each step. To illustrate that, [Output 15.7.12](#) shows the details of step 1 for the Weibull distribution. PROC SEVSELECT produces the step details because of the DETAILS=ALL option in the SELECTION statement. Step 1 is the step after the initial step (step 0), which fits the model with the intercept and carSafety effect that the INCLUDE= option forces in. The “Entry Candidates” table of step 1 shows the candidates that PROC SEVSELECT evaluates for entering the model. For each candidate, it shows the value of the SELECT= criterion, which is the SBC statistic, of the model that includes that candidate. The interaction effects carType*gender and carSafety*income are not present in this list of candidates, because of the constraint that the HIERARCHY=SINGLE option imposes. The model that includes the carType effect has the smallest SBC statistic, so PROC SEVSELECT selects it in step 1. The “Optimization Summary” table shows that PROC SEVSELECT uses the trust region optimization method, which is the default optimization method for fitting a model in each step. You can use the [SELECTNLOTECH=](#) option in the PROC SEVSELECT statement to specify a different optimization method to use during each step of the selection. The “Parameter Estimates” table shows that step 1’s model includes the best entry candidate, carType, and the carSafety effect, which entered the model in step 0. The estimate of the intercept effect is reflected in the estimate of the scale parameter Theta, as described in the section “[Reporting Estimates of Regression Parameters](#)” on page 881.

Output 15.7.12 Details of Step 1 of the Forward Selection Process for the Weibull Distribution

The SEVSELECT Procedure

Weibull Distribution

Forward Selection: Step 1

Effect Entered: carType

Entry Candidates		
Rank	Effect	SBC
1	carType	34586.1736
2	gender	36000.8378
3	education	36142.6117
4	income	36150.6917

Convergence Status	
Convergence criterion (GCONV=1E-8) satisfied.	

Optimization Summary	
Optimization Technique	Trust Region
Iterations	5
Function Calls	15
Log Likelihood	-17276.05243

Output 15.7.12 *continued*

Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	172.58489	4.92619	35.03	<.0001
Tau	1	1.39719	0.02266	61.65	<.0001
carType SUV	1	1.28993	0.02919	44.18	<.0001
carType Sedan	0	0	.	.	.
carSafety	1	-0.85372	0.04522	-18.88	<.0001

The fit summary of the final model for the Weibull distribution is shown in [Output 15.7.13](#). It shows that PROC SEVSELECT uses the trust region optimization method to estimate the parameters of the final model. The “Parameter Estimates” table shows that some regression parameters are not significant at the usual 95% confidence level. None of the selection criteria that PROC SEVSELECT supports depend directly on the p -values of regression effects. So a statistically insignificant effect might be included in the final model, as this example illustrates. If you do not want such effects in the final model, you can modify the selection method settings to search for a different model that contains all statistically significant effects. Alternatively, you can use your domain knowledge to decide which effects you want to retain from the selected model and then refit the model by appropriately modifying the SCALEMODEL statement.

Output 15.7.13 Selected Scale Regression Model for the Weibull Distribution

Convergence Status					
Convergence criterion (GCONV=1E-8) satisfied.					
Optimization Summary					
Optimization Technique	Trust Region				
Iterations	5				
Function Calls	14				
Log Likelihood	-16919.24895				
Parameter Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Approx Pr > t
Theta	1	177.53326	5.63376	31.51	<.0001
Tau	1	1.63078	0.02621	62.23	<.0001
carType SUV	1	1.27004	0.03578	35.50	<.0001
carType Sedan	0	0	.	.	.
gender Female	1	0.45939	0.02703	16.99	<.0001
gender Male	0	0	.	.	.
carType SUV * gender Female	1	0.08009	0.04981	1.61	0.1079
carType SUV * gender Male	0	0	.	.	.
carType Sedan * gender Female	0	0	.	.	.
carType Sedan * gender Male	0	0	.	.	.
carSafety	1	-0.82500	0.03881	-21.26	<.0001
income	1	-0.38016	0.02832	-13.42	<.0001
education AdvancedDegree	1	-0.32602	0.03563	-9.15	<.0001
education College	1	-0.04246	0.02748	-1.55	0.1224
education High School	0	0	.	.	.

Example 15.8: Estimating Parameters Using the Cramér–von Mises Estimator

The SEVSELECT procedure enables you to estimate model parameters by minimizing your own objective function. This example illustrates how you can use PROC SEVSELECT to implement the Cramér–von Mises estimator. Let $F(y_i; \Theta)$ denote the estimate of CDF at y_i for a distribution with parameters Θ , and let $F_n(y_i)$ denote the empirical estimate of CDF (EDF) at y_i that is computed from a sample y_i , $1 \leq i \leq N$. Then, the Cramér–von Mises estimator of the parameters is defined as

$$\hat{\Theta} = \arg \min_{\Theta} \sum_{i=1}^N (F(y_i; \Theta) - F_n(y_i))^2$$

This estimator belongs to the class of minimum distance estimators. It attempts to estimate the parameters such that the squared distance between the CDF and EDF estimates is minimized.

The following PROC SEVSELECT step uses the Cramér–von Mises estimator to fit four candidate distribution models, including the LOGNGPD mixed-tail distribution model that is defined in “[Example 15.2: Defining a Model for Mixed-Tail Distributions](#)” on page 940. The input sample is the same one that is used in that example.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*----- Fit LOGNGPD model with PROC SEVSELECT by using -----
----- the Cramer-von Mises minimum distance estimator -----*/
proc sevselect data=mycas.testmixdist objective=cvmobj print=all;
  loss y;
  dist logngpd burr logn gpd;

  * Cramer-von Mises estimator (minimizes the distance *
  * between parametric and nonparametric estimates)   *;
  cvmobj = (_cdf_(y) - _edf_(y))**2;
run;
```

The OBJECTIVE= option in the PROC SEVSELECT statement specifies that the objective function cvmobj should be minimized. The programming statements compute the contribution of each observation in the input data table to the objective function cvmobj. The use of the keyword functions _CDF_ and _EDF_ makes the program applicable to all the distributions. When PROC SEVSELECT runs on a CAS server that has more than one worker node, each worker node evaluates the _EDF_ function by using a local sample of the input data.

Some of the key results that PROC SEVSELECT produces are shown in [Output 15.8.1](#). The “Model Selection” table indicates that all models converged. When you specify a custom objective function, the default selection criterion is the value of the custom objective function. The “All Fit Statistics” table indicates that LOGNGPD is the best distribution according to all the statistics of fit. Comparing the fit statistics of [Output 15.8.1](#) with those of [Output 15.2.1](#) indicates that the use of the Cramér–von Mises estimator has resulted in smaller values for all the EDF-based statistics of fit for all the models, which is expected from a minimum distance estimator.

Output 15.8.1 Summary of Cramér–von Mises Estimation**The SEVSELECT Procedure**

Model Selection								
Distribution	Converged	cvmobj Selected						
Logngpd	Yes	0.28044	Yes					
Burr	Yes	0.37199	No					
Logn	Yes	0.31866	No					
Gpd	Yes	36.01725	No					

All Fit Statistics								
Distribution	cvmobj	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Logngpd	0.28044*	3661*	3671*	3671*	3696*	0.76415*	0.72304*	0.09517*
Burr	0.37199	3724	3730	3730	3745	0.85781	1.11785	0.12601
Logn	0.31866	3907	3911	3911	3921	0.82393	1.08382	0.10704
Gpd	36.01725	5400	5404	5404	5414	5.78875	63.08168	12.02024

Asterisk (*) denotes the best model in the column.

Example 15.9: Predicting Mean and Value-at-Risk by Using Scoring Functions

If you work in the risk management department of an insurance company or a bank, then one of your primary applications of severity loss distribution models is to predict the value-at-risk (VaR) so that there is a very low probability of experiencing a loss value greater than the VaR. The probability level at which VaR is measured is prescribed by industry regulations such as Basel III and Solvency II. The VaR level is usually specified in terms of $(1 - \alpha)$, where $\alpha \in (0, 1)$ is the probability that a loss value exceeds the VaR. Typical VaR levels are 0.95, 0.975, and 0.995.

In addition to predicting the VaR, which is regarded as an estimate of the worst-case loss, businesses are often interested in predicting the average loss by estimating either the mean or median of the distribution.

The estimation of the mean and VaR combined with the scale regression model is a very potent tool for analyzing worst-case and average losses for various scenarios. For example, if the regressors that are used in a scale regression model represent some key macroeconomic and operational indicators, which are widely referred to as key risk indicators (KRIs), then you can analyze the VaR and mean loss estimates over various values for the KRIs to get a more comprehensive picture of the risk profile of your organization across various market and internal conditions.

This example illustrates the use of scoring functions to simplify the process of predicting the mean and VaR of scale regression models.

To compute the mean, you need to ensure that the function to compute the mean of a distribution is available in the function library. If you define and fit your own distribution and you want to compute its mean, then you need to use the FCMP procedure to define that function, and you need to use the CMPLIB= system option to specify the location of that function. For your convenience, the *dist*_MEAN function (which computes the mean of the *dist* distribution) is already defined in the library Sashelp.Svrtldist for each of the 10 predefined distributions. The following statements display the definitions of MEAN functions of all distributions. Note that the MEAN functions of the Burr, Pareto, and generalized Pareto distributions check the existence of the first moment for specified parameter values.

```

/*----- Definitions distribution functions that compute the mean -----*/
proc fcmp library=sashelp.svrtldist outlib=work.means.scalemod;
  function BURR_MEAN(x, Theta, Alpha, Gamma);
    if not(Alpha * Gamma > 1) then
      return (.); /* first moment does not exist */
    return (Theta*gamma(1 + 1/Gamma)*gamma(Alpha - 1/Gamma)/gamma(Alpha));
  endsub;
  function EXP_MEAN(x, Theta);
    return (Theta);
  endsub;
  function GAMMA_MEAN(x, Theta, Alpha);
    return (Theta*Alpha);
  endsub;
  function GPD_MEAN(x, Theta, Xi);
    if not(Xi < 1) then
      return (.); /* first moment does not exist */
    return (Theta/(1 - Xi));
  endsub;
  function IGAUSS_MEAN(x, Theta, Alpha);
    return (Theta);
  endsub;
  function LOGN_MEAN(x, Mu, Sigma);
    return (exp(Mu + Sigma*Sigma/2.0));
  endsub;
  function PARETO_MEAN(x, Theta, Alpha);
    if not(Alpha > 1) then
      return (.); /* first moment does not exist */
    return (Theta/(Alpha - 1));
  endsub;
  function STWEEDIE_MEAN(x, Theta, Lambda, P);
    return (Theta* Lambda * (2 - P) / (P - 1));
  endsub;
  function TWEEDIE_MEAN(x, P, Mu, Phi);
    return (Mu);
  endsub;
  function WEIBULL_MEAN(x, Theta, Tau);
    return (Theta*gamma(1 + 1/Tau));
  endsub;
quit;

```

For your further convenience, the *dist_QUANTILE* function (which computes the quantile of the *dist* distribution) is also defined in the Sashelp.Svrtldist library for each of the 10 predefined distributions. Because the MEAN and QUANTILE functions satisfy the definition of a distribution function as described in the section “[Formal Description](#)” on page 914, PROC SEVSELECT can create scoring functions for them.

To illustrate the creation of scoring functions, let the following SAS statements simulate the input data set Work.Test_sev9 and load it into a CAS server:

```

/*----- Simulate a lognormal sample -----*/
data testSevscore(keep=y x1-x5
  label='A Lognormal Sample Affected by Regressors');
  array x{*} x1-x5;
  array b{6} _TEMPORARY_ (1 0.75 3 -1 0.25 5);

```

```

call streaminit(45678);
label y='Response Influenced by Regressors';
Sigma = 0.25;
do n = 1 to 500;
    Mu = b(1); /* log of base value of scale */
    do i = 1 to dim(x);
        x(i) = rand('NORMAL');
        Mu = Mu + b(i+1) * x(i);
    end;
    y = exp(Mu) * rand('LOGNORMAL')**Sigma;
    output;
end;
run;

/*----- Load data into the CAS server -----*/
data mycas.testSevscore;
    set testSevscore;
run;

```

The following PROC SEVSELECT step fits all regression-friendly predefined distributions to the data in the data table mycas.TestSevscore and generates the scoring functions for the MEAN, QUANTILE, and other distribution functions:

```

/*----- Fit all distributions and generate scoring functions -----*/
proc sevselect data=mycas.testSevscore outest=mycas.est print=all;
    loss y;
    scalemodel x1-x5;
    dist _predefined_ stweedie;
    nloptions tech=quanew;
    outscorelib outlib=scorefuncs commonpackage;
run;

```

The OUTLIB= option in the OUTSCORELIB statement requests that the scoring functions be written to the library Work.Scorefuncs, and the COMMONPACKAGE option in the OUTSCORELIB statement requests that all the functions be written to the same package. Upon completion, PROC SEVSELECT sets the CMPLIB= system option to the following value:

```
(sashelp.svrtldist work.scorefuncs)
```

PROC SEVSELECT detects all the distribution functions that are available in the current CMPLIB= search path (which always includes the Sashelp.Svrtldist library) for the distributions that you specify in the DIST statement, and it creates the corresponding scoring functions. You can define any distribution function that has the desired signature to compute an estimate of your choice, include its library in the CMPLIB= system option, and then specify the OUTSCORELIB statement to generate the corresponding scoring functions. Specifying the COMMONPACKAGE option in the OUTSCORELIB statement causes the name of the scoring function to take the form SEV_<function-suffix>_dist. If you do not specify the COMMONPACKAGE option, PROC SEVSELECT creates a scoring function named SEV_<function-suffix> in a package named dist. You can invoke functions from a specific package only inside the FCMP procedure. If you want to invoke the scoring functions from a DATA step, then it is recommended that you specify the COMMONPACKAGE option when you specify multiple distributions in the DIST statement.

You can get an idea of the structure of the scoring functions that PROC SEVSELECT defines on your behalf by examining the following PROC FCMP code that the preceding PROC SEVSELECT step creates and submits to define the scoring functions SEV_MEAN_LOGN and SEV_QUANTILE_IGAUSS:

```

proc fcmp library=(sashelp.svrtdist) outlib=work.scorefuncs.sevfit;
  function SEV_MEAN_LOGN(y, x{*});
    _logscale_=0;
    _logscale_ = _logscale_ + ( 7.64722238665070E-01 * x{1});
    _logscale_ = _logscale_ + ( 2.99209532114045E+00 * x{2});
    _logscale_ = _logscale_ + (-1.00788910539257E+00 * x{3});
    _logscale_ = _logscale_ + ( 2.58883411376730E-01 * x{4});
    _logscale_ = _logscale_ + ( 5.00927459151049E+00 * x{5});
    _logscale_ = _logscale_ + ( 9.95078801087240E-01);
    return (LOGN_MEAN(y, _logscale_, 2.31592996273020E-01));
  endsub;

  function SEV_QUANTILE_IGAUSS(y, x{*});
    _logscale_=0;
    _logscale_ = _logscale_ + ( 7.64581739155730E-01 * x{1});
    _logscale_ = _logscale_ + ( 2.99159055149487E+00 * x{2});
    _logscale_ = _logscale_ + (-1.00793496650702E+00 * x{3});
    _logscale_ = _logscale_ + ( 2.58870461417330E-01 * x{4});
    _logscale_ = _logscale_ + ( 5.00996884583256E+00 * x{5});
    _scale_ = 2.77854870199624E+00 * exp(_logscale_);
    return (IGAUSS_QUANTILE(y, _scale_, 1.815112308761660E+01));
  endsub;
quit;

```

PROC SEVSELECT creates and submits similar code to define several other scoring functions, such as `SEV_PDF_dist` and `SEV_SDF_dist`, for each distribution *dist* that you specify in the DIST statement.

To illustrate the use of scoring functions, let the following DATA step simulate the scoring data set `Work.RegInput`, where the values of regressors in each observation define one scenario:

```

/*--- Simulate scoring data ---*/
data reginput(keep=x1-x5);
  array x{*} x1-x5;
  call streaminit(9041);
  do n=1 to 10;
    do i = 1 to dim(x);
      x(i) = rand('NORMAL');
    end;
    output;
  end;
run;

```

The “All Fit Statistics” table in [Output 15.9.1](#) shows that the lognormal distribution’s scale model is the best and the inverse Gaussian’s scale model is a close second according to the likelihood-based statistics. So the following DATA step uses the scoring functions to compute the mean and VaR (at the 97.5% level) of the scale models of lognormal and inverse Gaussian distributions for each of the scenarios in the `Work.RegInput` data set:

```

/*--- Set VaR level ---*/
%let varLevel=0.975;

/*--- Compute scores (mean and var) for the ---
--- scoring data by using the scoring functions ---*/

```

```

data scores;
  array x{*} x1-x5;
  set reginput;

  igauss_mean = sev_mean_igauss(., x);
  igauss_var  = sev_quantile_igauss(&varLevel, x);
  logn_mean   = sev_mean_logn(., x);
  logn_var    = sev_quantile_logn(&varLevel, x);
run;

```

Output 15.9.1 Comparison of Fitted Scale Models for Mean and VaR Illustration**The SEVSELECT Procedure**

Distribution	All Fit Statistics						
	-2 Log Likelihood	AIC	AICC	SBC	KS	AD	CvM
Stweddie	449.72754	465.72754	466.02081	499.44440	6.13073	1972	12.45028
Burr	451.42238	467.42238	467.71565	501.13924	6.08421	1478	12.47837
Exp	1515	1527	1527	1552	5.18217	696.65946*	7.97923
Gamma	448.28222	462.28222	462.50986	491.78448	6.13516	2014	12.46485
Igauss	444.44512	458.44512	458.67276	487.94738	6.15964	2073	12.45994
Logn	444.43670*	458.43670*	458.66434*	487.93895*	6.15245	2042	12.47822
Pareto	1515	1529	1530	1559	5.21366	701.36259	8.00820
Gpd	1515	1529	1529	1559	5.18217*	696.66044	7.97923*
Weibull	527.28676	541.28676	541.51440	570.78902	6.06493	1635	12.13031

Asterisk (*) denotes the best model in the column.

The following DATA step accomplishes the same task by reading the parameter estimates in the OUTEST= data table that is created by the same PROC SEVSELECT step that created the scoring functions:

```

/*--- Bring the estimates data back to the client ---*/
data est;
  set mycas.est;
run;

/*--- Compute scores (mean and var) for the ---
--- scoring data by using the OUTEST= data set ---*/
data scoresWithOutest(keep=x1-x5 igauss_mean igauss_var logn_mean logn_var);
  array _x{*} x1-x5;
  array _xparmIgauss_{5} _temporary_;
  array _xparmLogn_{5} _temporary_;

  retain _Theta0_ Alpha0;
  retain _Mu0_ Sigma0;
  *--- read parameter estimates for igauss and logn models ---*;
  if (_n_ = 1) then do;
    set est(where=(upcase(_MODEL_)='IGAUSS' and _TYPE_='EST'));
    _Theta0_ = Theta; Alpha0 = Alpha;
    do _i_=1 to dim(_x_);
      if (_x_(_i_) = .R) then _xparmIgauss_(_i_) = 0;
      else _xparmIgauss_(_i_) = _x_(_i_);
    end;
  end;

```

```

set est(where=(upcase(_MODEL_='LOGN' and _TYPE_='EST')));
_Mu0_ = Mu; Sigma0 = Sigma;
do _i_=1 to dim(_x_);
    if (_x_(_i_) = .R) then _xparmLogn_(_i_) = 0;
    else _xparmLogn_(_i_) = _x_(_i_);
end;
end;

set reginput;

*--- predict mean and VaR for inverse Gaussian ---*;
* first compute X'*beta for inverse Gaussian *;
_xbeta_ = 0.0;
do _i_ = 1 to dim(_x_);
    _xbeta_ = _xbeta_ + _xparmIgauss_(_i_) * _x_(_i_);
end;
* now compute scale for inverse Gaussian *;
_SCALE_ = _Theta0_ * exp(_xbeta_);
igauss_mean = igauss_mean(., _SCALE_, Alpha0);
igauss_var = igauss_quantile(&varLevel, _SCALE_, Alpha0);

*--- predict mean and VaR for lognormal ---*;
* first compute X'*beta for lognormal*;
_xbeta_ = 0.0;
do _i_ = 1 to dim(_x_);
    _xbeta_ = _xbeta_ + _xparmLogn_(_i_) * _x_(_i_);
end;
* now compute Mu=log(scale) for lognormal *;
_MU_ = _Mu0_ + _xbeta_;
logn_mean = logn_mean(., _MU_, Sigma0);
logn_var = logn_quantile(&varLevel, _MU_, Sigma0);
run;

```

The “Values Comparison Summary” table in [Output 15.9.2](#) shows that the difference between the estimates that are produced by both methods is within the acceptable machine precision. However, if you compare the DATA step complexity of the methods, you clearly see that the method that uses the scoring functions is much easier because it saves a lot of programming effort. Further, new distribution functions are automatically discovered and converted to scoring functions by PROC SEVSELECT. That enables you to focus your efforts on writing the distribution function that computes your desired score, which needs to be done only once. Then, you can create and use the corresponding scoring functions multiple times with much less effort.

Output 15.9.2 Comparison of Mean and VaR Estimates of Two Scoring Methods

The COMPARE Procedure
 Comparison of WORK.SCORESWITHOUTEST with WORK.SCORES
 (Method=RELATIVE(0.0222), Criterion=1.0E-12)

NOTE: All values compared are within the equality criterion used. However, 38 of the values compared are not exactly equal.

References

- Burr, I. W. (1942). "Cumulative Frequency Functions." *Annals of Mathematical Statistics* 13:215–232.
- D'Agostino, R. B., and Stephens, M., eds. (1986). *Goodness-of-Fit Techniques*. New York: Marcel Dekker.
- Danielsson, J., de Haan, L., Peng, L., and de Vries, C. G. (2001). "Using a Bootstrap Method to Choose the Sample Fraction in Tail Index Estimation." *Journal of Multivariate Analysis* 76:226–248.
- Dunn, P. K., and Smyth, G. K. (2005). "Series Evaluation of Tweedie Exponential Dispersion Model Densities." *Statistics and Computing* 15:267–280.
- Frydman, H. (1994). "A Note on Nonparametric Estimation of the Distribution Function from Interval-Censored and Truncated Observations." *Journal of the Royal Statistical Society, Series B* 56:71–74.
- Gentleman, R., and Geyer, C. J. (1994). "Maximum Likelihood for Interval Censored Data: Consistency and Computation." *Biometrika* 81:618–623.
- Greenwood, M. (1926). "The Natural Duration of Cancer." In *Reports of Public Health and Related Subjects*, vol. 33, 1–26. London: Her Majesty's Stationery Office.
- Hill, B. M. (1975). "A Simple General Approach to Inference about the Tail of a Distribution." *Annals of Statistics* 3:1163–1173.
- Jørgensen, B. (1987). "Exponential Dispersion Models." *Journal of the Royal Statistical Society, Series B* 49:127–162. With discussion.
- Kaplan, E. L., and Meier, P. (1958). "Nonparametric Estimation from Incomplete Observations." *Journal of the American Statistical Association* 53:457–481.
- Klein, J. P., and Moeschberger, M. L. (1997). *Survival Analysis: Techniques for Censored and Truncated Data*. New York: Springer-Verlag.
- Klugman, S. A., Panjer, H. H., and Willmot, G. E. (1998). *Loss Models: From Data to Decisions*. New York: John Wiley & Sons.
- Koziol, J. A., and Green, S. B. (1976). "A Cramér–von Mises Statistic for Randomly Censored Data." *Biometrika* 63:466–474.
- Lai, T. L., and Ying, Z. (1991). "Estimating a Distribution Function with Truncated and Censored Data." *Annals of Statistics* 19:417–442.
- Lynden-Bell, D. (1971). "A Method of Allowing for Known Observational Selection in Small Samples Applied to 3CR Quasars." *Monthly Notices of the Royal Astronomical Society* 155:95–118.
- Rodriguez, R. N., and Balan, T. E. (2006). "Creating Statistical Graphics in SAS 9.2: What Every Statistical User Should Know." In *Proceedings of the Thirty-First Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. <http://www2.sas.com/proceedings/sugi31/192-31.pdf>.
- Turnbull, B. W. (1976). "The Empirical Distribution Function with Arbitrarily Grouped, Censored, and Truncated Data." *Journal of the Royal Statistical Society, Series B* 38:290–295.

Tweedie, M. C. K. (1984). "An Index Which Distinguishes between Some Important Exponential Families." In *Statistics: Applications and New Directions—Proceedings of the Indian Statistical Institute Golden Jubilee International Conference*, edited by J. K. Ghosh and J. Roy, 579–604. Calcutta: Indian Statistical Institute.

Chapter 16

The TSINFO Procedure

Contents

Overview: TSINFO Procedure	985
PROC TSINFO Compared with the TIMEID Procedure	986
Using CAS Sessions and CAS Engine Librefs	986
Getting Started: TSINFO Procedure	987
Syntax: TSINFO Procedure	988
Functional Summary	988
PROC TSINFO Statement	988
BY Statement	989
ID Statement	989
Details: TSINFO Procedure	990
Time ID Diagnostics	990
Inferring Time Intervals and Alignments	990
Printed Tabular Output	991
Examples: TSINFO Procedure	992
Example 16.1: Examining a Weekly Time ID Variable and Inferring a Date Interval . .	992
Example 16.2: Examining Multiple BY Groups	992

Overview: TSINFO Procedure

The TSINFO procedure evaluates a variable in an input data table for its suitability as a time ID variable in SAS procedures and solutions that are used for time series analysis. PROC TSINFO assesses how well a time interval specification fits SAS date values, SAS datetime values, or observation numbers that are used to index a time series. The time interval used in this analysis can be either specified explicitly as input or inferred based on values of the time ID variable. The TSINFO procedure produces diagnostic information in the form of data tables and ODS tables. These diagnostic results summarize characteristics of the time ID variable that can help determine its use as an index in other time series procedures and solutions.

PROC TSINFO is intended for use as a tool to either identify the time interval of a variable or prepare problematic data sets for use in subsequent time series analyses. In particular, this procedure can be used to investigate inconsistencies between time ID values and the ID statement options that are used in other SAS procedures and solutions.

PROC TSINFO Compared with the TIMEID Procedure

The TSINFO procedure is the next generation of the TIMEID procedure (a SAS/ETS procedure) for time series information analysis. PROC TSINFO was developed specifically for SAS Viya. The syntax of PROC TSINFO is similar to the syntax of PROC TIMEID, from which it borrows the underlying methodology and goals. PROC TSINFO is designed to run on a cluster of machines that distribute the data and computations to multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Services (CAS) server.

Table 16.1 shows the differences between the TSINFO and TIMEID procedures.

Table 16.1 PROC TSINFO versus PROC TIMEID

Feature	TSINFO	TIMEID
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable bottom-up

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Getting Started: TSINFO Procedure

When a data table contains a time ID variable that has corrupted, missing, or duplicate values, PROC TSINFO can help isolate and identify these problematic observations. For larger data tables whose quality is unknown, it can be useful to get a general overview of the relative number of observations that have problematic time ID values. When prior knowledge of the time interval that separates observations is incomplete, PROC TSINFO can be used to infer the interval.

The following example uses the Sashelp.Air data set to illustrate how to analyze some time series information.

The following DATA step loads the Air data set from the Sashelp directory to a table named mycas.air in your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
  set sashelp.air;
do class=1 to 10;
  output;
end;
run;
```

The following statements use the TSINFO procedure to infer the interval of the time series:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval1
           nthreads=2;
  id date align=BEGIN;
  by class;
run;
```

PROC TSINFO requires the following inputs:

- The input table, specified in the DATA= option in the PROC statement
- BY variables (class), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSINFO Procedure

The following statements are available in the TSINFO procedure:

```
PROC TSINFO < options > ;
BY variables ;
ID variable < options > ;
```

The PROC TSINFO and ID statements are required. The following sections describe the PROC TSINFO statement and then describe the other statements in alphabetical order.

Functional Summary

Table 16.2 summarizes the statements and options that are used with the TSINFO procedure.

Table 16.2 PROC TSINFO Functional Summary

Description	Statement	Option
Data Options		
Specifies the input data table	PROC TSINFO	DATA=
Specifies the detailed output interval data table	PROC TSINFO	OUTINTERVALDETAILS=
Specifies the number of threads to be used in the procedure	PROC TSINFO	NTHREADS=
Time ID Options		
Specifies the alignment of time ID values	ID	ALIGN=
Specifies the ending time ID value	ID	END=
Specifies the starting time ID value	ID	START=

PROC TSINFO Statement

```
PROC TSINFO options ;
```

The PROC TSINFO statement invokes the TSINFO procedure.

You must specify the following data tables:

DATA=*CAS-libref.data-table*

specifies the SAS input data table that contains the input data for the procedure. If the DATA= option is not specified, the most recently created SAS data set is used. *CAS-libref.data-set* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

OUTINTERVALDETAILS =CAS-libref.data-table

names the output data table to contain the time ID interval information for each BY group in the DATA= data set. For more information, see the section “[OUTINTERVALDETAILS= Data Set](#)” on page 991. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

You can also specify the following *option*:

NTHREADS=number

specifies the number of threads to use in the computation.

By default, NTHREADS=1.

BY Statement

BY variables ;

You can use a BY statement to obtain separate analyses for groups of observations that are defined by the *variables*, which can be either character or numeric.

When a BY statement is included, the procedure expects the input data to be sorted in the order of the BY variables.

ID Statement

ID variable </options> ;

The ID statement names a numeric *variable* that identifies observations in the input and output data. The ID variable values are assumed to be SAS date or datetime values.

You can specify the following *options* to determine how to space the time ID values and align them relative to a SAS date or datetime interval:

ALIGN=BEGINNING | MIDDLE | ENDING | INFER

specifies the alignment of the identifying SAS date or datetime that is used to represent intervals. The value of the ALIGN= option is used in the analysis of the time ID variable. You can specify the following values:

BEGINNING BEG B	uses the beginning date in the interval as the identifying date for the interval.
MIDDLE MID M	uses the middle date in the interval as the identifying date for the interval.
ENDING END E	uses the end date in the interval as the identifying date for the interval.
INFER	infers that the alignment of values within time intervals from the time ID values.

By default, ALIGN=BEGIN.

END=*value*

specifies a SAS date, datetime, or time value that represents when the SAS date or datetime values end. If the largest *variable* value is less than *value*, then this option has no effect.

START=*value*

specifies a SAS date, datetime, or time value that represents the *variable* value at which the SAS date or datetime values begins.

Details: TSINFO Procedure

Time ID Diagnostics

For a specified time interval, PROC TSINFO decomposes the raw time ID values in an input data table into the following three quantities, whose values are represented by nonnegative integers at each unique time ID value in the input series:

intervalcounts	the number of observations that share each time interval in the data set.
offsets	the numerical difference between a time ID value and the aligned value for that time interval. The unit of measure that is used to express this distance is days for date values and seconds for datetime values. The offset is computed for each time ID value, t_i , by using the following SAS expression:

$$\text{offset}_i = t_i - \text{INTNX}(\text{interval}, t_i, 0, \text{alignment})$$

spans	the number of intervals between each time ID value and the previous time ID value. The spans value is equivalent to the number returned by the following SAS expression:
-------	--

$$\text{spans}_i = \text{INTCK}(\text{interval}, t_{i-1}, t_i)$$

Inferring Time Intervals and Alignments

A time interval is inferred from the time ID values in the input data. The technique that is used to infer a time interval involves searching for the interval that fits the greatest number of time ID values. First, time ID values are sampled from the input data to generate a set of candidate intervals. Then the candidate interval that is consistent with greatest number of time ID values is chosen to represent the time series.

When the **ALIGN=INFER** option is specified, the convention that is used to specify time interval alignment is inferred from the time ID variable values by using a similar technique. When both the time interval and its alignment are to be inferred, each of the possible alignments (**BEGIN**, **MIDDLE**, and **END**) is considered in the search. Precedence in the search is given to intervals that use the **BEGIN** alignment.

Data Table Output

The TSINFO procedure creates an ODS output data table. This data table contains the variables that are specified in the BY statement along with variables that characterize the time ID values. The information in this data table summarizes time ID diagnostic information across all BY groups in the table that is specified in the DATA= option.

OUTINTERVALDETAILS= Data Set

The OUTINTERVALDETAILS= data set contains statistics about the time interval that is specified in the ID statement or inferred from the time ID values for each BY group. The following variables represent these statistics:

TIMEID	time ID variable name
START	starting time ID interval
END	ending time ID interval
INTERVAL	time interval that was specified or is recommended
INTNAME	time interval base name that was specified or is recommended
MULTIPLIER	time interval multiplier that was specified or is recommended
SHIFT_INDEX	time interval shift index that was specified or is recommended
ALIGNMENT	time interval alignment that was specified or is recommended
SEASONALITY	seasonality determined from the specified or recommended time interval
FORMAT	format of the time ID variable

The START and END variables are reported using the interval and alignment values that are either specified in the ID statement or inferred from the time ID values.

Printed Tabular Output

The TSINFO procedure produces printed output by using the Output Delivery System (ODS). By default, the procedure produces the overall time series information table.

Table 16.3 ODS Tables Produced in PROC TSINFO

ODS Name	Description
DataSet	Information about the input data table
Decomposition	Time ID counts, offsets, and spans
Interval	Information about the time interval
IntervalCountsComponent	Frequency distribution of interval counts
Align	Time ID alignment information

Table 16.3 *continued*

ODS Table Name	Description
Seasonality	Summary of the number of valid observations

Examples: TSINFO Procedure

Example 16.1: Examining a Weekly Time ID Variable and Inferring a Date Interval

This example illustrates how a time ID variable can be inferred from data when a sufficient number of observations are present. The following DATA step loads the Air data set onto the CAS server. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
  set sashelp.air;
run;
```

The following TSINFO procedure statements generate an ODS display of the time series that characterizes interval counts, alignment, and seasonality in the time ID variable:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;
  id date align=B;
run;
```

There are 144 observations in the mycas.Air data table. The 144 observations are enough to determine that the Date time ID variable is represented by the MONTH interval.

Example 16.2: Examining Multiple BY Groups

This example illustrates how a time ID variable can be examined independently over each BY group and summarized over all observations in the data table that is specified in the DATA= option.

```
data mycas.air(promote=yes);
  set sashelp.air;
  do class=1 to 10;
    output;
  end;
run;
```

The following TSINFO procedure statements generate data tables that summarize data that have four BY groups:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;  
  id date align=B;  
  by class;  
run;
```

The summarized information shows that BY groups 1 to 10 in the mycas.interval2 data table contain the interval fit information of the time ID values and shows the number of counts, alignment, and seasonality for each by group. All BY groups conform exactly to the MONTH interval. The information can be useful in diagnosing time series that contain many time intervals.

Chapter 17

The TSMODEL Procedure

Contents

Overview: TSMODEL Procedure	996
Comparison of the TSMODEL and TIMESERIES Procedures	996
Comparison of the TSMODEL and TIMEDATA Procedures	996
Using CAS Sessions and CAS Engine Librefs	999
Getting Started: TSMODEL Procedure	1000
Syntax: TSMODEL Procedure	1002
Functional Summary	1003
PROC TSMODEL Statement	1004
BY Statement	1009
ID Statement	1009
INSCALARS Statement	1017
OUTARRAYS Statement	1017
OUTSCALARS Statement	1017
PRINT Statement	1018
VAR Statement	1018
REQUIRE Statement	1019
SUBMIT Statement	1019
ENDSUBMIT Statement	1020
SAS Programming Statements	1021
Details: TSMODEL Procedure	1021
Accumulation	1021
Missing Value Interpretation	1023
Summary Statistics	1023
SAS Programming Statements	1023
Predefined Symbols	1023
Auxiliary Tables	1024
Table Output	1028
Printed Output	1030
Examples: TSMODEL Procedure	1031
Example 17.1: Accumulating Transactional Data into Time Series Data	1031
Example 17.2: User-Defined Program Statements	1033
Example 17.3: Using Auxiliary Tables	1034
References	1037

Overview: TSMODEL Procedure

The TSMODEL procedure is a SAS Viya procedure that executes user-defined programs on time series data. The TSMODEL procedure analyzes timestamped transactional data with respect to time and accumulates the data into a time series format.

The TSMODEL procedure forms time series from input timestamped transactional data and writes the accumulated time series variables to an output table. Time series are delineated based on the distinct values of the variables that are listed in the BY statement.

Timestamped transactional data are recorded at no fixed interval. Analysts often want to use time series analysis techniques that require fixed-time intervals. Therefore, the transactional data must be accumulated to form a fixed-interval time series, such as daily, weekly, or monthly.

The TSMODEL procedure forms time series vectors from timestamped data and then provides these vectors as array variables for subsequent processing by your program statements. Your program statements are processed independently for each BY group. The TSMODEL procedure is similar to the SAS DATA step for time series data. The SAS DATA step processes data by each row, whereas the TSMODEL procedure processes time series vectors for the BY groups.

Because PROC TSMODEL runs on SAS Cloud Analytic Services (CAS), it can process the time series data in parallel. Time series for the BY groups are divided across the nodes of the CAS session, and then threads are used on each node to process the node's BY groups concurrently.

All results of the time series analysis can be stored to CAS tables.

Comparison of the TSMODEL and TIMESERIES Procedures

The TSMODEL procedure has similarities to the TIMESERIES procedure in SAS/ETS. The TIMESERIES procedure enables you to perform a variety of standard time series analysis techniques with its various statements, whereas the TSMODEL procedure enables you to define your own analyses via user-defined program statements that you include in the procedure's statement block. The TSMODEL procedure provides no built-in time series analysis capabilities. You must provide a program to analyze the time series data. For more information about PROC TIMESERIES, see *SAS/ETS User's Guide*.

Comparison of the TSMODEL and TIMEDATA Procedures

The syntax of the TSMODEL procedure is similar to the syntax of the TIMEDATA procedure in SAS/ETS software. For more information about PROC TIMEDATA, see *SAS/ETS User's Guide*.

PROC TSMODEL requires that you specify CAS tables for all input data and all output data. Unlike PROC TIMEDATA, no actual processing of the time series data occurs in PROC TSMODEL; data in PROC TSMODEL are processed in the CAS session that is associated with the CAS librefs that you specify in the PROC TSMODEL statement.

Like PROC TIMEDATA, PROC TSMODEL supports auxiliary input tables, which you specify in the AUXDATA= option. However, PROC TSMODEL requires that either all or none of the BY variables are

present in an AUXDATA= table. No partial BY group matching for the AUXDATA= tables is supported at this time.

A simple example provides insight about the relative ease of moving from PROC TIMEDATA to PROC TSMODEL. Consider the following PROC TIMEDATA example, which rescales the Sale and Price variables for each of the Product BY groups:

```
proc timedata data=mylib.pricex
    outsum=mylib.pricexsum
    outarray=mylib.pricexoa
    outscalar=mylib.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale / accumulate=total;
var Price / accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
    if Sale[i] ne . then do;
        relsale[i] = Sale[i]/sbase;
    end;
    if Price[i] ne . then do;
        relprice[i]=Price[i]/pbase;
    end;
end;
run;
```

When you convert the preceding statements to use PROC TSMODEL with a CAS table, the code looks like this:

```
proc tsmodel data=mycas.pricex
    outsum=mycas.pricexsum
    outarray=mycas.pricexoa
    outscalar=mycas.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale /accumulate=total;
var Price /accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
submit;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
    if Sale[i] ne . then do;
        relsale[i] = Sale[i]/sbase;
    end;
    if Price[i] ne . then do;
        relprice[i]=Price[i]/pbase;
    end;
end;
endsubmit;
quit;
```

Examine these two examples closely to see the differences. At first glance, the differences might not be very obvious. Although the differences in code syntax are subtle, the underlying differences in the procedure processing are profound.

One important difference between PROC TIMEDATA and PROC TSMODEL is that PROC TSMODEL requires you to use the SUBMIT and ENDSUBMIT statements to begin and end the set of programming statements that you want to execute on the time series BY groups. You can include SAS macro statements in the statements in your SUBMIT/ENDSUBMIT block. Table 17.1 shows the differences between the processing of programming statements in PROC TSMODEL compared to PROC TIMEDATA.

Table 17.1 Comparison of PROC TSMODEL and PROC TIMEDATA

Processing Step	PROC TSMODEL	PROC TIMEDATA
Statement parsing	Any parse-time errors are reported only after the ENDSUBMIT statement. Those errors come from the parsing and compilation of your code in your CAS session. SAS line numbers for incorrect statements are not reported in the SAS log.	Statements are parsed as the procedure works its way through its statement block. Any parse-time errors are interspersed with the program statements in the SAS log.
PROC FCMP subroutines and functions	Does not support as many built-in PROC FCMP functions and subroutines.	Supports built-in PROC FCMP functions and subroutines.
RUN_MACRO	Not supported because it cannot be performed by the CAS server when it executes your program statements in the context of your CAS session.	Can call the PROC FCMP subroutine RUN_MACRO to run a SAS macro, which can then call a SAS procedure.
ID statement with a SAS date or datetime variable	An ID variable is required because the value of the ID variable is needed to determine the time index of each row within the BY group. The accumulation frequency that you specify in the INTERVAL= option in the ID statement is required.	An ID statement is not required.
Custom intervals	Custom intervals are not supported at this time.	Custom intervals are supported.
Accumulation modes	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are not supported.	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are supported.

Table 17.1 *continued*

Processing Step	PROC TSMODEL	PROC TIMEDATA
Output data	Can be printed and plotted through other SAS procedures.	Supports PRINT and PLOT options to display data.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “[Introduction to Shared Concepts](#)” on page 12 in Chapter 3, “[Shared Concepts](#).”

Getting Started: TSMODEL Procedure

This section outlines the use of the TSMODEL procedure and describes some of the analysis techniques that you can perform on timestamped transactional data.

Suppose that a bank wants to analyze the transactions that are associated with each of its customers over time. Further, suppose that the CAS table `mycas.transactions` contains four variables that are related to these transactions: `Customer`, `Date`, `Withdrawals`, and `Deposits`. The following examples illustrate possible ways to analyze these transactions by using the TSMODEL procedure.

The following statements accumulate the timestamped transactional data to form a daily time series based on the accumulated daily totals of each type of transaction (`Withdrawals` and `Deposits`). These statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.transactions
    out=mycas.timeseries(replace=yes)
    outarray=mycas.arrays(replace=yes);
  by Customer;
  id Date interval=day accumulate=total;
  var withdrawals deposits;
  outarrays Balance;
  submit;
  Balance[1] = deposits[1] - withdrawals[1];
  do t = 2 to _LENGTH_;
    Balance[t] = Balance[t-1] + (deposits[t] - withdrawals[t]);
  end;
  endsubmit;
quit;
```

The `OUT=` option requests that the resulting time series data for each customer be stored in the table `mycas.timeseries`. The `OUTARRAY=` option requests that the resulting time series data along with a newly created variable, `Balance`, be stored in the table `mycas.arrays`. Both tables are created in the CAS session's current caslib.

The `INTERVAL=DAY` option requests that the transactions be accumulated on a daily basis within each Customer according to the values of the ID variable `Date`. The `ACCUMULATE=TOTAL` option requests that the sum of the transactions be calculated. After the transactional data are accumulated into a time series format, the example code computes a daily balance for each customer. Many of the procedures provided with SAS software can be used to perform further processing on the resulting time series data. The tables that are produced by PROC TSMODEL can also be used as input to subsequent PROC TSMODEL steps.

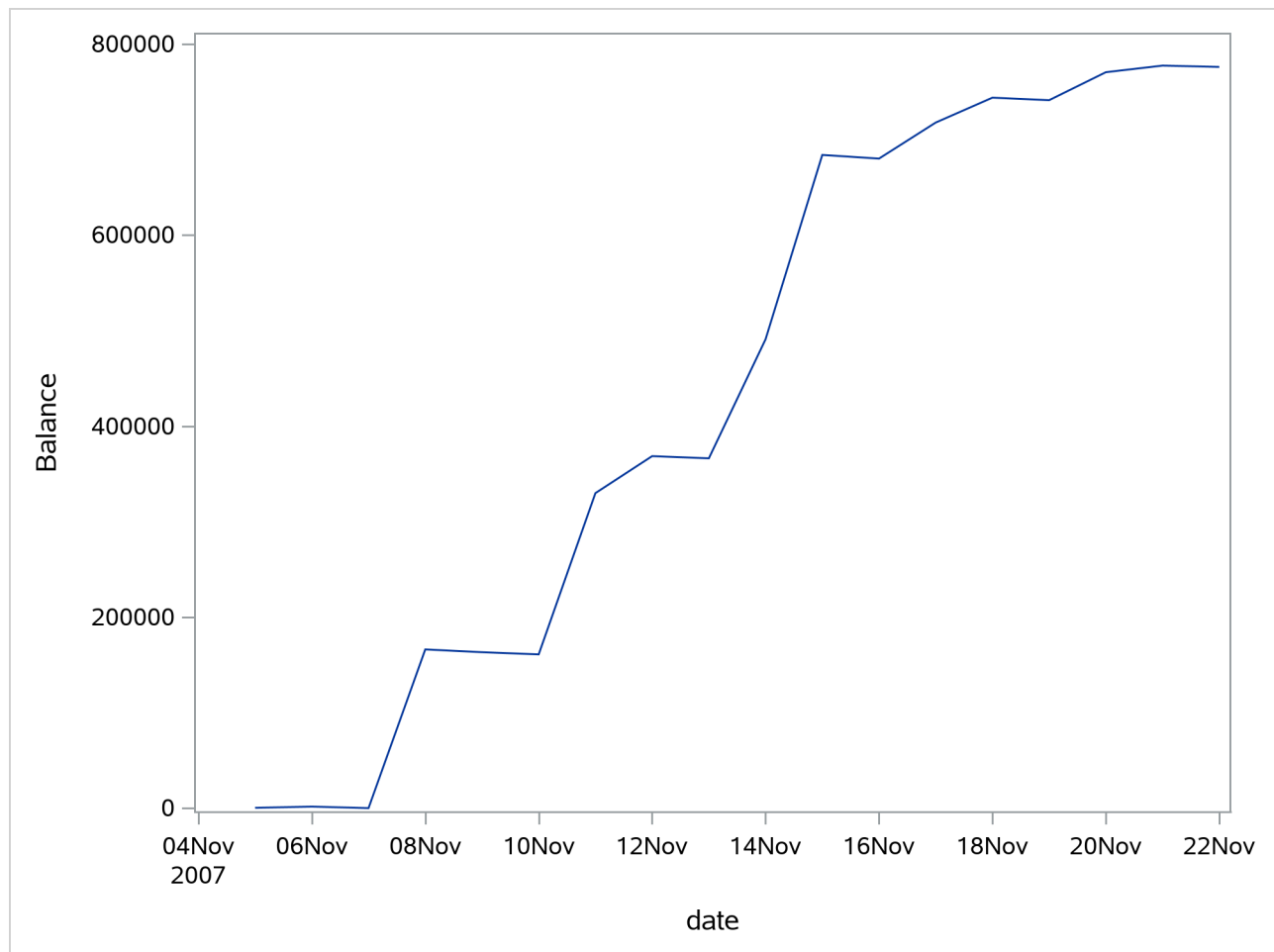
The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Figure 17.1.

Figure 17.1 Summary of Time Series Processing
The TSMODEL Procedure

Processing Summary	
CAS table	TRANSACTIONS
Number of analysis variables	2
Number of rows read	600
Number of groups read	3
Memory for group packages (KB)	1
Time to load groups (seconds)	0.0144879818
Minimum time ID	05NOV2007
Maximum time ID	22NOV2007
Minimum time periods	18
Maximum time periods	18
Number of nodes run	4
Number of nodes with data	2
Number of nodes with groups	2
Number of threads budgeted	48
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	48
Maximum threads active	48
Number of groups processed by submitted code	3
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0355381966
Number of output table rows produced	54
Number of array table rows produced	54

You might want to plot the generated Balance series for some particular customer. The following code produces a graph for the customer named 'Bill'; the graph is shown in [Figure 17.2](#).

```
proc sgplot data=mycas.arrays (where=(customer='Bill')) ;
    series x=Date y=balance;
run;
```

Figure 17.2 Balance for Bill

Syntax: TSMODEL Procedure

The following statements are available in the TSMODEL procedure:

```

PROC TSMODEL options ;
  BY variables ;
  ID variable INTERVAL=interval < options > ;
  OUTARRAYS array-name-list ;
  OUTSCALARS scalar-name-list ;
  INSCALARS scalar-name-list ;
  VAR variable-list / options ;
  REQUIRE package-list ;
  PRINT print-options ;
  SUBMIT < FILE= SAS-file-ref | 'File-path' > < submit-options > ;
  Programming statements ;
ENDSUBMIT ;

```

The PROC TSMODEL and ID statements are required.

The following sections present a summary of the statements and options that are used in PROC TSMODEL, a description of the TSMODEL statement, and then descriptions of the other statements.

Functional Summary

Table 17.2 summarizes the statements and options that control the TSMODEL procedure.

Table 17.2 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies variables to analyze	VAR	
Specifies the time ID variable	ID	
Specifies the arrays to output	OUTARRAYS	
Specifies the scalars to output	OUTSCALARS	
Specifies the scalars that are input	INSCALARS	
Specifies the packages to include	REQUIRE	
Specifies the beginning of program statements	SUBMIT	
Specifies the end of program statements	ENDSUBMIT	
Print results from program execution	PRINT	
Table Options		
Specifies the auxiliary input tables	PROC TSMODEL	AUXDATA=
Specifies the input table	PROC TSMODEL	DATA=
Specifies the output table	PROC TSMODEL	OUT=
Specifies the array output table	PROC TSMODEL	OUTARRAY=
Specifies the scalar output table	PROC TSMODEL	OUTSCALAR=
Specifies the summary statistics output table	PROC TSMODEL	OUTSUM=
Specifies the table to contain the BY-group message output log	PROC TSMODEL	OUTLOG=
Specifies whether messages are output to the OUTLOG= table by severity	PROC TSMODEL	LOGCONTROL=
Specifies the input table for scalar variables	PROC TSMODEL	INSCALAR=
Specifies a collector object for an output table	PROC TSMODEL	OUTOBJ=
Specifies a repeater object for an input table	PROC TSMODEL	INOBJ=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of the seasonal cycle	PROC TSMODEL	SEASONALITY=

Table 17.2 *continued*

Description	Statement	Option
Specifies the interval alignment	ID	ALIGN=
Specifies the method for partitioning input tables	ID	DISTRIBUTE=
Specifies the use of the best international format for the time ID values in output tables	ID	NLFORMAT=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, VAR	ACCUMULATE=
Specifies how to interpret missing values	ID, VAR	SETMISSING=
Specifies the method for trimming BY groups	ID	TRIMID=
Miscellaneous Options		
Specifies the forecast horizon or lead used to extend the CAS table	PROC TSMODEL	LEAD=
Specifies the format for the time ID variable	ID	FORMAT=
Specifies the file that contains the user-defined program	SUBMIT	FILE=
Specifies variable values available to the user-defined program	SUBMIT	DYNAMICS=

PROC TSMODEL Statement

PROC TSMODEL *options* ;

The PROC TSMODEL statement invokes the TSMODEL procedure. You can specify the following *options*:

AUXDATA=CAS-libref.data-table

names a table that contains auxiliary input data for the procedure to use for supplying time series variables. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999. For more information, see the section “Auxiliary Tables” on page 1024.

DATA=CAS-libref.data-table

names the input data table for PROC TSMODEL to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

<i>CAS-libref</i>	refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME
-------------------	--

statement. For more information about *CAS-libref*, see the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

data-table specifies the name of the input data table.

INOBJ=(*object-name*=*CAS-libref.data-table* ...)

specifies pairs, each of which binds a repeater object specified by *object-name* with an input table specified by *CAS-libref.data-table*. You can specify one or more object-table pairs as needed to associate the repeater objects that you declare in your user-defined program with their input tables. You must specify a binding for any repeater object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
inobj=(inest=mycas.outest
       inspec=mycas.outspec)
```

This code binds the repeater objects named INEST and INSPEC to the CAS tables mycas.outest and mycas.outspec, respectively. In addition to the columns that are required to satisfy the built-in table schema of a particular repeater object, each specified table must have all or none of the BY variables of the primary DATA= table. When the INOBJ= table has none of the BY variables, all the CAS table rows are input.

Repeater objects are defined in various packages that use PROC TSMODEL as a method to input data that are required for each application. For more information about creating repeater objects for a package, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section “REQUIRE Statement” on page 1019.

INSCALAR=*CAS-libref.data-table*

specifies a table to supply scalar dynamic variables to be included and made accessible to your program code as it executes.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

When you specify BY variables for the DATA= table, the INSCALAR= table must contain those BY variables. If you do not specify BY variables, then the INSCALAR= table is read unqualified for the BY variables across the CAS workers in the CAS session. In this case, only a single value for each variable is needed, and only a single row is required in the table. If the table contains multiple rows in this case, an error is generated when the procedure is called. If you specify this option, then you must also specify one or more INSCALARS statements to specify the variables that you want to include for your program to access.

If BY variables are specified in a BY statement, then the table specified in the INSCALAR= option must contain all the specified BY variables or none of them. If BY variables are present in the INSCALAR= table, then the values for the variables specified in the INSCALARS statement are input subject to BY-group processing. If BY variables are not present in the INSCALAR= table, then only a single value of each variable specified in the INSCALARS statement is input for all BY groups. It is an error for the INSCALARS= table to contain more than one value (row) for the variables if the table

is not subject to BY-group processing. If the INSCALARS= table is subject to BY-group processing and multiple values (rows) are present for any BY group, then the value can be input from any row, leading to inconsistent results. For consistent results, you should prepare the INSCALAR= tables such that only a single value is input for each BY group.

LEAD=*n*

specifies the number of periods ahead to extend time series arrays for the variables in both the VAR and OUTARRAYS statements that are output to the CAS table. You can specify this option to accommodate a forecast lead or horizon when you are preparing time series data for forecasting.

The value of *n* is relative to the ending value of the input time ID for each BY group as specified by the **TRIMID=** option in the ID statement; it is not relative to the last nonmissing observation of a particular series. By default, LEAD=0.

LOGCONTROL=(*severity*=IGNORE | KEEP < *severity*=IGNORE | KEEP . . . >)

specifies pairs that define error severity and associated message disposition for the **OUTLOG=** option. You can specify multiple LOGCONTROL= options.

You can specify zero or more pairs. In these pairs, *severity* can take one of the following values:

NONE	specifies messages that have no severity classification.
NOTE	specifies messages whose severity classification is NOTE.
WARNING	specifies messages whose severity classification is WARNING.
ERROR	specifies messages whose severity classification is ERROR.

You can specify the following values to indicate the associated message disposition:

IGNORE	ignores messages of the specified severity.
KEEP	retains messages of the specified severity.

This option is applied only when you specify an OUTLOG= option; otherwise, it has no effect. By default, LOGCONTROL=(ERROR=KEEP) when the OUTLOG= option is specified. This default value retains only messages whose severity classification is ERROR and discards all others. However, the default behavior no longer applies when you specify the LOGCONTROL= option. For example, if you specify LOGCONTROL=(NONE=KEEP), then only messages that have no severity classification are retained and all others (i.e., NOTE, WARNING, and ERROR) are discarded.

OUT=CAS-libref.data-table

names the output table to contain the time series variables that are specified in the subsequent VAR statements.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the **DATA=** option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

If BY variables are specified, they are also included in this output table. The ID variable’s fixed-interval time ID sequence is included in the OUT= CAS table. The time series variables are accumulated based on the INTERVAL= option and the variable’s ACCUMULATE= option. The OUT= CAS table is particularly useful when you want to further analyze, model, or forecast the resulting time series with other SAS procedures.

OUTARRAY=CAS-libref.data-table

names the output table to contain the time series vectors that are specified in the VAR and OUTARRAYS statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

This table also contains the variables that are specified in the BY, ID, and VAR statements in addition to the arrays that are specified in the OUTARRAYS statements.

OUTLOG=CAS-libref.data-table

names the output table to contain textual messages that are collected from the execution of the BY-group processing.

Messages captured for the BY group are subject to prefiltering by their severity based on the setting of the [LOGCONTROL=](#) option. If [PUTTOLOG=YES](#) is specified, then messages from the PUT programming statement are also included in this table. This table has rows only for BY groups that generate text messages. Messages that are related to the PROC TSMODEL syntax are not included in this table. Normally, this table contains no rows.

OUTOBJ=(object-name=CAS-libref.data-table ...)

specifies pairs, each of which binds a collector object specified by *object-name* with an output table specified by *CAS-libref.data-table*.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

You can specify one or more object-table pairs as needed to associate the collector objects that you declare in your user-defined program with their output tables. You must specify a binding for any collector object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
outobj=(oss=mycas.saleoss1
        pest=mycas.salepest1
        ofor=mycas.salefor1
        oind=mycas.saleind1
        ostat=mycas.salestat1)
```

This code binds the collector objects named OSS, PEST, OFOR, OIND, and OSTAT to the tables named SALEOSS1, SALEPEST1, SALEFOR1, SALEIND1, and SALESTAT1, respectively. These tables are all created using CAS-related context from the mycas libref.

Collector objects are defined in various packages that can be run by PROC TSMODEL. For more information about using packages, see [SAS Visual Forecasting: Time Series Packages](#). For more information about package access, see the section “[REQUIRE Statement](#)” on page 1019.

OUTSCALAR=CAS-libref.data-table

names the output table to contain the scalar names that are specified in the OUTSCALARS statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

This table also contains the variables that are specified in the BY statement and the scalars that are specified in the OUTSCALARS statements.

OUTSUM=CAS-libref.data-table

names the output table to contain the descriptive statistics. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the [DATA=](#) option and the section “Using CAS Sessions and CAS Engine Librefs” on page 999.

The descriptive statistics are based on the accumulated time series when the ACCUMULATE= option, the SETMISSING= option, or both are specified in the ID or VAR statements. This table is particularly useful when you want to analyze large numbers of series and you need a summary of the results.

PUTTOLOG=YES | NO

specifies whether to capture messages from the PUT programming statement to the BY group’s row in the table that is specified in the [OUTLOG=](#) option. You can specify the following values:

NO	does not capture messages.
YES	captures messages.

This option is applied only when you specify the OUTLOG= option; otherwise, it has no effect. This option should be used only to aid in debugging your user-defined programs on small amounts of data. This option can produce large amounts of output that increases processing time and memory usage in your CAS session processes and should be used with caution. By default, PUTTOLOG=NO.

SEASONALITY=number

specifies the length of the seasonal cycle. For example, SEASONALITY=3 means that every group of three time periods forms a seasonal cycle. By default, the length of the seasonal cycle is 1 (no seasonality) or the length that is implied by the INTERVAL= option in the ID statement. For example, INTERVAL=MONTH implies that the length of the seasonal cycle is 12.

BY Statement

BY *variables* ;

You can include a BY statement with PROC TSMODEL to obtain separate processing for groups of observations that are defined by the BY *variables*. The rows in each distinct BY group are used to accumulate the time series vectors from the timestamped row data according to a desired frequency (which is specified in the INTERVAL= option) and the accumulation mode (which is specified in the ACCUMULATE= option).

For more information about table output and BY group processing, see the section “[Table Output](#)” on page 1028.

ID Statement

ID *variable* **INTERVAL=***interval* < *options* > ;

The ID statement names a numeric *variable* that identifies the temporal order (time sequence) of observations in the input and output tables. The values of variables are assumed to be SAS date, SAS datetime, or observation values. In addition, the ID statement specifies the frequency to be associated with the time series. The ID statement *options* also specify a global treatment for how the time series variables are accumulated from the BY group’s rows and how the time ID values are aligned to form the time series. The specified information affects all variables that are specified in subsequent VAR statements. The ID statement and the INTERVAL= option are required to specify the desired accumulation frequency.

You must specify the following option:

INTERVAL=*interval*

specifies the frequency of the accumulated time series. For example, if the input table consists of quarterly observations, then specify INTERVAL=QTR.

Interval names are constructed from a basic interval type with an optional multiplier and shift. The general form of an interval name is as follows:

type< *multiplier*>.< *shift*>

The interval *type* for SAS date values are summarized as follows:

YEAR

specifies yearly intervals. Abbreviations are YEAR, YEARS, YEARLY, YR, ANNUAL, ANNUALLY, and ANNUALS. The starting subperiod *shift* is in months ([MONTH](#)).

YEARV

specifies ISO 8601 yearly intervals. The ISO 8601 year starts on the Monday immediately preceding January 4 (or on January 4 if it is a Monday). Note that it is possible for the ISO 8601 year to start in December of the preceding year. Also, some ISO 8601 years contain a leap week. For more information about ISO weeks, see Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). The starting subperiod *shift* is in ISO 8601 weeks ([WEEKV](#)).

R445YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-5-4 months (R454MON). For more information about the retail 4-5-4 calendar, see National Retail Federation (2007).

R544YR

is the same as YEARV except that the starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIYEAR

specifies semiannual intervals (every six months). Abbreviations are SEMIYEAR, SEMIYEARS, SEMIYEARLY, SEMIYR, SEMIANNUAL, and SEMIANN.

The starting subperiod *shift* is in months (MONTH). For example, SEMIYEAR.3 intervals are March–August and September–February.

QTR

specifies quarterly intervals (every three months). Abbreviations are QTR, QUARTER, QUARTERS, QUARTERLY, QTRLY, and QTRS. The starting subperiod *shift* is in months (MONTH).

R445QTR

specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454QTR

specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544QTR

specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

MONTH

specifies monthly intervals. Abbreviations are MONTH, MONTHS, MONTHLY, and MON. The starting subperiod *shift* is in months (MONTH). For example, MONTH2.2 intervals are February–March, April–May, June–July, August–September, October–November, and December–January of the following year.

R445MON

specifies retail 4-4-5 monthly intervals. The 3rd, 6th, 9th, and 12th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. All other months are four ISO 8601 weeks long. R445MON intervals begin with the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th weeks of the ISO year. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454MON

specifies retail 4-5-4 monthly intervals. The 2nd, 5th, 8th, and 11th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R454MON intervals begin with the 1st, 5th, 10th, 14th, 18th, 23rd, 27th, 31st, 36th, 40th, 44th, and 49th weeks of the ISO year. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544MON

specifies retail 5-4-4 monthly intervals. The 1st, 4th, 7th, and 10th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R544MON intervals begin with the 1st, 6th, 10th, 14th, 19th, 23rd, 27th, 32nd, 36th, 40th, 45th, and 49th weeks of the ISO year. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIMONTH

specifies semimonthly intervals. SEMIMONTH breaks each month into two periods, starting on the 1st and 16th days. Abbreviations are SEMIMONTH, SEMIMONTHS, SEMIMONTHLY, and SEMIMON. The starting subperiod *shift* is in SEMIMONTH periods. For example, SEMIMONTH2.2 specifies intervals from the 16th of one month through the 15th of the next month.

TENDAY

specifies 10-day intervals. TENDAY breaks the month into three periods, the 1st through the 10th day of the month, the 11th through the 20th day of the month, and the remainder of the month. (TENDAY is a special interval typically used for reporting automobile sales data.) The starting subperiod *shift* is in TENDAY periods. For example, TENDAY4.2 defines 40-day periods that start at the second TENDAY period.

WEEK

specifies weekly intervals of seven days. Abbreviations are WEEK, WEEKS, and WEEKLY. The starting subperiod *shift* is in days (DAY), with the days of the week numbered as 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, and 7=Saturday. For example, WEEK.7 means weekly with Saturday as the first day of the week.

WEEKV

specifies ISO 8601 weekly intervals of seven days. Each week starts on Monday. The starting subperiod *shift* is in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 starts on Monday, WEEKV.2 starts on Tuesday, and so on.

WEEKDAY**WEEKDAYdW****WEEKDAYddW****WEEKDAYdddW**

specifies daily intervals with weekend days included in the preceding weekday. Note that for a five-day work week that starts on Monday, the appropriate interval is WEEKDAY5.2. Abbreviations are WEEKDAY and WEEKDAYS. The starting subperiod *shift* is in weekdays (WEEKDAY).

The WEEKDAY interval is the same as DAY except that weekend days are absorbed into the preceding weekday. Thus, there are five WEEKDAY intervals in a calendar week: Monday, Tuesday, Wednesday, Thursday, and the three-day period Friday-Saturday-Sunday.

The default weekend days are Saturday and Sunday, but any one to six weekend days can be listed after the WEEKDAY string and followed by a W. Weekend days are specified as ‘1’ for Sunday, ‘2’ for Monday, and so forth. For example, WEEKDAY67W specifies a Friday-Saturday weekend. WEEKDAY1W specifies a six-day work week with a Sunday weekend. WEEKDAY17W is the same as WEEKDAY.

DAY

specifies daily intervals. Abbreviations are DAY, DAYS, and DAILY. The starting subperiod *shift* is in days (DAY).

The interval *type* for SAS datetime values are summarized as follows:

DTSAS-date-interval

specifies a SAS datetime interval that corresponds to the SAS date interval but operates on SAS datetime values. The SAS datetime interval is created by adding the prefix “DT” to the SAS date interval name. For example, DTMONTH.

HOURL

specifies hourly intervals. Aliases are HOUR, DTHOUR, HOURS, DTHOURS, HOURLY, DTHOURLY, HR, and DTHR. The starting subperiod *shift* is in hours (HOUR).

MINUTE

specifies minute intervals. Aliases are MINUTE, DTMINUTE, MINUTES, DTMINUTES, MIN, and DTMIN. The starting subperiod *shift* is in minutes (MINUTE).

SECOND

specifies second intervals. Aliases are SECOND, DTSECOND, SECONDS, DTSECONDS, SEC, and DTSEC. The starting subperiod *shift* is in seconds (SECOND).

The interval *type* for SAS observation values is summarized as follows:

OBS

n

specifies that observation numbers identify the time periods. You can specify INTERVAL=*n* as an alias for INTERVAL=OBS*n*.

If the SEASONALITY= option is not specified in the PROC TSMODEL statement, then the length of the seasonal cycle is implied from the INTERVAL= option. For example, INTERVAL=QTR implies a seasonal cycle of length 4. If the ACCUMULATE= option is also specified, the INTERVAL= option determines the time periods for the accumulation of observations. The INTERVAL= option is required and must be specified in the ID statement.

You can also specify the following *options*:

ACCUMULATE=option

specifies how to accumulate the table observations within each time period into a single value. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID *variable* contains the time ID values. The value of the time ID *variable* identifies the time period of the observation. The accumulated values for each time period form the time series, which is used in subsequent analysis.

The ACCUMULATE= option is necessary when multiple input observations identify the same time period. An example of this is timestamped transactional data.

The following *options* determine how to accumulate the observations within each time period based on the ID variable and the frequency specified by the INTERVAL= option. Each value indicates how the accumulated value is calculated:

TOTAL SUM	calculates as the total sum of the nonmissing values.
AVERAGE AVG	calculates as the average of the nonmissing values.
MINIMUM MIN	calculates as the minimum of the nonmissing values.
MAXIMUM MAX	calculates as the maximum of the nonmissing values.
N	calculates as the number of nonmissing observations.
NMISS	calculates as the number of missing observations.
STDDEV STD	calculates as the standard deviation of the nonmissing values.
CSS	calculates as the corrected sum of squares of the nonmissing values.
USS	calculates as the uncorrected sum of squares of the nonmissing values.

By default, ACCUMULATE=TOTAL.

The SETMISSING= option is useful for specifying how to treat missing values in the accumulated time series variable. If missing values should be interpreted as 0, then specify SETMISSING=0. For more information about accumulation, see the section “[Accumulation](#)” on page 1021.

ALIGN=option

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, the ALIGN= option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations that are supplied by the user, the ALIGN= option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often are not supplied by the user. You can specify the following options:

BEGINNING BEG B	represents each time period using the beginning SAS date or datetime value of the time period.
ENDING END E	represents each time period using the ending SAS date or datetime value of the time period.
MIDDLE MID M	represents each time period using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

DISTRIBUTE=option

specifies the method that is used to distribute the input time series observations in the input tables across the CAS worker nodes in the CAS session. This process, which is known as “autopartitioning,” is performed in order to form complete time series that are suitable for processing. You can specify the following options:

- AUTO** automatically selects the best approach, either the **PREACCUMULATE** or **SHUFFLE** option, to distribute the input time series observations across the CAS worker nodes in the CAS session. The selection occurs on the basis of both the accumulation time interval that you specify in the **INTERVAL=** option and the level of scattering of the time series observations across the CAS worker nodes.
- PREACCUMULATE | PREACC** generates partially accumulated time series (that is, those that have a fixed length and a fixed interval) in each CAS worker node in the CAS session and then distributes their observations to their destination CAS worker nodes. These partially accumulated time series are accumulated again upon arrival at their destination CAS worker nodes in order to form complete time series. This option can usually reduce the amount of time that is required in order to distribute time series observations across the CAS worker nodes, especially in the case in which you specify an accumulation interval in the **INTERVAL=** option that is larger than the natural frequency of the input data. However, this option can require additional memory in order to generate the partially accumulated time series in the CAS worker nodes.
- SHUFFLE** distributes the raw time series observations to their destination CAS worker nodes in the CAS session before accumulating them into complete time series. This option is recommended when you specify an accumulation time interval in the **INTERVAL=** option that is close to the natural frequency of the input data. This option requires less memory than the **PREACCUMULATE** option because it does not have to generate partially accumulated time series in the memory of each CAS worker node. However, this option can lead to increased run time compared to the **PREACCUMULATE** option when it is used outside the recommended setting.

By default, **DISTRIBUTE=****AUTO**.

END=*value*

specifies a SAS date or datetime value that represents the end of the data. If the last time ID variable value is less than *value*, then the series is extended with missing values. If the last time ID variable value is greater than *value*, then the series is truncated. For example, **END=**“&sysdate”D uses the automatic macro variable **SYSDATE** to extend or truncate the series to the current date. You can specify the **START=** and **END=** options to ensure that the data that are associated within each **BY** group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If this option is omitted, then the default format is inferred from the **INTERVAL=** option. You can specify **FORMAT=****_DATA_** to force the SAS format of the time ID variable to be automatically propagated to the time ID values.

NLFORMAT=*option*

enables the selection of the best international SAS format for the time ID values on the basis of the accumulation time interval that you specify in the **INTERVAL=** option. This option is ignored when you specify the **FORMAT=** option. You can specify the following options:

NO	selects the best English-language-based SAS format for the time ID values.
YES	selects the best international SAS format for the time ID values.

By default, NLFORMAT=NO.

SETMISSING=option

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can use the following *options* to determine how to interpret missing values:

<i>n</i>	interprets a missing value as having the value <i>n</i> . You can specify any number for <i>n</i> , but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because no recorded data usually implies no activity.
MISSING	interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
AVERAGE AVG	interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
MINIMUM MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=value

specifies a SAS date or datetime value that represents the beginning of the data. If the first time ID variable value is greater than *value*, then missing values are added at the beginning of the series. If the first time ID variable value is less than *value*, then the series is truncated. You can specify the START= and END= options to ensure that data associated with each BY group contain the same number of observations.

TRIMID=method

specifies the method for trimming the data in the BY groups when time series vectors are input to the user-defined program. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable will be recalculated according to the value of the **LEAD=** option (if one is specified).

You can specify one of the following *methods*:

- NONE** uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.
- LEFT** uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.
- RIGHT** uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.
- BOTH** uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

Note that when you do not specify the **START=** and **END=** options in the **ID** statement, the TSMODEL procedure is required to make an additional pass through the primary data set that you specified in the **DATA=** option in order to determine its time index span. This is required so that the total number of accumulation time intervals can be computed using the accumulation frequency that you specified in the **INTERVAL=** option. In this scenario, the TSMODEL procedure will read all rows from the primary data set exactly twice. Moreover, this effect will be captured in the numeric value associated with the row labeled 'Number of rows read' of the time series processing summary table that is generated at the end of the execution of the TSMODEL procedure, as is illustrated in [Figure 17.1](#) and [Output 17.1.1](#).

INSCALARS Statement

INSCALARS *scalar-name-list* ;

INSCALAR *scalar-name-list* ;

The INSCALARS statement specifies which scalar variables to automatically include for your program to use. These variables can be numeric or character. Each variable that you name in an INSCALARS statement must be defined in the table specified in the **INSCALAR=** option in the PROC TSMODEL statement. You must specify at least one INSCALARS statement if you specify the INSCALAR= option in the PROC TSMODEL statement. You can specify multiple INSCALARS statements. For more information, see the **INSCALAR=** option.

OUTARRAYS Statement

OUTARRAYS *array-name-list* ;

OUTARRAY *array-name-list* ;

The OUTARRAYS statement specifies a list of array names; each name specifies a numeric output array variable to be stored in the table that is specified in the **OUTARRAY=** option in the PROC TSMODEL statement. You can specify multiple OUTARRAYS statements.

Your program statements can create and use any number of arrays. The array variables that are specified in the OUTARRAYS statement are created automatically for you, and their lengths are predetermined by the length of the time ID vector for the BY group. The arrays are initialized to missing values at the start of each BY group. Although you can define and use many arrays in your program, only arrays that are listed in the OUTARRAYS statement are included in the table that is specified in the OUTARRAY= option.

OUTSCALARS Statement

OUTSCALARS *scalar-name-list* ;

OUTSCALAR *scalar-name-list* ;

The OUTSCALARS statement specifies a list of scalar names; each name specifies a numeric or character output scalar variable to be stored in the table that is specified in the **OUTSCALAR=** option in the PROC TSMODEL statement. You can specify multiple OUTSCALARS statements.

Your program statements can create and use any number of scalars. Only scalars that are listed in the OUTSCALARS statement are predefined and included in the table that is specified in the OUTSCALAR= option. The scalars are initialized to missing values at the start of each BY group. Each scalar name can specify numeric or character scalar variables which are defined as follows:

var-list defines the variables in the *var-list* as numeric scalars.

var-list \$*n* defines the variables in the *var-list* as character scalars with length *n*.

The following statement defines scalar variables rc, holdout, and nchanges as numeric scalars, and defines selected as character with length 32.

```
outscalar rc selected $32 holdout nchanges;
```

PRINT Statement

PRINT *print-options* ;

The PRINT statement allows you to display results that are saved to CAS tables that you specify in the PROC TSMODEL statement options. You can include multiple PRINT statements in your PROC TSMODEL statement block. Each PRINT statement is processed independently in the order that you specify them.

The PRINT OUTLOG statement has the following form:

PRINT OUTLOG < **WHERE** *where-clause* > ;

PRINT OUTLOG displays information from the OUTLOG= table to the SAS LOG. If you specify a BY statement, then the messages that are stored to the OUTLOG table are printed in BY group order followed with a BY line separator. You can specify an optional WHERE clause to select the rows from the OUTLOG table that you want to print. The WHERE clause must reference columns of the OUTLOG= table.

Following are four examples of PRINT OUTLOG statements that you might specify:

```
print outlog;
print outlog where regionName eq 'Region1';
print outlog where _errno_ ne 0;
print outlog where _log_ contains "sale";
```

The first PRINT statement prints all OUTLOG= table rows to the SAS LOG, the second selects rows where the regionName column has the value 'Region1', the third selects rows where the _errno_ column has a non-zero value, and the last selects rows where the _log_ column text contains the word 'sale'.

VAR Statement

VAR *variable-list* < / *options* > ;

The VAR statement lists the numeric variables in the DATA= table or AUXDATA= tables whose values are to be accumulated to form the time series.

An input table variable can be specified in only one VAR statement. You can specify any number of VAR statements. You can also specify the following *options* in the VAR statements:

ACCUMULATE=*option*

specifies how to accumulate the table observations within each time period for the variables in the *variable-list*. If you do not specify this option, accumulation is determined by the **ACCUMULATE=** option in the ID statement.

SETMISS=*option* | *number*

SETMISSING=*option* | *number*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series for variables in the *variable-list*. If you do not specify this option, missing values are set based on the **SETMISSING**= option in the ID statement.

REQUIRE Statement

REQUIRE *package-1* < *package-2* ... *package-n* > ;

REQUIRE *package* (*class-1* ... *class-n*) ;

The REQUIRE statement specifies which time series and time frequency analysis packages to make available for your user-defined program. These packages include functions, subroutines, and objects that you can utilize from your program to perform sophisticated time series processing. These packages provide functionality that ranges from a simple function to count missing observations in an array to very sophisticated objects that perform automatic time series modeling and forecasting.

The first form of the REQUIRE statement enables you to specify a list of package names that are available for use. If you use this form, you must specify at least one package name. You can specify multiple packages in the same REQUIRE statement.

The second form of the REQUIRE statement enables you to selectively identify which objects in *package* you intend to use. Packages might contain a number of different objects, and it can be more efficient to register only the ones that you plan to use.

You can specify multiple REQUIRE statements. All packages that are specified in REQUIRE statements are loaded prior to parsing your program statements so that any references are defined at the time your code is parsed. If you specify an invalid package name, then an error is returned prior to parsing your program statements. For more information about the packages that are available in SAS Visual Forecasting, see [SAS Visual Forecasting: Time Series Packages](#). For more information about the packages that are available in SAS Econometrics, see [SAS Econometrics: Econometrics Procedures](#).

SUBMIT Statement

SUBMIT < **FILE**= *SAS-file-ref* | '*File-path*' > < *submit-options* > ;

The SUBMIT statement indicates the start of your user-defined program statements or specifies the location of a file that contains your user-defined program statements.

If the SUBMIT statement does not specify a file, then all lines of code that are specified between the SUBMIT statement and the ENDSUBMIT statement are executed on the time series BY groups that arise from your timestamped input data. Compilation errors are reported back to the SAS log, and no further processing occurs.

Notice the placement of the SUBMIT and ENDSUBMIT in the example in the “[Comparison of the TSMODEL and TIMEDATA Procedures](#)” on page 996. The use of SUBMIT and ENDSUBMIT statements to delineate of the program block is a major difference between PROC TSMODEL and PROC TIMEDATA.

You can specify the following *submit-options*:

FILE=*SAS-file-ref* | '*File-path*'

specifies a file that contains your user-defined program. If you specify this option, you must not specify the ENDSUBMIT statement.

You can specify the file either by including a *SAS-file-ref* that is defined in a prior SAS FILENAME statement or by including a quoted string that identifies the host-specific file path.

The following rules apply when you include the FILE= option:

- The text file you specify must contain only the program statements that you want to submit and none of the PROC TSMODEL statements. You must not include an ENDSUBMIT statement in the file.
- The text file you specify must not contain any SAS macro statements or macro variable references. The file content is submitted as specified.
- You are free to use any mechanism of your choosing to generate the program statements in the text file. You can use a SAS DATA step with PUT statements, PROC LUA, PROC STREAM, and so on to generate the file from within a SAS program. You can use your favorite editor to generate the text file and then specify it in the SUBMIT statement if the content is static. For more information, see *Getting Started with SAS Viya for Lua*.

DYNAMICS=(*var-1*=[*number* | "*string*"] < *var-2*=[*number* | "*string*"]... >)

defines variables (constants) outside of the scope of the submitted code. These variables function as global arguments to the submitted code. In conjunction with the INSCALARS statement, these values can be used and evaluated within the submitted code. You can specify both numeric and character variable types. The type of the variable is determined by the right-hand side of the variable's declaration. For example, consider the following DYNAMICS= option:

```
DYNAMICS= (SOF= 'MAPE'  HOLDOUT=10)
```

The variable SOF is defined as a character variable, and its static value is 'MAPE'. The variable HOLDOUT is defined as a numeric variable, and its static value is 10.

ENDSUBMIT Statement

ENDSUBMIT ;

The ENDSUBMIT statement is required to terminate the user-defined program statements in the PROC TSMODEL statement block whenever you specify the SUBMIT statement without the FILE= option. Do not specify the ENDSUBMIT statement if you specify the **FILE=** option in the SUBMIT statement. If you do not specify the FILE= option and you fail to include an ENDSUBMIT statement to accompany your SUBMIT statement, then the PROC TSMODEL step ends with an error.

SAS Programming Statements

Programming statements ;

You can use all the programming statements that are allowed in PROC FCMP except the following:

- CALL RUN_MACRO subroutine
- CALL READ_ARRAY subroutine
- CALL WRITE_ARRAY subroutine
- Custom intervals in the INTCK and INTNX functions

Details: TSMODEL Procedure

The TSMODEL procedure forms time series data from transactional data, which are analyzed according to the following steps if the relevant option listed on the right is specified:

- | | |
|---------------------------------|---|
| 1. accumulation | ACCUMULATE= option in the ID or VAR statement |
| 2. missing value interpretation | SETMISSING= option in the ID or VAR statement |
| 3. program execution | user-defined program statements |
| 4. descriptive statistics | OUTSUM= option |

Accumulation

If you specify the ACCUMULATE= option in the ID or VAR statement, observations in the table that is specified in the DATA= option in the PROC TSMODEL statement are accumulated within each time period. The frequency (width of each time interval) is specified by the INTERVAL= option in the ID statement. The ID variable contains the time ID values. Each time ID value is a SAS date or datetime that identifies the time period that contains the date or datetime. Accumulation is useful when the input table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series, which is used in subsequent analyses.

For example, suppose an input table contains the following observations:

```
19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
23MAY1999    20
```

If INTERVAL=MONTH is specified in the ID statement, then all the preceding observations fall within a three-month period of time between March 1999 and May 1999. The observations are accumulated within each time period as follows:

- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

01MAR1999	40
01APR1999	.
01MAY1999	90

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

01MAR1999	20
01APR1999	.
01MAY1999	30

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

01MAR1999	10
01APR1999	.
01MAY1999	20

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

01MAR1999	30
01APR1999	.
01MAY1999	50

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

01MAR1999	14.14
01APR1999	.
01MAY1999	17.32

As you can see from the preceding examples, the accumulated time series can have missing values even though the input table observations contain no missing values.

Missing Value Interpretation

Once the data has been accumulated to form a time series based on the `INTERVAL=` and the `ACCUMULATE=` options, missing value interpretation is performed. Sometimes missing values should be interpreted as unknown values. In other situations, missing values are known, such as when missing values are created from accumulation and no observation should be interpreted as no value—that is, 0. In the former case, you can specify the `SETMISSING=` option to interpret how to treat missing values. In the latter case, you can specify `SETMISSING=0` in order to treat missing observations as no (zero) values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Summary Statistics

You can compute summary statistics from the working series by specifying the `OUTSUM=` option in the `PROC TSMODEL` statement.

SAS Programming Statements

The user-defined program for `PROC TSMODEL` can contain most of the SAS programming statements and functions available in the `DATA` step or in the `FCMP` procedure. However, there are a few differences as noted in “[SAS Programming Statements](#)” on page 1021. For more information, see the “[FCMP Procedure](#)” chapter in the *Base SAS Procedures Guide*.

All variables that are specified in the `ID` and `VAR` statements are assigned as predefined arrays for subsequent processing. In addition, all array names that are specified in the `OUTARRAYS` statements and all the scalars names that are specified in the `OUTSCALARS` statements are assigned as predefined symbols for subsequent processing.

Predefined Symbols

In addition to the predefined arrays that are specified in the `OUTARRAYS` statements and the predefined scalars that are specified in the `OUTSCALARS` statements, the `TSMODEL` procedure creates predefined symbols that are automatically available for use in the programming statements. The name and description of the predefined symbols are shown in the following subsections; these names must not be used as variable names in any input data set.

Predefined Scalar Values

<code>_FORMAT_</code>	time format, which is either implied by the <code>INTERVAL=</code> option or specified in the <code>FORMAT=</code> option in the <code>ID</code> statement
<code>_HORIZON_</code>	the time ID value that is one period beyond <code>_TEND_</code>
<code>_INTERVAL_</code>	time interval, which is specified in the <code>INTERVAL=</code> option in the <code>ID</code> statement

<code>_LEAD_</code>	forecast horizon or lead, which is specified in the <code>LEAD=</code> option in the PROC TSMODEL statement
<code>_LENGTH_</code>	length of the time series that is associated with the current BY group
<code>_SERIES_</code>	series index or BY-group counter. When PROC TSMODEL runs in a parallel processing environment, the index value is relative to the machine that processes the series or BY group. Therefore, values of <code>_SERIES_</code> might not be unique and could also vary for multiple executions of the procedure.
<code>_SEASONALITY_</code>	length of the seasonal cycle, which is specified in the <code>SEASONALITY=</code> option in the PROC TSMODEL statement or implied by the <code>INTERVAL=</code> option in the ID statement
<code>_TEND_</code>	the last time ID value over all the BY groups. <code>_TEND_</code> is independent of the value of the <code>TRIMID=</code> option in the ID statement, but the value of <code>_TEND_</code> is the same as the ending value of the output time ID variable span if you specify <code>TRIMID=NONE</code> .
<code>_TSTART_</code>	the first time ID value over all the BY groups. <code>_TSTART_</code> is independent of the value of the <code>TRIMID=</code> option in the ID statement, but the value of <code>_TSTART_</code> is the same as the starting value of the output time ID variable span if you specify <code>TRIMID=NONE</code> .

Predefined Array Values

<code>_SEASON_</code>	season index values
<code>_CYCLE_</code>	life-cycle index values

Auxiliary Tables

The TSMODEL procedure can use auxiliary tables to contribute input variables to the run of the procedure step. This functionality creates a virtual data source that allows some of the input variables to physically reside in different tables. Some input variables can reside in the primary table, which is specified in the `DATA=` option, and other input variables can reside in the tables that are specified in one or more `AUXDATA=` options. This functionality enables sharing of common time series data across multiple projects.

You can specify more than one auxiliary data source to be used to input time series vectors across a particular BY-group hierarchy. To simplify data management, you can isolate variables that have naturally different levels of BY-group qualification into separate tables and use separate `AUXDATA=` options to supply them.

AUXDATA Functionality

There are two classes of time series table sources:

- a primary table from the `DATA=` option
- auxiliary data sources from `AUXDATA=` options

The `AUXDATA=` option specifies an auxiliary table that provides time series variables that are required for processing but are not included in the table that is specified in the `DATA=` option.

You can specify multiple `AUXDATA=` options in the `PROC TSMODEL` statement. Each `AUXDATA=` option establishes an auxiliary table source to supply variables that are declared in subsequent statements in the procedure step. If no auxiliary data sources are required, then the `AUXDATA=` option can be omitted.

Variables referenced in the `TSMODEL` procedure fall into three classes:

- variables that must be physically present in the primary table, which is specified in the `DATA=` option
- variables that must be physically present in each auxiliary table that is specified in an `AUXDATA=` option
- variables that can reside in either the primary or an auxiliary table

The `ID` variable for `PROC TSMODEL` must be present in the primary table and all the auxiliary tables that you specify. Variables that you specify in the `BY` statement must be present in the primary table. The auxiliary tables must contain all of those `BY` variables or none of them. Partial matching of a leftmost subset of the `BY` variables is not supported for the auxiliary tables.

The time series variables that you specify in `VAR` statements can be input from either the primary table or an auxiliary table. Variable resolution proceeds in reverse order from the last `AUXDATA=` option in the `PROC TSMODEL` statement to the first. If the variable in question is not found in any of those, the variable must be present in the primary table for the procedure step to be successful.

AUXDATA Alignment across BY Groups

All variables in the `BY` statement must be physically present in the primary table. However, it is not necessary to have the `BY` variables present in any of the auxiliary tables. All or none of the `BY` variables can be present in any auxiliary table.

For example, suppose you have a hierarchy of (`REGION`, `PRODUCT`) in the primary table `Sales`, which holds the time series variables for monthly sales metrics. Suppose you have an auxiliary table called `Promotions` that has no `BY` variables and contains analysis variables for promotions, and another table called `Returns` that contains time series analysis variables for (`REGION`, `PRODUCT`) level groupings. In this scenario, each (`REGION`, `PRODUCT`) group in the `Sales` table always includes the time series variables from the `Promotions` table, and the analysis variables for the matching (`REGION`, `PRODUCT`) `BY` groups from the `Returns` table. So if (`'SOUTH'`, `'EDSEL'`) is a `BY` group from the primary table, any matching rows from the `Returns` table are used to define the time series variables that are contributed from that table. The time series variables that are contributed by the `Promotions` table are always included in every `BY` group from the `Sales` table.

AUXDATA Alignment over the Time Dimension

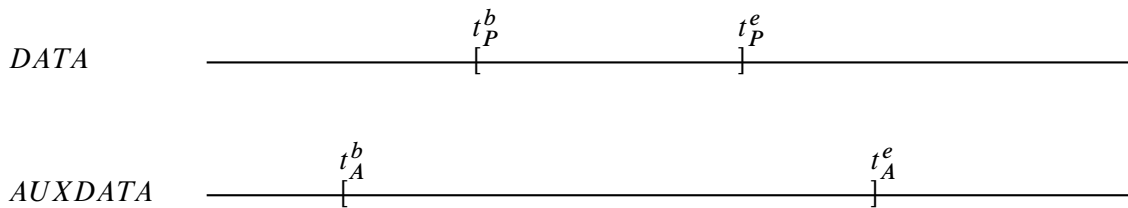
The series from each `BY` group of the primary table defines a reference time span for the auxiliary tables. Only the intersection of the time span for each auxiliary series with the reference span is input. Leading or trailing missing values are added to the auxiliary series as required to create a time series that has the same span as the reference span. The leading or trailing missing values, if any, are then interpreted according to the value of the `SETMISSING=` option. For more information, see the `SETMISSING=` option in the `ID` statement and the `SETMISSING=` option in the `VAR` statement.

When time series are input from a single primary table, the values for all observations of all the time series are contained in the primary table, and no time series needs to be extended with leading or trailing missing values. However, when time series are input from both the primary table and an auxiliary table, the time series from the auxiliary table are truncated or extended as required if the span of the auxiliary series is not identical to the reference span.

For the preceding (REGION, PRODUCT) example, which includes a primary table and two auxiliary tables, consider how differences in the reference span from each BY group affect the time series input from the auxiliary tables in the following cases. In these cases:

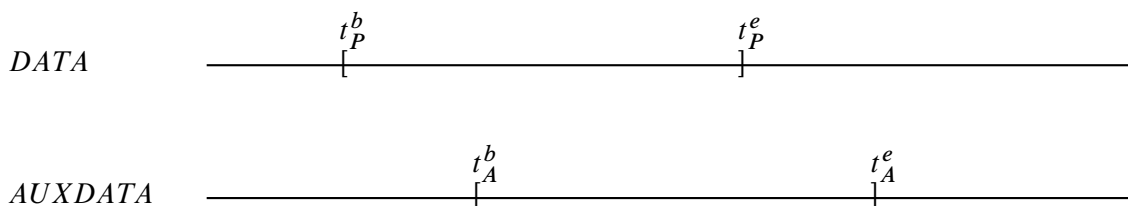
- The DATA line show a series that contains observations in the table specified in the DATA= option.
- The AUXDATA line show a series that contains observations in the table specified in an AUXDATA= option.
- t_P^b denotes the beginning time ID of the primary (DATA) series.
- t_P^e denotes the ending time ID of the primary (DATA) series.
- t_A^b denotes the beginning time ID of the AUXDATA series.
- t_A^e denotes the ending time ID of the AUXDATA series.
- $[t_P^b, t_P^e]$ denotes the time span for the primary (DATA) series (also known as the reference time span).
- $[t_A^b, t_A^e]$ denotes the time span for the AUXDATA series.

Case 1:



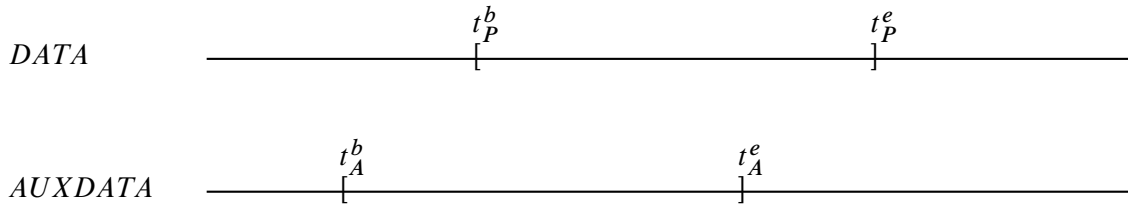
In this case, $[t_P^b, t_P^e] \subseteq [t_A^b, t_A^e]$: the auxiliary time span includes the reference span as a subset. Values in the AUXDATA series to the left of t_P^b and values to the right of t_P^e are truncated from the AUXDATA series. AUXDATA series values in $[t_P^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 2:



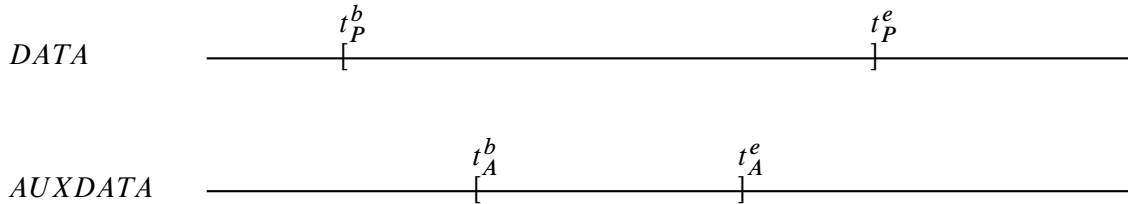
In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^b) \cup [t_A^b, t_P^e]$: the reference time span leads the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^b)$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_P^e, t_A^e]$ are truncated.

Case 3:



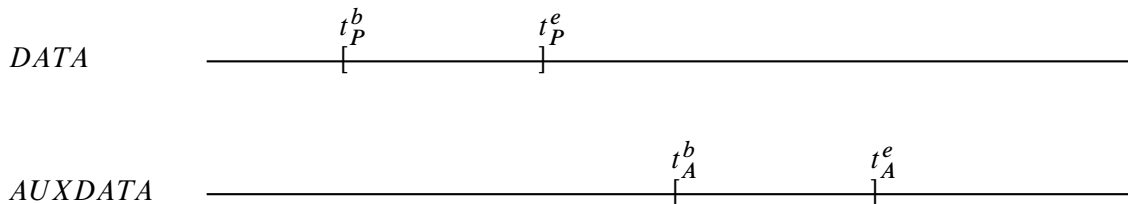
In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^e] \cup (t_A^e, t_P^e]$: the reference time span lags the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^b, t_P^b]$ are truncated.

Case 4:



In this case, $[t_A^b, t_A^e] \subset [t_P^b, t_P^e]$: the auxiliary time span is a subset of the reference time span. AUXDATA series values in $[t_P^b, t_A^b)$ and values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 5:



In this case, $[t_P^b, t_P^e] \cap [t_A^b, t_A^e] = \emptyset$: the auxiliary time span does not intersect the reference time span at all. AUXDATA series values in $[t_P^b, t_P^e]$ are set to missing values, and then those missing values are interpreted according to the value of the SETMISSING= option.

Table Output

The TSMODEL procedure can create the **OUT=**, **OUTARRAY=**, **OUTSCALAR=**, **OUTSUM=**, **OUTLOG=**, and **OUTOBJ=** tables. These tables always contain the variables that are specified in the BY statement. If a BY-group analysis step fails, then the values of this step are not recorded or are set to missing in the related output tables. Appropriate error or warning messages (or both) are recorded in the OUTLOG= table, subject to the value of the LOGCONTROL= option.

It is important to note that output tables created by PROC TSMODEL are naturally partitioned by the BY variables because of the way the BY groups are input into CAS session processes to form the time series data frames. The entire time series data for each BY group is input completely in a particular CAS session process. Consequently, all the time series processing for a particular BY group occurs in the context of a single CAS session node. Subsequent use of PROC TSMODEL to process the output tables makes use of this table-level partitioning.

The BY group partitioning of the output tables should not be confused with BY-group sort order or time ID sort order when the output tables are read by the CAS libname engine back into the SAS DATA step code for subsequent processing. If BY-group order is needed, or if time ID sort order within the BY groups is needed, you must specify PROC SORT steps in your SAS code to order the data set. For example, consider the PROC TSMODEL step in the example “[Comparison of the TSMODEL and TIMEDATA Procedures](#)” on page 996. In order to process the OUTSUM=, OUTSCALAR=, and OUTARRAY= tables in a SAS DATA step that expects the BY-group rows to be contiguous, you must specify BY Product in the SORT procedure to sort the tables mycas.Pricexsum and mycas.Pricexos. The table mycas.Pricexoa is a time series table, and you must specify BY Product Date in the SORT procedure to sort the data so that they are suitable for normal SAS time series processing. Failure to sort output tables that are generated by PROC TSMODEL can lead to errors when the tables are used as input in other procedures or steps.

OUT= Table

The **OUT=** table contains the variables that are specified in the BY, ID, or VAR statements. The ID variable values are aligned and extended based on the **ALIGN=**, **INTERVAL=**, and **LEAD=** options. The values of the variables specified in the VAR statements are accumulated based on the **ACCUMULATE=** option, and missing values are interpreted based on the **SETMISSING=** option.

OUTARRAY= Table

The **OUTARRAY=** table contains the variables that are specified in the BY, ID or VAR statements. If the ID statement is specified, then the ID variable values are aligned and extended based on the **ALIGN=** and **INTERVAL=** options. The values of the variables specified in the VAR statements are accumulated based on the **ACCUMULATE=** option, and missing values are interpreted based on the **SETMISSING=** option. In addition, the OUTARRAY= table contains the variables that are specified in the OUTARRAYS statements and the following variables:

STATUS	status flag that indicates whether the requested analyses were successful
TIMEID	time ID values
SEASON	season index values
CYCLE	life-cycle index values

Array-Variable-Names variables that are specified in the OUTARRAYS statement

The OUTARRAY= table contains the arrays that are related to the (accumulated) time series.

OUTSCALAR= Table

The **OUTSCALAR=** table contains the variables that are specified in the BY statement. In addition, the table contains the variables that are specified in the OUTSCALARS statements and the following variables:

STATUS status flag that indicates whether the requested analyses were successful

Scalar-Variable-Names variables that are specified in the OUTSCALARS statement

The OUTSCALAR= table contains the scalars that are related to the (accumulated) time series.

OUTSUM= Table

The **OUTSUM=** table contains the variables that are specified in the BY statement and the variables in the following list. This table also records the descriptive statistics for each variable that is specified in a VAR statement. Variables that are related to descriptive statistics are based on the ACCUMULATE= and SETMISSING= options in the ID and VAR statements.

NAME variable name

STATUS status flag that indicates whether the requested analyses were successful

START the starting date of each series

END the ending date of each series

STARTOBS the beginning observation number of each series

ENDOBS the ending observation number of each series

NOBS number of observations

N number of nonmissing observations

NMISS number of missing observations

MINIMUM minimum value

MAXIMUM maximum value

AVG average value

STDDEV standard deviation

OUTLOG= Table

The **OUTLOG=** table contains the variables that are specified in the BY statement and the variables in the following list. The OUTLOG= table records textual messages that arise from the processing of the BY group's time series data. Messages are filtered based on the value of the **LOGCONTROL=** option. In addition to the BY variables, the OUTLOG= table includes the following columns:

<code>_ERRNO_</code>	a numeric variable that stores the <code>_ERRNO_</code> variable for the BY group. The value of <code>_ERRNO_</code> might be set by the user-defined program directly or might be set implicitly by calling a function or method that sets the <code>_ERRNO_</code> value.
<code>_LOGLEN_</code>	a numeric variable that stores the length of the <code>_LOG_</code> variable text (byte count).
<code>_LOG_</code>	a character variable that stores the messages that are logged from the execution of the user-defined program on the BY group's time series data. All messages from the BY group are concatenated into the variable. End-of-line characters separate the individual messages. If <code>PUTTOLOG=YES</code> is specified in the PROC TSMODEL statement, the <code>_LOG_</code> variable also contains messages from any PUT programming statements that are specified.

`_STATUS_` Variable Values

The `_STATUS_` variable that appears in the `OUTARRAY=`, `OUTSCALAR=` and `OUTSUM=` tables contains a value that specifies whether the analysis has been successful or not. The `_STATUS_` variable can take the following values:

0	Analysis was successful.
3000	Accumulation failed.
4000	Missing value interpretation failed.
6000	Series is all missing.
9000	Descriptive statistics could not be computed.

Printed Output

The TSMODEL procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC TSMODEL. Printing of other results is best accomplished by the use of targeted data queries to subset and display the information from the tables that are produced by the TSMODEL procedure. For example, you might want to print the `OUTARRAY=` results only for BY groups that have a particular `_STATUS_` value in the `OUTSUM=` table.

Examples: TSMODEL Procedure

Example 17.1: Accumulating Transactional Data into Time Series Data

This example uses the TSMODEL procedure to accumulate timestamped transactional data that have been recorded at no particular frequency into time series data at a specific frequency.

Suppose that the input table `mycas.retail` contains variables `Store` and `Timestamp` and numerous other numeric transaction variables. The BY variable `Store` contains values that divide the transactions into groups (BY groups). The time ID variable `Timestamp` contains SAS date values that are recorded at no particular frequency. The other variables, `Item1`–`Item8`, contain the numeric transaction values to be analyzed. The following statements form monthly time series from the transactional data.

These statements assume that your CAS engine libref is named `mycas`, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.retail
    out=mycas.mseries(replace=yes);
    by store;
    id timestamp interval=month
        accumulate=avg
        setmiss=0
        start='01jan1998'd
        end  ='31dec2000'd;
    var item1-item8;
run;
```

The `ACCUMULATE=AVG` option in the `ID` statement accumulates the transactions that are recorded with each time period based on the average values. The `SETMISS=0` option in the `ID` statement sets the accumulated time series values for time periods that have no transactions to 0 instead of to missing. The `START='01JAN1998'D` and `END='31DEC2000'D` options request that only transactions recorded between the first day of 1998 and the last day of 2000 be considered and, if needed, extended to include this range.

The monthly time series data are stored in the table `mycas.mseries`. Each BY group that is associated with the BY variable `Store` contains an observation for each of the 36 months that are associated with the years 1998, 1999, and 2000. Each observation contains the values `Store`, `Timestamp`, and each of the analysis variables `Item1`–`Item8` in the input table.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in [Output 17.1.1](#).

Output 17.1.1 Summary of Accumulation Processing
The TSMODEL Procedure

Processing Summary		
CAS table		RETAIL
Number of analysis variables		8
Number of rows read		400
Number of groups read		4
Memory for group packages (KB)		1
Time to load groups (seconds)	0.0141348839	
Minimum time ID		NOV1998
Maximum time ID		DEC2000
Minimum time periods		36
Maximum time periods		36
Number of nodes run		4
Number of nodes with data		2
Number of nodes with groups		2
Number of threads budgeted		48
Minimum thread group count		0
Maximum thread group count		1
Minimum threads active		48
Maximum threads active		48
Number of groups processed by submitted code		4
Number of groups failing		0
Elapsed time to process groups (seconds)	0.0265569687	
Number of output table rows produced		144

After each set of transactions has been accumulated to form a corresponding time series, the accumulated time series can be analyzed using various time series analysis techniques. For example, exponentially weighted moving averages can be used to smooth each series.

If the time ID variable Timestamp contains SAS datetime values instead of SAS date values, the INTERVAL=, START=, and END= options must be changed accordingly and the following statements could be used:

```
proc tsmode1 data=mycas.retail
  out=mycas.tseries(replace=yes);
  by store;
  id timestamp interval=dtmonth
    accumulate=median
    setmiss=0
    start='01jan1998:00:00:00'dt
    end   ='31dec2000:00:00:00'dt;
  var item1-item8;
run;
```

The monthly time series data are stored in the table mycas.tseries, and the time ID values use a SAS datetime representation.

Example 17.2: User-Defined Program Statements

The following SAS macro statements create a user-defined subroutine and a user-defined function. Mylog is a subroutine that log-transforms a time series. Mymean is a function that computes the mean of a time series.

```
%macro fcmpcode;
  subroutine mylog(actual[, transform[.]]);
    outargs transform;
    actlen = DIM(actual);
    do t = 1 to actlen;
      transform[t] = log(actual[t]);
    end;
  endsub;

  function mymean(z[.]);
    nz = DIM(z);
    sum = 0;
    nnmiss = 0;
    do t = 1 to nz;
      if z[t] ne . then do;
        sum = sum + z[t];
        nnmiss = nnmiss + 1;
      end;
    end;
    if nnmiss eq 0 then return(.);

    return(sum/nnmiss);
  endsub;
%mend;
```

The following DATA step loads data set Sashelp.Air, which contains airline data into the table mycas.air. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
  set sashelp.air;
run;
```

The input table mycas.air contains the variables Air and Date. The time series is recorded monthly.

The following statements form quarterly time series from the monthly series based on the median value of the total of the transactions recorded within each month. The OUTARRAYS statement specifies the Logair and Logairc arrays as output. The OUTSCALARS statement specifies the Meanlog scalars as output. The other arrays and scalars are not part of the output. The subsequent program statements create the output arrays and scalars.

```
proc tsmodel data=mycas.air
  outarray=mycas.arrays(replace=yes)
  outscalar=mycas.scalars(replace=yes)
  outsum=mycas.airsum(replace=yes);
  id date interval=qtr acc=total;
  var air;
```

```

outarrays logair logairc;
outscalars meanlog;
submit;
  %fcmpcode;
  call mylog(air,logair);
  meanlog = mymean(air);
  do t = 1 to dim(air);
    logairc[t] = logair[t] - meanlog;
  end;
endsubmit;
quit;

```

The following PROC PRINT step displays the OUTSUM= table, as shown in [Output 17.2.1](#).

```
proc print data=mycas.airsum; run;
```

Output 17.2.1 Summary Statistics for mycas.Air

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV
1	AIR	48	48	0	341	1736	840.89583333	356.43144067
2	logair	48	48	0	5.8318824773	7.4593388952	6.6430769906	0.4395319391
3	logairc	48	48	0	-835.0639509	-833.4364944	-834.2527563	0.4395319393

Obs	_STATUS_	START	END	STARTOBS	ENDOBS
1	0	1949:1	1960:4	1	48
2	0	1949:1	1960:4	1	48
3	0	1949:1	1960:4	1	48

Example 17.3: Using Auxiliary Tables

This example demonstrates the use of the AUXDATA= option in the PROC TSMODEL statement. The data set Sashelp.Gulfoil contains oil and gas production data from the Gulf of Mexico. The following DATA step loads data set Sashelp.Gulfoil into the table mycas.gulfoil.

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```

data mycas.gulfoil(replace=yes);
  set sashelp.gulfoil;
run;

```

The variable RegionName define a time series hierarchy of interest. Suppose you want to generate two new series that contain the region's total share of oil and gas production for each month in the mycas.gulfoil table.

You first use PROC TSMODEL to perform temporal aggregation (accumulation) of the time series:

```

proc tsmode1 data=mycas.gulfoil
  outarray=mycas.allreg(replace=yes);
  id date interval=month accumulate=total;
  var oil gas;
  outarray alloil allgas;
  submit;
  do t=1 to _length_;

```

```

        alloil[t]=oil[t];
        allgas[t]=gas[t];
    end;
endsubmit;
quit;

```

In the preceding PROC TSMODEL statements, the Oil and Gas variables are accumulated across all regions because no BY statement is specified. In the following DATA step, the variables are dropped from the mycas.allreg table to avoid conflict with the original time series variables in the mycas.gulfoil table:

```

data mycas.allreg;
    set mycas.allreg;
    drop oil gas;
run;

```

You can then use PROC TSMODEL with the AUXDATA= option to compute the share of oil and gas production that is contributed by each region for each month. PROC TSMODEL reads a monthly time series for each RegionName group for the variables Oil and Gas from Mycas.Gulfoil. Two new series are produced in the variables Oilshare and Gasshare, which contain the region's share of the oil and gas production, respectively. Those share variables are specified in the OUTARRAY statement for inclusion in the table (mycas.shares), which is specified in the OUTARRAY= option. The time series that are acquired for the variables alloil and allgas are common across all of the RegionName BY groups.

```

proc tsmodel data=mycas.gulfoil
    auxdata=mycas.allreg
    outarray=mycas.shares;
by regionname;
outarray oilshare gasshare;
var oil gas alloil allgas;
id date interval=month accumulate=total;
submit;
    do i=1 to _length_;
        oilshare[i] = oil[i] / alloil[i];
        gasshare[i] = gas[i] / allgas[i];
    end;
endsubmit;
quit;

```

The following code demonstrates that the computed shares sum to 1 for each time index in the resulting Oilshare and Gasshare series. PROC TSMODEL is used to accumulate the shares for these respective variables from the table mycas.shares, and the accumulated share series at the RegionName level are stored to the table mycas.rshares with variable names Oilsum and Gassum, respectively. The tables mycas.shares and mycas.rshares are shown in [Output 17.3.1](#).

```

proc tsmodel data=mycas.shares
    outscalar=mycas.rshares(replace=yes)
    outsum=mycas.rsum(replace=yes);
id date interval=month accumulate=total;
var oilshare gasshare;
outscalar oilsum gassum;
submit;
    oilsum = 0;
    gassum = 0;
    do t=1 to _length_;

```

```

        oilsum = oilsum + oilshare[t];
        gassum = gassum + gasshare[t];
    end;
endsubmit;
quit;

proc print data=mycas.rshares; run;
proc print data=mycas.rsum; run;

```

Output 17.3.1 Validation of Oil and Gas Shares by Region

		Obs _STATUS_ oilsum gassum			
		1	0	123	123

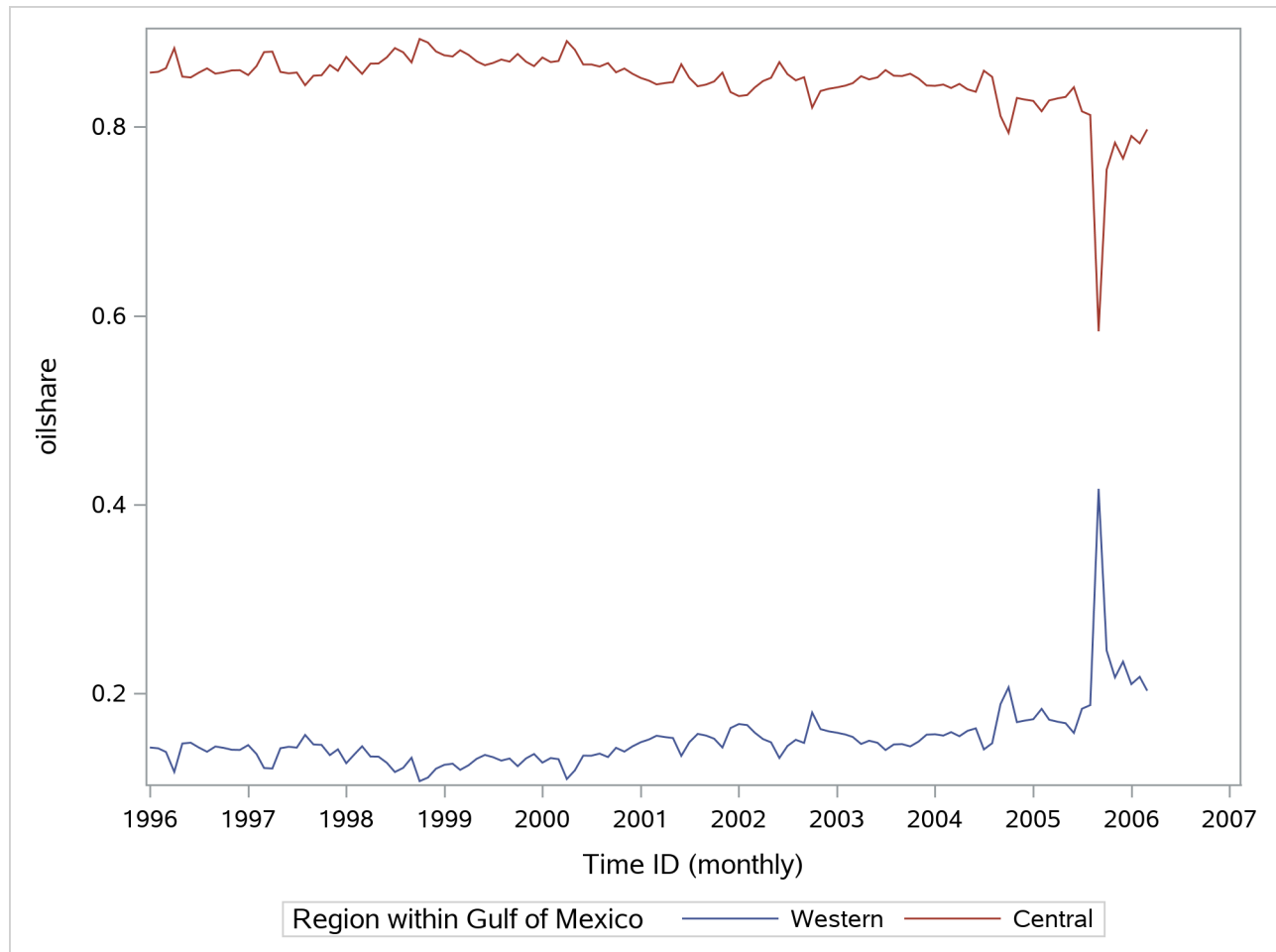
Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV	_STATUS_	START	END	STARTOBS	ENDOBS
1	oilshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123
2	gasshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123

You might also want to plot the share series. The following code produces a graph that overlays the region level share series for oil production as shown in [Output 17.3.2](#).

```

proc sgplot data=mycas.shares;
    series x=Date y=OilShare/group=RegionName;
run;

```

Output 17.3.2 Region Share of Oil Production

References

- Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2004). "Segmenting Time Series: A Survey and Novel Approach." In *Data Mining in Time Series Databases*, edited by M. Last, A. Kandel, and H. Bunke, 1–22. London: World Scientific. <https://www.worldscientific.com/worldscibooks/10.1142/5210>.
- National Retail Federation (2007). "National Retail Federation 4-5-4 Calendar." <http://www.nrf.com/>.
- Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). *ISO 8601:2004 Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times*. 3rd ed. Technical report, International Organization for Standardization.

Chapter 18

Time Series Analysis Package

Contents

Overview	1040
Using CAS Sessions and CAS Engine Librefs	1040
CORR Class	1041
CORR Object	1041
OUTCORR Object	1045
CROSSCORR Class	1048
CROSSCORR Object	1048
OUTCROSSCORR Object	1052
DECOMP Class	1055
DECOMP Object	1055
OUTDECOMP Object	1059
OUTDECOMP2 Object	1061
Details	1062
DPF Class	1064
DPF Object	1064
OUTDPE Object	1067
OUTDPROB Object	1069
Example	1070
FREQ Class	1071
FREQ Object	1071
OUTFREQ Object	1075
Example	1077
TSA Object	1077
Functional Summary	1077
ACCUMULATE Method	1079
ACCUMULATE2 Method	1081
ACF Method	1083
ARMAORDERS Method	1085
CCF Method	1086
IACF Method	1088
INTERMITTENCYTEST Method	1089
MOVINGSUMMARY Method	1091
PACF Method	1094
SCALE Method	1095
SEASONALDECOMP Method	1097

SEASONALINDICES Method	1098
SEASONTTEST Method	1100
SIMILARITY Method	1101
STATIONARITYTEST Method	1104
TRANSFORM Method	1105
UNBIASEDNESS Method	1107
WHITENOISE Method	1108
References	1109

Overview

The time series analysis (TSA) package contains a set of time series analysis functions that can be used as part of the programming statements in the TSMODEL procedure. This package provides a flexible way to analyze time series within the procedure.

NOTE: The mycas libref in the examples refers to the CAS library that is linked to a caslib. The mycas.air data table that is used in the examples refers to Sashelp.Air data. All the examples in this chapter assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref. For more information about CAS engine librefs, see *SAS Cloud Analytic Services: User's Guide*.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the `CAS` statement as follows:

```
cas mysess terminate;
```

For more information about the `CAS` statement and the `LIBNAME` statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about `CAS` and `CAS` sessions, see *SAS Cloud Analytic Services: Fundamentals*.

CORR Class

CORR Object

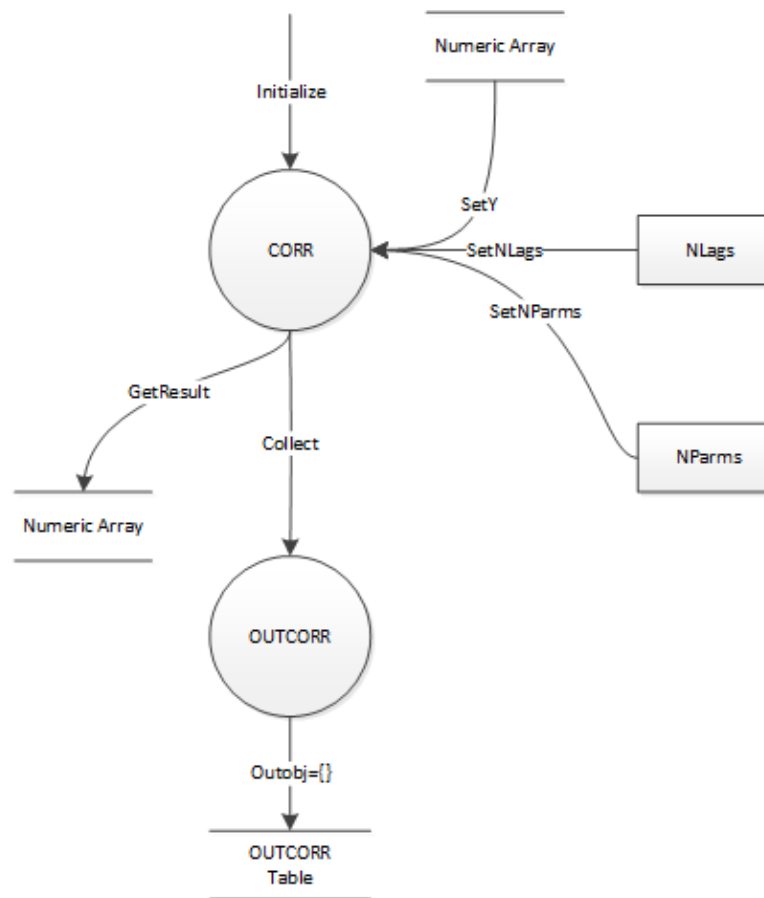
The `CORR` class object functions perform the autocorrelation and autocovariance analysis of a time series array and output all the values and results.

Table 18.1 summarizes the methods that are associated with the `CORR` object.

Table 18.1 Methods of the `CORR` Object

Method	Description
<code>GetResult</code>	Get the analysis result
<code>Initialize</code>	Initialize the <code>CORR</code> object
<code>Run</code>	Run the analysis
<code>SetNLags</code>	Set the number of lags
<code>SetNParms</code>	Set the number of parameters
<code>SetY</code>	Set the input time series

Figure 18.1 diagrams the methods of the `CORR` object.

Figure 18.1 CORR Data Flow

CORR Synopsis

DECLARE OBJECT *obj* (CORR) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetNLags (Value) ;
rc=obj.SetNParms (Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;

```

CORR Methods

CORR.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

<i>Name</i>	specifies a string that identifies the result series to return. You can specify only one of the following values within single quotation marks:
<i>acov</i>	returns an array of covariance estimates, with <i>nlag</i> + 1 entries.
<i>acf</i>	returns an array of autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>acfstd</i>	returns an array of standard errors, with <i>nlag</i> + 1 entries.
<i>acf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>acfnorm</i>	returns an array of normalized autocorrelation, with <i>nlag</i> + 1 entries.
<i>acfprob</i>	returns an array of autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>acflprob</i>	returns an array of autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>pacf</i>	returns an array of partial autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>pacfstd</i>	returns an array of partial autocorrelation standard errors, with <i>nlag</i> + 1 entries.
<i>pacf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>pacfnorm</i>	returns an array of normalized partial autocorrelation, with <i>nlag</i> + 1 entries.
<i>pacfprob</i>	returns an array of partial autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>pacflprob</i>	returns an array of partial autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>iacf</i>	returns an array of inverse autocorrelation estimates, with <i>nlag</i> + 1 entries.
<i>iacfstd</i>	returns an array of inverse autocorrelation standard errors, with <i>nlag</i> + 1 entries.
<i>iacf2std</i>	returns an array of twice standard errors, with <i>nlag</i> + 1 entries.
<i>iacfnorm</i>	returns an array of normalized inverse autocorrelation, with <i>nlag</i> + 1 entries.
<i>iacfprob</i>	returns an array of inverse autocorrelation probabilities, with <i>nlag</i> + 1 entries.
<i>iacflprob</i>	returns an array of inverse autocorrelation log probabilities, with <i>nlag</i> + 1 entries.
<i>wn</i>	returns an array of Ljung-Box white noise tests, with <i>nlag</i> + 1 entries.
<i>wnprob</i>	returns white noise probabilities.
<i>wnlprob</i>	returns white noise log probabilities.

OutputArray specifies a numeric array to receive the result series.

CORR.Initialize Method

rc=obj.Initialize () ;

Initializes a CORR object to an empty state. This method must be called before the time series arrays and other attributes are specified for the CORR object.

Arguments There are no arguments associated with this method.

CORR.Run Method

```
rc=obj.Run () ;
```

Runs the CORR object to perform the autocorrelation and autocovariance analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the CORR object.

CORR.SetNLags Method

```
rc=obj.SetNLags (Value) ;
```

Sets a named option for the CORR object.

Input Arguments You must specify the following input argument:

Value specifies the number of lags to use in the calculation. The default value is 24.

CORR.SetNParms Method

```
rc=obj.SetNParms (Value) ;
```

Sets a named option for the CORR object.

Input Arguments You must specify the following input argument:

Value specifies the degrees of freedom associated with the Ljung-Box statistics. The default value is 0.

CORR.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the CORR object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CORR object.

Arguments There are no arguments associated with this method.

OUTCORR Object

The OUTCORR object collects output from a CORR object.

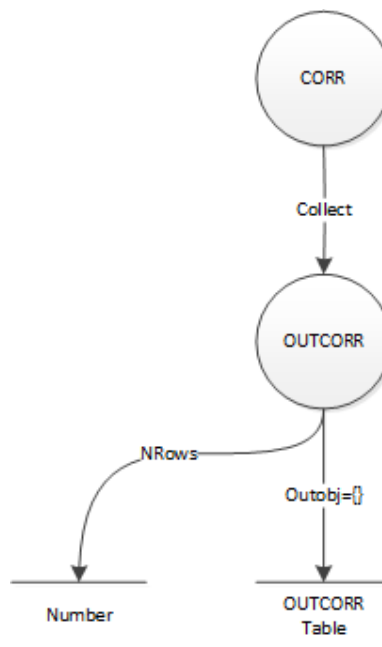
Table 18.2 summarizes the methods that are associated with the OUTCORR object.

Table 18.2 Methods of the OUTCORR Object

Method	Description
Collect	Collect the results of a CORR object
nrows	Get the current row count from the OUTCORR object

Figure 18.2 diagrams the methods of the OUTCORR object.

Figure 18.2 OUTCORR Data Flow



OUTCORR Methods

OUTCORR.Collect Method

rc=obj.Collect () ;

Collects the results from a CORR object and saves the result to a CAS table.

Arguments There are no arguments associated with this method.

OUTCORR.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTCORR object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTCORR, collects the results of a CORR object.

OUTCORR collects the results of a CORR object. The output table contains the columns that are shown in Table 18.3.

Table 18.3 OUTCORR Output Table Contents

Column Name	Column Content	Statistics Used to Calculate Content
NAME	Name of target variable	N/A
LAG	Array of time lags	$h \in \{0, \dots, H\}$
N	Number of variance products	N_h , the number of observed products at lag h , ignoring missing values
MU	Mean value	Mean value
ACOV	Array of covariance estimates, with $nlag + 1$ entries	$\hat{\gamma}(h) = \frac{1}{T} \sum_{t=h+1}^T (y_t - \bar{y})(y_{t-h} - \bar{y})$, or $\hat{\gamma}(h) = \frac{1}{N_h} \sum_{t=h+1}^T (y_t - \bar{y})(y_{t-h} - \bar{y})$ when embedded missing values are present
ACF	Array of autocorrelation estimates, with $nlag + 1$ entries	$\hat{\rho}(h) = \hat{\gamma}(h)/\hat{\gamma}(0)$
ACFSTD	Array of standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\rho}(h)) = \sqrt{\frac{1}{T} \left(1 + 2 \sum_{j=1}^{h-1} \hat{\rho}(j)^2 \right)}$
ACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\rho}(h)) = \begin{cases} 1 & \hat{\rho}(h) > 2\text{Std}(\hat{\rho}(h)) \\ 0 & -2\text{Std}(\hat{\rho}(h)) < \hat{\rho}(h) < 2\text{Std}(\hat{\rho}(h)) \\ -1 & \hat{\rho}(h) < -2\text{Std}(\hat{\rho}(h)) \end{cases}$
ACFNORM	Array of normalized autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\rho}(h)) = \hat{\rho}(h)/\text{Std}(\hat{\rho}(h))$

Table 18.3 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
ACFPROB	Array of autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\rho}(h)) = 2(1 - \Phi(\text{Norm}(\hat{\rho}(h))))$
ACFLPROB	Array of autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\rho}(h)) = -\log_{10}(\text{Prob}(\hat{\rho}(h)))$
PACF	Array of partial autocorrelation estimates, with $nlag + 1$ entries	$\hat{\phi}(h) = \Gamma_{(0,h-1)}\{\gamma_j\}_{j=1}^h$
PACFSTD	Array of partial autocorrelation standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\phi}(h)) = 1/\sqrt{N_0}$
PACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\phi}(h)) = \begin{cases} 1 & \hat{\phi}(h) > 2\text{Std}(\hat{\phi}(h)) \\ 0 & -2\text{Std}(\hat{\phi}(h)) < \hat{\phi}(h) < 2\text{Std}(\hat{\phi}(h)) \\ -1 & \hat{\phi}(h) < -2\text{Std}(\hat{\phi}(h)) \end{cases}$
PACFNORM	Array of normalized partial autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\rho}(h)) = \hat{\rho}(h)/\text{Std}(\hat{\rho}(h))$
PACFPROB	Array of partial autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\phi}(h)) = 2(1 - \Phi(\text{Norm}(\hat{\phi}(h))))$
PACFLPROB	Array of partial autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\phi}(h)) = -\log_{10}(\text{Prob}(\hat{\phi}(h)))$
IACF	Array of inverse autocorrelation estimates, with $nlag + 1$ entries	$\hat{\theta}(h)$
IACFSTD	Array of inverse autocorrelation standard errors, with $nlag + 1$ entries	$\text{Std}(\hat{\theta}(h)) = 1/\sqrt{N_0}$
IACF2STD	Array of twice standard errors, with $nlag + 1$ entries	$\text{Flag}(\hat{\theta}(h)) = \begin{cases} 1 & \hat{\theta}(h) > 2\text{Std}(\hat{\theta}(h)) \\ 0 & -2\text{Std}(\hat{\theta}(h)) < \hat{\theta}(h) < 2\text{Std}(\hat{\theta}(h)) \\ -1 & \hat{\theta}(h) < -2\text{Std}(\hat{\theta}(h)) \end{cases}$

Table 18.3 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
IACFNORM	Array of normalized inverse autocorrelation, with $nlag + 1$ entries	$\text{Norm}(\hat{\theta}(h)) = \hat{\theta}(h) / \text{Std}(\hat{\theta}(h))$
IACFPROB	Array of inverse autocorrelation probabilities, with $nlag + 1$ entries	$\text{Prob}(\hat{\theta}(h)) = 2 \left(1 - \Phi \left(\text{Norm}(\hat{\theta}(h)) \right) \right)$
IACFLPROB	Array of inverse autocorrelation log probabilities, with $nlag + 1$ entries	$\text{LogProb}(\hat{\theta}(h)) = -\log_{10}(\text{Prob}(\hat{\theta}(h)))$
WN	Array of Ljung-Box white noise tests, with $nlag + 1$ entries	$Q(h) = T(T+2) \sum_{j=1}^h \rho(j)^2 / (T-j)$
WNPROB	White noise probabilities	$\text{Prob}(Q(h)) = \chi_{\max(1, h-p)}(Q(h))$
WNLPROB	White noise log probabilities	$\text{LogProb}(Q(h)) = -\log_{10}(\text{Prob}(Q(h)))$

CROSSCORR Class

CROSSCORR Object

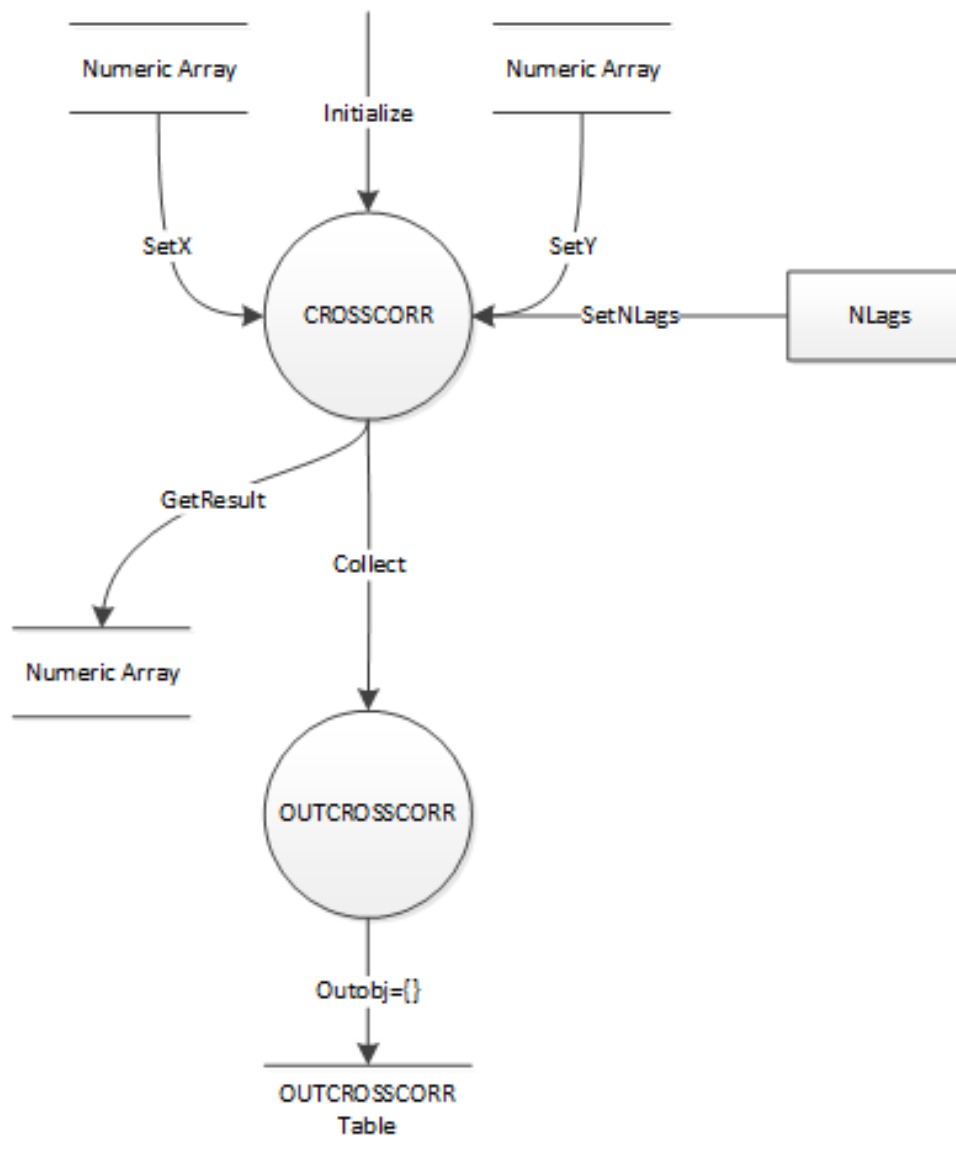
The CROSSCORR class object functions perform the cross-correlation and cross-covariance analysis of two time series arrays and output all the values and results.

Table 18.4 summarizes the methods that are associated with the CROSSCORR object.

Table 18.4 Methods of the CROSSCORR Object

Method	Description
GetResult	Get the analysis result
Initialize	Initialize the CROSSCORR object
Run	Run the analysis
SetNLags	Set the number of lags
SetX	Set the cross time series array
SetY	Set the primary time series array

Figure 18.3 diagrams the methods of the CROSSCORR object.

Figure 18.3 CROSSCORR Data Flow

CROSSCORR Synopsis

DECLARE OBJECT *obj* (CROSSCORR) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetX (XSeries) ;
rc=obj.SetNLags (Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;
  
```

CROSSCORR Methods

CROSSCORR.GetResult Method

`rc=obj.GetResult ('Name',OutputArray) ;`

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

<i>Name</i>	specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:
<i>ccov</i>	returns an array of cross-covariance estimates, with $2 \times nlag + 1$ entries.
<i>ccf</i>	returns an array of cross-correlation estimates, with $2 \times nlag + 1$ entries.
<i>ccfst</i>	returns an array of standard errors, with $2 \times nlag + 1$ entries.
<i>ccf2std</i>	returns an array of double standard errors, with $2 \times nlag + 1$ entries.
<i>ccfnorm</i>	returns an array of normalized cross-correlation, with $2 \times nlag + 1$ entries.
<i>ccfprob</i>	returns an array of probabilities, with $2 \times nlag + 1$ entries.
<i>ccflprob</i>	returns an array of log probabilities, with $2 \times nlag + 1$ entries.

OutputArray specifies a numeric array to receive the result series.

CROSSCORR.Initialize Method

`rc=obj.Initialize () ;`

Initializes an CROSSCORR object to an empty state. This method must be called before the time series arrays and other attributes are specified for the CROSSCORR object.

Arguments There are no arguments associated with this method.

CROSSCORR.Run Method

`rc=obj.Run () ;`

Runs the CROSSCORR object to perform the cross-correlation and cross-covariance analysis by using the time series arrays *YSeries* and *XSeries* that have been specified for it. Upon successful completion, various results can be extracted from the CROSSCORR object.

Arguments There are no arguments associated with this method.

CROSSCORR.SetNLags Method

`rc=obj.SetNLags (Value) ;`

Sets a named option for the CROSSCORR object.

Input Arguments You must specify the following input argument:

Value specifies the number of lags to use in the calculation. The default value is 24.

CROSSCORR.SetX Method

rc=obj.SetX (XSeries) ;

Specifies the cross time series array (*XSeries*) for the CROSSCORR object.

Input Arguments You must specify the following input argument:

XSeries specifies a numeric array that contains the dependent series for the CROSSCORR object.

CROSSCORR.SetY Method

rc=obj.SetY (YSeries) ;

Specifies the primary time series array (*YSeries*) for the CROSSCORR object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CROSSCORR object.

OUTCROSSCORR Object

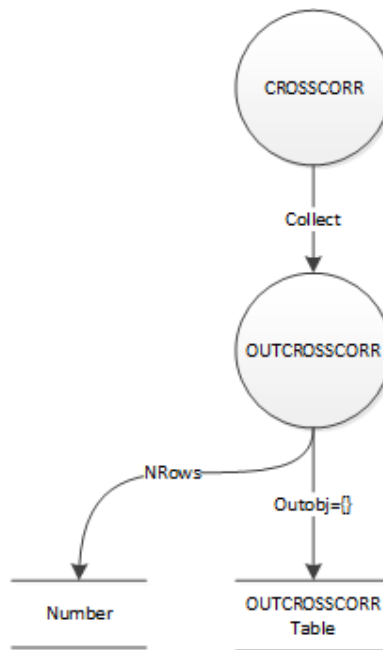
The OUTCROSSCORR object collects output from a CROSSCORR object.

Table 18.5 summarizes the methods that are associated with the OUTCROSSCORR object.

Table 18.5 Methods of the OUTCROSSCORR Object

Method	Description
Collect	Collect the results of a CROSSCORR object
nrows	Get the current row count from the OUTCROSSCORR object

Figure 18.4 diagrams the methods of the OUTCROSSCORR object.

Figure 18.4 OUTCROSSCORR Data Flow

OUTCROSSCORR Methods

OUTCROSSCORR.Collect Method

rc=obj.Collect () ;

Collects the results from a CROSSCORR object and saves the result to a CAS table.

Arguments There are no arguments associated with this method.

OUTCROSSCORR.nrows Attribute

nrows=obj.nrows () ;

Gets the current row count from the OUTCROSSCORR object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTCROSSCORR, collects the results of a CROSSCORR object.

OUTCROSSCORR collects the results of a CROSSCORR object. The output table contains the columns that are shown in [Table 18.6](#).

Table 18.6 OUTCROSSCORR Output Table Contents

Column Name	Column Content	Statistics Used to Calculate Content
NAME	Variable name	N/A
CROSS	Cross variable name	N/A
LAG	Array of time lags	$h \in \{-H, \dots, 0, \dots, H\}$
N	Number of variance products	N_h , the number of observed products at lag h , ignoring missing values
MEAN	Mean of variable	
CROSSMEAN	Mean of cross variable	
CCOV	Array of cross-covariance estimates, with $2 \times nlag + 1$ entries	$\hat{\gamma}_{x,y}(h) = \begin{cases} \frac{1}{T} \sum_{t=h+1}^T (x_t - \bar{x})(y_{t-h} - \bar{y}) & 0 \leq h < T \\ \frac{1}{T} \sum_{t= h +1}^T (x_{t- h } - \bar{x})(y_t - \bar{y}) & -T < h < 0 \end{cases}$ or $\hat{\gamma}_{x,y}(h) = \begin{cases} \frac{1}{N_h} \sum_{t=h+1}^T (x_t - \bar{x})(y_{t-h} - \bar{y}) & 0 \leq h < T \\ \frac{1}{N_h} \sum_{t= h +1}^T (x_{t- h } - \bar{x})(y_t - \bar{y}) & -T < h < 0 \end{cases}$ when embedded missing values are present
CCF	Array of cross-correlation estimates, with $2 \times nlag + 1$ entries	$\hat{\rho}_{x,y}(h) = \hat{\gamma}_{x,y}(h) / \sqrt{\hat{\gamma}_x(0)\hat{\gamma}_y(0)}$
CCFSTD	Array of cross-correlation standard errors, with $2 \times nlag + 1$ entries	$\text{Std}(\hat{\rho}_{x,y}(h)) = 1/\sqrt{N_0}$
CCF2STD	Array of indicators of sample cross-correlation outside 2 standard errors, with $2 \times nlag + 1$ entries	$\text{Flag}(\hat{\rho}_{x,y}(h)) = \begin{cases} 1 & \hat{\rho}_{x,y}(h) > 2\text{Std}(\hat{\rho}_{x,y}(h)) \\ 0 & -2\text{Std}(\hat{\rho}_{x,y}(h)) < \hat{\rho}_{x,y}(h) < 2\text{Std}(\hat{\rho}_{x,y}(h)) \\ -1 & \hat{\rho}_{x,y}(h) < -2\text{Std}(\hat{\rho}_{x,y}(h)) \end{cases}$
CCFNORM	Array of normalized cross-correlations, with $2 \times nlag + 1$ entries	$\text{Norm}(\hat{\rho}_{x,y}(h)) = \hat{\rho}_{x,y}(h) / \text{Std}(\hat{\rho}_{x,y}(h))$

Table 18.6 *continued*

Column Name	Column Content	Statistics Used to Calculate Content
CCFPROB	Array of cross-correlation probabilities, with $2 \times nlag + 1$ entries	$\text{Prob}(\hat{\rho}_{x,y}(h)) = 2 (1 - \Phi(\text{Norm}(\hat{\rho}_{x,y}(h))))$
CCFLPROB	Array of cross-correlation log probabilities, with $2 \times nlag + 1$ entries	$\text{LogProb}(\hat{\rho}_{x,y}(h)) = -\log_{10}(\text{Prob}(\hat{\rho}_{x,y}(h)))$

DECOMP Class

DECOMP Object

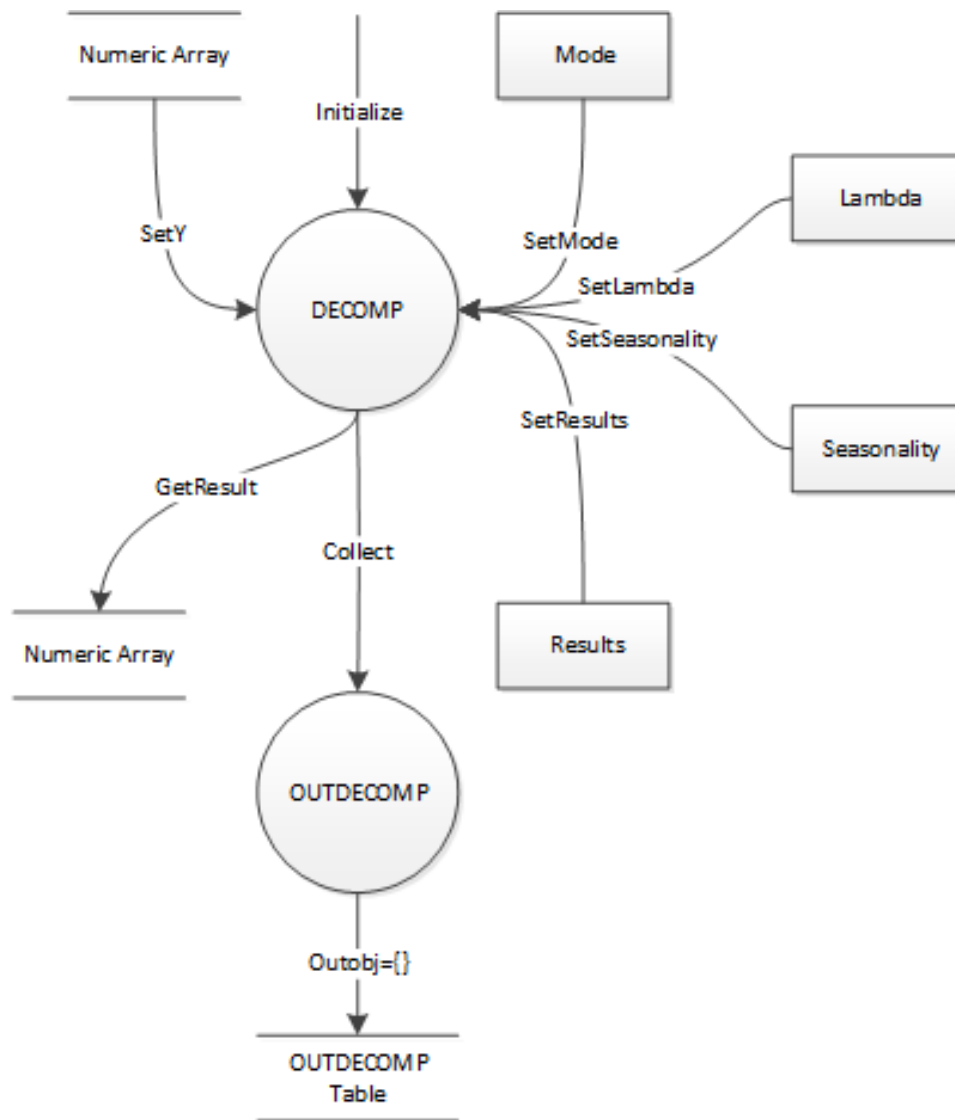
The DECOMP class object functions perform the seasonal decomposition or analysis of the time series arrays and output all the values and results.

Table 18.7 summarizes the methods that are associated with the DECOMP object.

Table 18.7 Methods of the DECOMP Object

Method	Description
GetResult	Get the analysis results
Initialize	Initialize the DECOMP object
Run	Run the analysis
SetLambda	Set the Hodrick-Prescott filter parameter for trend-cycle decomposition
SetMode	Set the type of decomposition
SetResults	Set the analysis results to be generated
SetSeasonality	Set the length of the seasonal cycles
SetY	Set the time series array

Figure 18.5 diagrams the methods of the DECOMP object.

Figure 18.5 DECOMP Data Flow

DECOMP Synopsis

DECLARE OBJECT *obj* (DECOMP) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetMode ('Name') ;
rc=obj.SetResults ('Name'<,'Name',... >) ;
rc=obj.SetSeasonality (Value) ;
rc=obj.SetLambda (Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name',OutputArray) ;

```

DECOMP Methods

DECOMP.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

Name specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:

<i>tcc</i>	returns the trend-cycle component.
<i>sic</i>	returns the seasonal-irregular component.
<i>sc</i>	returns the seasonal component.
<i>scstd</i>	returns the seasonal component standard errors.
<i>tcs</i>	returns the trend-cycle-seasonal component.
<i>ic</i>	returns the irregular component.
<i>sa</i>	returns the seasonally adjusted series.
<i>pcsa</i>	returns the percentage of change in seasonally adjusted series.
<i>tc</i>	returns the trend component.
<i>cc</i>	returns the cycle component.

OutputArray specifies a numeric array to receive the result series.

DECOMP.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes a DECOMP object to an empty state. This method must be called before the time series array and other attributes are specified for the DECOMP object.

Arguments There are no arguments associated with this method.

DECOMP.Run Method

```
rc=obj.Run () ;
```

Runs the DECOMP object to perform the seasonal decomposition or analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the DECOMP object.

Arguments There are no arguments associated with this method.

DECOMP.SetLambda Method

```
rc=obj.SetLambda (Value) ;
```

Sets the Hodrick-Prescott filter parameter for trend-cycle decomposition.

Input Arguments You must specify the following input argument:

Value specifies the length of the seasonal cycle. The default value is 1.

DECOMP.SetMode Method

rc=obj.SetMode (Value) ;

Sets the type of decomposition to be used to decompose the time series.

Input Arguments You can specify one of the following *Values*:

ADD | ADDITIVE specifies additive decomposition.
LOGADD | LOGADDITIVE specifies log-additive decomposition.
MULT | MULTIPLICATIVE specifies multiplicative decomposition.
MULTORADD specifies multiplicative or additive decomposition, depending on the data.
PSEUDOADD | PSEUDOADDITIVE specifies pseudo-additive decomposition.

DECOMP.SetResults Method

rc=obj.SetResults ('Name'<,'Name',... >) ;

Set a group of results to be generated in the DECOMP object.

Input Arguments You must specify the following input arguments:

Name specifies one or multiple strings that identify the result series to be generated. You can specify the following values within single quotation marks:

<i>tcc</i>	specifies the trend-cycle component.
<i>sic</i>	specifies the seasonal-irregular component.
<i>sc</i>	specifies the seasonal component.
<i>scstd</i>	specifies the seasonal component standard errors.
<i>tcs</i>	specifies the trend-cycle-seasonal component.
<i>ic</i>	specifies the irregular component.
<i>sa</i>	specifies the seasonally adjusted series.
<i>pcsa</i>	specifies the percentage of change in seasonally adjusted series.
<i>tc</i>	specifies the trend component.
<i>cc</i>	specifies the cycle component.

OutputArray specifies a numeric array to receive the result series.

DECOMP.SetSeasonality Method

rc=obj.SetSeasonality (Value) ;

Sets the length of seasonal cycles for the DECOMP object.

Input Arguments You must specify the following input argument:

Value specifies the length of the seasonal cycle. The default value is 1.

DECOMP.SetY Method

rc=obj.SetY (YSeries) ;

Specifies the time series array (*YSeries*) for the DECOMP object.

Input Arguments You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the DECOMP object.

OUTDECOMP Object

The OUTDECOMP object collects output from DECOMP object.

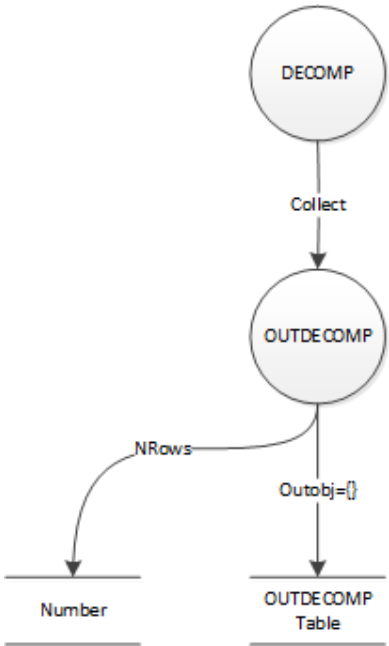
Table 18.8 summarizes the methods that are associated with the OUTDECOMP object.

Table 18.8 Methods of the OUTDECOMP Object

Method	Description
Collect	Collect the results of a DECOMP object
nrows	Get the current row count from the OUTDECOMP object

Figure 18.6 diagrams the methods of the OUTDECOMP object.

Figure 18.6 OUTDECOMP Data Flow



OUTDECOMP Methods

OUTDECOMP.Collect Method

```
rc=obj.Collect () ;
```

Collects the results from a DECOMP object and saves them to a CAS table.

Arguments There are no arguments associated with this method.

OUTDECOMP.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTDECOMP object.

Arguments

There are no arguments associated with this method.

Output Table Schema

The collector object, OUTDECOMP, collects the results of a DECOMP object.

OUTDECOMP collects the results of a DECOMP object. The output table contains the columns that are shown in [Table 18.9](#).

Table 18.9 OUTDECOMP Output Table Contents

Column Name	Column Content
NAME	Variable name
MODE	Mode of decomposition
TIMEID	Time ID values
SEASON	Seasonal index
ORIGINAL	Original series
TCC	Trend-cycle component
SIC	Seasonal-irregular component
SC	Seasonal component
SCSTD	Seasonal component standard error
TCS	Trend-cycle-seasonal component
IC	Irregular component
SA	Seasonally adjusted series
PCSA	Percentage change in seasonally adjusted series
TC	Trend component
CC	Cycle component

For more information about each components, see [Table 18.12](#).

OUTDECOMP2 Object

The OUTDECOMP2 object collects output from DECOMP object. In this object, the time periods are recorded as the column names instead of decomposition components.

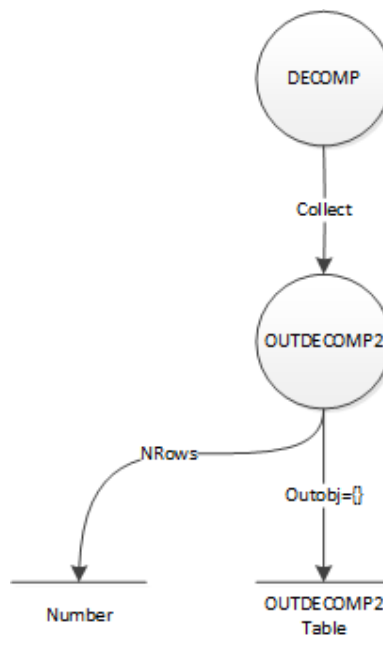
Table 18.10 summarizes the methods that are associated with the OUTDECOMP2 object.

Table 18.10 Methods of the OUTDECOMP2 Object

Method	Description
Collect	Collect the results of a DECOMP object
nrows	Get the current row count from the OUTDECOMP2 object

Figure 18.7 diagrams the methods of the OUTDECOMP2 object.

Figure 18.7 OUTDECOMP2 Data Flow



OUTDECOMP2 Methods

OUTDECOMP2.Collect Method

```
rc=obj.Collect () ;
```

Collects the results from a DECOMP object and saves them to a CAS table.

Arguments There are no arguments associated with this method.

OUTDECOMP2.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTDECOMP2 object.

Arguments

There are no arguments associated with this method.

Output Table Schema

The collector object, OUTDECOMP2, collects the results of a DECOMP object.

OUTDECOMP2 collects the results of a DECOMP object. The output table contains the columns that are shown in [Table 18.11](#).

Table 18.11 OUTDECOMP2 Output Table Contents

Column Name	Column Content
NAME	Variable name
MODE	Mode of decomposition
COMPONENT	Component Name
PERIOD	Period

Details

You can use the DECOMP object to perform seasonal decomposition (also called seasonal adjustment) on the working series. The DECOMP object uses classical decomposition. More complex seasonal decomposition or adjustment analysis can be performed by using the X11 or the X12 procedure of SAS/ETS.

The SetMode method determines the mode of the seasonal adjustment decomposition to be performed. There are four modes: multiplicative (MULT), additive (ADD), pseudo-additive (PSEUDOADD), and log-additive (LOGADD) decomposition (in which the components are exponentiated to the original metric). The default value for the SetMode method is MULTORADD, which uses multiplicative mode for series that are strictly positive, pseudo-additive mode for series that are nonnegative, and additive mode for series that are not nonnegative.

The SetLambda method specifies the Hodrick-Prescott filter parameter (Hodrick and Prescott 1980) for trend-cycle decomposition. The Hodrick-Prescott filter is used to decompose the trend-cycle component into the trend component and the cycle component in an additive fashion. The SetLambda method takes a single argument *Value*, which specifies the length of the seasonal cycle. A smaller *Value* assigns less significance to the cycle; that is, a value of 0 implies no cycle component. The default value is 1600.

[Table 18.12](#) shows the notation and keywords that are associated with seasonal decomposition (adjustment) analysis.

Table 18.12 Seasonal Adjustment Formulas

Component	Input Argument	SetMode Value	Formula
Original series	original	MULT	$O_t = TC_t S_t I_t$
		ADD	$O_t = TC_t + S_t + I_t$
		LOGADD	$\log(O_t) = TC_t + S_t + I_t$
		PSEUDOADD	$O_t = TC_t(S_t + I_t - 1)$
Trend-cycle component	tcc	MULT	Centered moving average of O_t
		ADD	Centered moving average of O_t
		LOGADD	Centered moving average of $\log(O_t)$
		PSEUDOADD	Centered moving average of O_t
Seasonal-irregular component	sic	MULT	$SI_t = S_t I_t = O_t / TC_t$
		ADD	$SI_t = S_t + I_t = O_t - TC_t$
		LOGADD	$SI_t = S_t + I_t = \log(O_t) - TC_t$
		PSEUDOADD	$SI_t = S_t + I_t - 1 = O_t / TC_t$
Seasonal component	sc	MULT	Seasonal averages of SI_t
		ADD	Seasonal averages of SI_t
		LOGADD	Seasonal averages of SI_t
		PSEUDOADD	Seasonal averages of SI_t
Irregular component	ic	MULT	$I_t = SI_t / S_t$
		ADD	$I_t = SI_t - S_t$
		LOGADD	$I_t = SI_t - S_t$
		PSEUDOADD	$I_t = SI_t - S_t + 1$
Trend-cycle-seasonal component	tcs	MULT	$TCS_t = TC_t S_t = O_t / I_t$
		ADD	$TCS_t = TC_t + S_t = O_t - I_t$
		LOGADD	$TCS_t = TC_t + S_t = O_t - I_t$
		PSEUDOADD	$TCS_t = TC_t S_t$
Trend component	tc	MULT	$T_t = TC_t - C_t$
		ADD	$T_t = TC_t - C_t$
		LOGADD	$T_t = TC_t - C_t$
		PSEUDOADD	$T_t = TC_t - C_t$
Cycle component	cc	MULT	$C_t = TC_t - T_t$
		ADD	$C_t = TC_t - T_t$
		LOGADD	$C_t = TC_t - T_t$
		PSEUDOADD	$C_t = TC_t - T_t$
Seasonally adjusted series	sa	MULT	$SA_t = O_t / S_t = TC_t I_t$
		ADD	$SA_t = O_t - S_t = TC_t + I_t$
		LOGADD	$SA_t = O_t / \exp(S_t) = \exp(TC_t + I_t)$
		PSEUDOADD	$SA_t = TC_t I_t$

When s is odd, the trend-cycle component is computed from the s -period-centered moving average as follows:

$$TC_t = \sum_{k=-\lfloor s/2 \rfloor}^{\lfloor s/2 \rfloor} y_{t+k}/s$$

When s is even, the trend-cycle component is computed from the s -period-centered moving average as

follows:

$$TC_t = \sum_{k=-s/2}^{s/2-1} (y_{t+k} + y_{t+1+k})/2s$$

The seasonal component is obtained by averaging the seasonal-irregular component for each season,

$$S_{k+js} = \sum_{t=k \bmod s} \frac{SI_t}{T/s}$$

where $0 \leq j \leq T/s$ and $1 \leq k \leq s$. The seasonal components are normalized to sum to 1 (multiplicative) or 0 (additive).

DPF Class

DPF Object

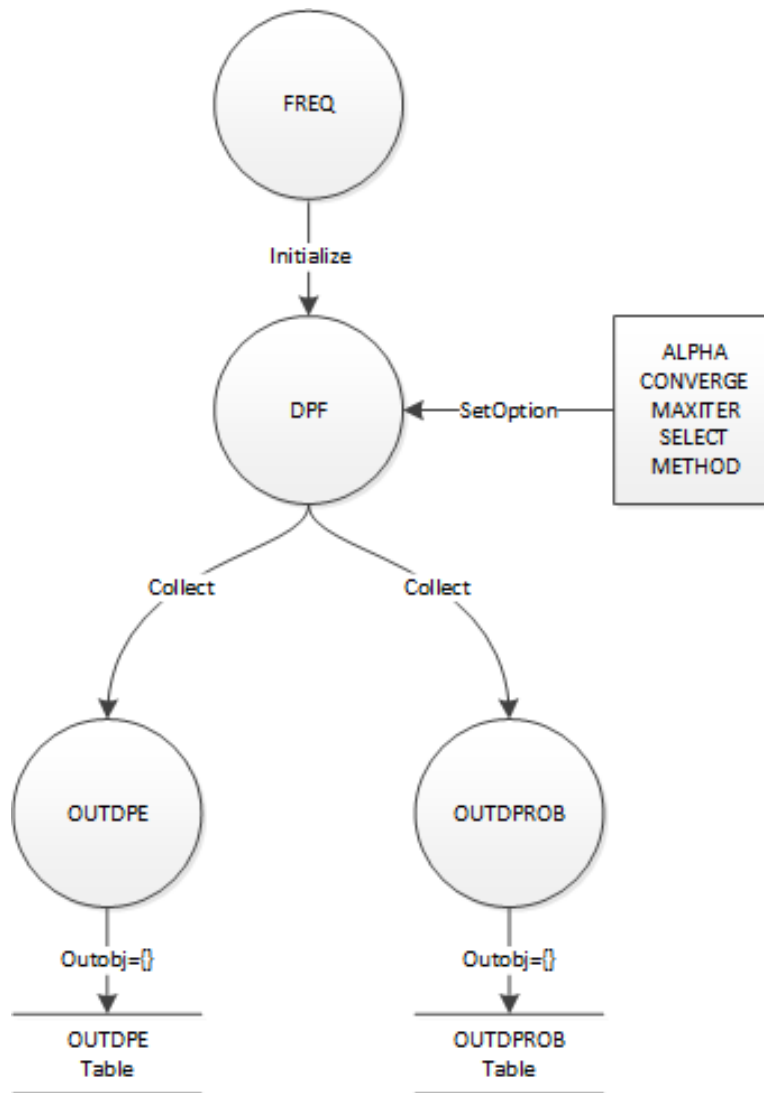
The DPF class object functions take the output of the FREQ class object as input. The DPF class object performs count distribution analysis for a time series.

Table 18.13 summarizes the methods that are associated with the DPF object.

Table 18.13 Methods of the DPF Object

Method	Description
Initialize	Initialize the DPF object
Run	Run the analysis
SetOption	Set the specifications

Figure 18.8 diagrams the methods of the DPF object.

Figure 18.8 DPF Data Flow

DPF Synopsis

DECLARE OBJECT *obj* (DPF) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize (FREQObj) ;
rc=obj.SetOption ('Name',Value) ;
rc=obj.Run () ;

```

DPF Methods

DPF.Initialize Method

```
rc=obj.Initialize (FREQObj) ;
```

Specifies a FREQ object, which includes frequency analysis results. This method must be called before the time series arrays and other attributes are specified for the DPF object.

Input Arguments

You must specify a FREQ object, which includes frequency analysis results.

FREQObj specifies the FREQ object as input.

DPF.Run Method

```
rc=obj.Run () ;
```

Runs the DPF object to perform the count distribution analysis. Upon successful completion, various results can be extracted from the DPF object.

DPF.SetOption Method

```
rc=obj.SetOption ('Name', Value) ;
```

Sets a named option for the DPF object.

Input Arguments

You can specify one of the following for '*Name*' and its associated *Value*:

ALPHA	specifies the confidence level size, where alpha must be between 0 and 1. The default value is 0.05.								
CONVERGE	specifies the convergence criterion.								
MAXITER	specifies the maximum number of iterations, where <i>Value</i> is an integer.								
'SELECT'	takes a string <i>Value</i> that specifies the distribution selection criterion. You can specify one of the following <i>Values</i> : <table data-bbox="500 1386 1242 1533"> <tr> <td>AIC</td><td>specifies Akaike's information criterion.</td></tr> <tr> <td>BIC</td><td>specifies the Bayesian information criterion.</td></tr> <tr> <td>LOGLIK</td><td>specifies the log-likelihood.</td></tr> </table> The default value is LOGLIK.	AIC	specifies Akaike's information criterion.	BIC	specifies the Bayesian information criterion.	LOGLIK	specifies the log-likelihood.		
AIC	specifies Akaike's information criterion.								
BIC	specifies the Bayesian information criterion.								
LOGLIK	specifies the log-likelihood.								
'METHOD'	takes a string <i>Value</i> that specifies the candidate distribution to use in the analysis. You can specify one of the following <i>Values</i> : <table data-bbox="500 1690 1442 1911"> <tr> <td>BEST</td><td>specifies the best distribution, based on the value of the select argument.</td></tr> <tr> <td>BINOMIAL</td><td>specifies the binomial distribution.</td></tr> <tr> <td>GEOMETRIC</td><td>specifies the geometric distribution.</td></tr> <tr> <td>NEGBINOMIAL</td><td>specifies the negative binomial distribution.</td></tr> </table>	BEST	specifies the best distribution, based on the value of the select argument.	BINOMIAL	specifies the binomial distribution.	GEOMETRIC	specifies the geometric distribution.	NEGBINOMIAL	specifies the negative binomial distribution.
BEST	specifies the best distribution, based on the value of the select argument.								
BINOMIAL	specifies the binomial distribution.								
GEOMETRIC	specifies the geometric distribution.								
NEGBINOMIAL	specifies the negative binomial distribution.								

- POISSON** specifies the Poisson distribution.
- ZMBINOMIAL** specifies the zero-modified binomial distribution.
- ZMGEOMETRIC** specifies the zero-modified geometric distribution.
- ZMNEGBINOMIAL** specifies the zero-modified negative binomial distribution.
- ZMPOISSON** specifies the zero-modified Poisson distribution.

The default distribution is BEST.

OUTDPE Object

The OUTDPE object collects output from a DPF object.

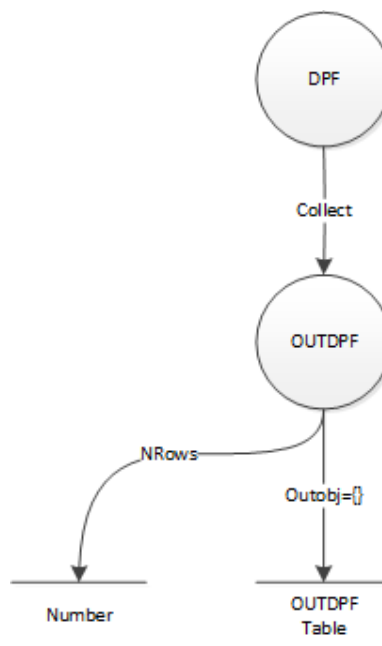
Table 18.14 summarizes the methods that are associated with the OUTDPE object.

Table 18.14 Methods of the OUTDPE Object

Method	Description
<code>Collect</code>	Collect the results of a DPF object
<code>nrows</code>	Get the current row count from the OUTDPE object

Figure 18.9 diagrams the methods of the OUTDPF object.

Figure 18.9 OUTDPE Data Flow



OUTDPE Methods

OUTDPE.Collect Method

```
rc=obj.Collect () ;
```

Collects the results from a DPE object and saves the result to a CAS table.

Arguments There are no arguments associated with this method.

OUTDPE.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTDPE object.

Arguments There are no arguments associated with this method.

Output Table Schema

OUTDPE collects predictions using the count distribution that has the best fit. The output table contains the columns that are shown in [Table 18.15](#).

Table 18.15 OUTDPE Output Table Contents

Column Name	Column Content
NAME	Name of target variable
Distribution	Distribution
Parameter	Name of parameter
Estimate	Parameter estimate
StdErr	Standard error
Tvalue	<i>t</i> value
Probt	Approximate probability > <i>t</i>
Lower	Lower 95% of parameter estimate
Upper	Upper 95% of parameter estimate

OUTDPROB Object

The OUTDPROB object collects output from a DPF object.

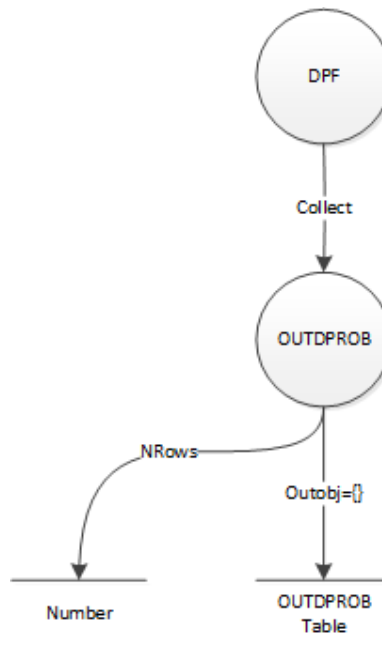
Table 18.16 summarizes the methods that are associated with the OUTDPROB object.

Table 18.16 Methods of the OUTDPROB Object

Method	Description
<code>Collect</code>	Collect the results of a DPF object
<code>nrows</code>	Get the current row count from the OUTDPROB object

Figure 18.10 diagrams the methods of the OUTDPROB object.

Figure 18.10 OUTDPROB Data Flow



OUTDPROB Methods

OUTDPROB.Collect Method

rc=obj.Collect () ;

Collects the results from a DPE object and saves the result to a CAS table.

Arguments There are no arguments associated with this method.

OUTDPROB.nrows Attribute

```
nrows=obj.nrows () ;
```

Gets the current row count from the OUTDPROB object.

Arguments There are no arguments associated with this method.

Output Table Schema

OUTDPROB collects predictions using the count distribution that has the best fit. The output table contains the columns that are shown in Table 18.17.

Table 18.17 OUTDPROB Output Table Contents

Column Name	Column Content
NAME	Name of target variable
Index	Index
Value	Values
Observedzeros	Observed zeros
Expectedzeros	Expected zeros
Observed	Observed counts
Expected	Expected counts
Expectedlower	Expected lower confidence
Expectedupper	Expected upper confidence
Probability	Probability
Probabilitylower	Probability lower confidence
Probabilityupper	Probability upper confidence
Chisquare	Chi-square statistic
Chisquareprob	Chi-square probability
Chisquarelogprob	Chi-square log probability

Example

This example uses the TSMODEL procedure to analyze the frequency of the time series *Air* and then conducts a count distribution analysis.

The following DATA step loads the *Sashelp.Air* data set onto the CAS server. This DATA step assumes that your CAS engine libref is named *mycas*, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air (replace=yes);
    set Sashelp.air;
run;
```



```

proc tsmodel data=mycas.air outobj=(of=mycas.outfreq(replace=YES)
                                odpe=mycas.dpe(replace=YES)
                                odprob=mycas.dprob(replace=YES)
                                );

var air;
id date interval=month;
require tsa;
submit;
  declare object f(FREQ);
  declare object of(OUTFREQ);
  declare object dpf(DPF);
  declare object odpe(OUTDPE);
  declare object odprob(OUTDPROB);
  rc = f.Initialize();
  rc = f.SetOption("SEASONALITY",12);
  rc = f.SetY(air);
  rc = f.Run();
  rc = of.Collect(f);

  rc = dpf.Initialize(f);
  rc = dpf.SetOption("SELECT","AIC","METHOD","POISSON");
  rc = dpf.Run();
  rc = odpe.Collect(dpf);
  rc = odprob.Collect(dpf);
endsubmit;
run;

```

FREQ Class

FREQ Object

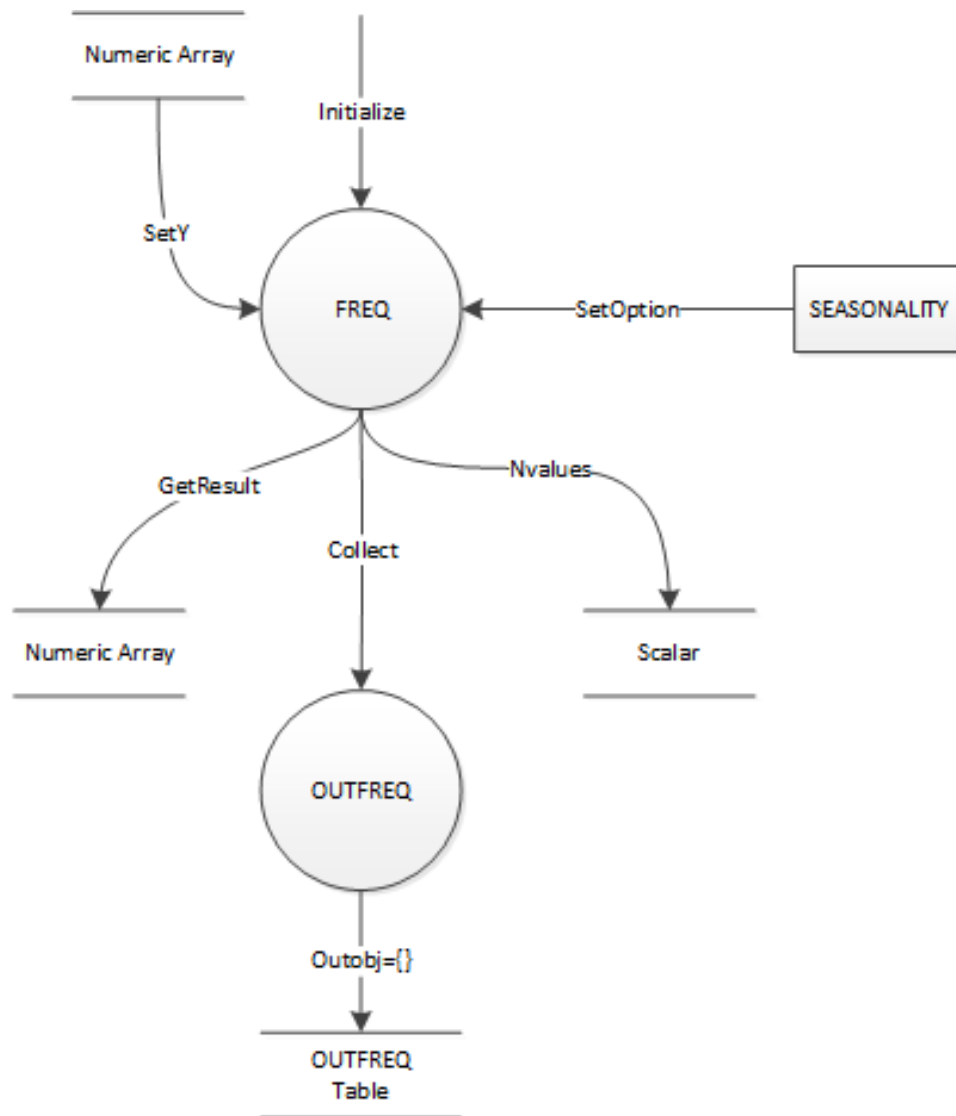
The FREQ class object function analyzes the frequency of a time series and outputs all unique values and corresponding counts for the time series.

Table 18.18 summarizes the methods that are associated with the FREQ object.

Table 18.18 Methods of the FREQ Object

Method	Description
GetResult	Get the analysis result
Initialize	Initialize the FREQ object
Nvalues	Return the number of unique values in the input series
Run	Run the analysis
SetOption	Set the specifications
SetY	Set the input time series

Figure 18.11 diagrams the methods of the FREQ object.

Figure 18.11 FREQ Data Flow

FREQ Synopsis

DECLARE OBJECT *obj* (FREQ) ;

Method syntax, in order of typical usage:

```

rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.SetOption ('Name', Value) ;
rc=obj.Run () ;
rc=obj.GetResult ('Name', OutputArray) ;
rc=obj.Nvalues () ;

```

FREQ Methods

FREQ.GetResult Method

```
rc=obj.GetResult ('Name',OutputArray) ;
```

Outputs the selected results to an array.

Input Arguments You must specify the following input arguments:

Name specifies a string that identifies the result series to return. You can specify the following values within single quotation marks:

'VALUES'	retrieves a monotonically increasing list of the unique values that are found in the input series that is specified by using the SetY method.
'COUNTS'	retrieves the number of occurrences of each unique value.
'PCTS'	retrieves the percentage of total occurrences of each unique value.

OutputArray specifies a numeric array to receive the result series.

FREQ.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes an FREQ object to an empty state. This method must be called before the time series arrays and other attributes are specified for the FREQ object.

Arguments There are no arguments associated with this method.

FREQ.Nvalues Method

```
rc=obj.Nvalues () ;
```

Returns the number of unique values in the input series to *rc*. This number is also the length of all result series that are returned by the GetResult method. It is set to a missing value if the Run method did not complete successfully.

Arguments There are no arguments associated with this method.

FREQ.Run Method

```
rc=obj.Run () ;
```

Runs the FREQ object to perform the autocorrelation and autocovariance analysis by using the time series array *YSeries* that has been specified for it. Upon successful completion, various results can be extracted from the FREQ object.

FREQ.SetOption Method

```
rc=obj.SetOption ('Name',Value) ;
```

Sets a named option for the FREQ object.

Input Arguments

You can specify the following for '*Name*' and its associated *Value*:

'SEASONALITY' specifies the seasonality of the time series, where *Value* is an integer. The default value is 0.

FREQ.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the time series array (*YSeries*) for the FREQ object.

Input Arguments You must specify the following input argument:

YSeries specifies the time series array to analyze.

OUTFREQ Object

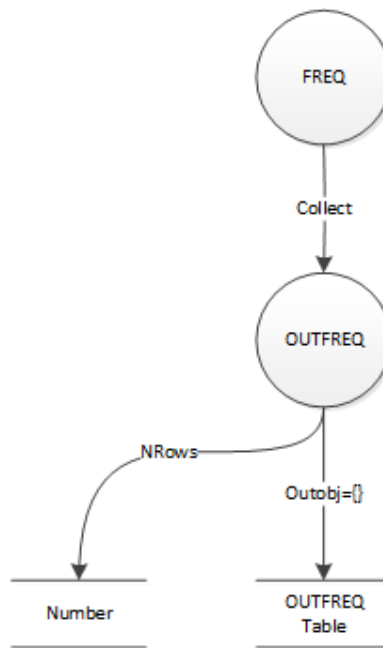
The OUTFREQ object collects output from a FREQ object.

Table 18.19 summarizes the methods that are associated with the OUTFREQ object.

Table 18.19 Methods of the OUTFREQ Object

Method	Description
Collect	Collect the results of a FREQ object
nrows	Get the current row count from the OUTFREQ object

Figure 18.12 diagrams the methods of the OUTFREQ object.

Figure 18.12 OUTFREQ Data Flow

OUTFREQ Methods

OUTFREQ.Collect Method

`rc=obj.Collect () ;`

Collects the results from a FREQ object and saves the result to a CAS table.

Arguments There are no arguments associated with this method.

OUTFREQ.nrows Attribute

`nrows=obj.nrows () ;`

Gets the current row count from the OUTFREQ object.

Arguments There are no arguments associated with this method.

Output Table Schema

The collector object, OUTFREQ, collects the results of frequency function.

OUTFREQ collects the results of frequency function. The output table contains the columns that are shown in [Table 18.20](#).

Table 18.20 OUTFREQ Output Table Contents

Column Name	Column Content
NAME	Name of target variable
Values	Array of series values
Counts	Array of frequency series counts
Percent	Percentage of total frequency

Example

This example uses the TSMODEL procedure to analyze the frequency of the time series Air:

```
proc tsmodel data=mycas.air outobj=(of=mycas.outfreq(replace=YES));
  var air;
  id date interval=month;
  require tsa;
  submit;
  declare object f(FREQ);
  declare object of(OUTFREQ);
  rc = f.Initialize();
  rc = f.SetOption("SEASONALITY",12);
  rc = f.SetY(air);
  rc = f.Run();
  rc = of.Collect(f);
endsubmit;
run;
```

TSA Object

Functional Summary

NOTE: Each function in this section has a prefix of “TSA.”; however, the prefixes are omitted in descriptions for better readability.

Table 18.21 summarizes the functions in the TSA package.

Table 18.21 Functions in Time Series Analysis Package

TSA Function	Description
ACCUMULATE	Accumulates a univariate time series to a particular frequency
ACCUMULATE2	Accumulates a high-frequency time series to a lower frequency and expands the lower-frequency series to have the same length as the high-frequency series

Table 18.21 *continued*

TSA Function	Description
ACF	Computes autocorrelation and autocovariance for a time series array
ARMAORDERS	Performs tests to tentatively identify the autoregressive and moving average orders of mixed autoregressive moving average models
CCF	Computes the cross-correlation and cross-covariance for two time series arrays
DPF Class Object	Performs count distribution analysis for time series
FREQ Class Object	Performs frequency analysis of a time series
INTERMITTENCYTEST	Tests for intermittency of a univariate time series
IACF	Computes the inverse autocorrelation for a time series array
MOVINGSUMMARY	Computes statistics for a set of values within a moving time window
PACF	Computes the partial autocorrelation for a time series array
SCALE	Scales a time series between the minimum value and the maximum value of the original time series
SEASONALDECOMP	Computes the seasonal indices of a univariate time series using classical decomposition
SEASONALINDICES	Computes the seasonal indices of a univariate time series by using regression seasonal dummies
SEASONTTEST	Tests for seasonality of a univariate time series
SIMILARITY	Performs similarity analysis for time series
STATIONARITYTEST	Tests for stationarity of a univariate time series
TRANSFORM	Transforms time series according to the specified transformation type
UNBIASEDNESS	Tests whether a univariate time series is unbiased
WHITENOISE	Tests for white noise of a time series array

ACCUMULATE Method

`rc = TSA.ACCUMULATE (time, y, 'interval', id, z, <'accumulate'>, <'setmiss'>, <'zeromiss'>);`

The ACCUMULATE function accumulates a univariate time series to a particular frequency.

Required Arguments

You must specify the following arguments, separated by commas:

time specifies the time ID array for the time series.
y specifies the times series array to accumulate.
'interval' specifies the time interval.

You can specify the following values within single quotation marks:

DAY specifies a seasonal cycle of length 7.
MONTH specifies a seasonal cycle of length 12.
QTR specifies a seasonal cycle of length 4.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'accumulate' specifies the accumulation statistic.

You can specify the following values within single quotation marks:

AVERAGE | AVG specifies the average of the values in the time series.
CSS specifies the corrected sum of squares of the values in the time series.
FIRST specifies the first value of the time series.
LAST specifies the last value of the time series.
MAXIMUM | MAX specifies the maximum value in the time series.
MEDIAN | MED specifies the median of the values in the time series.
MINIMUM | MIN specifies the minimum value in the time series.
N specifies the number of nonmissing observations.
NMISS specifies the number of missing observations.
NOBS specifies the number of observations.
STDDEV | STD specifies the standard deviation of the values in the time series.
TOTAL specifies the total sum of the values in the time series.
USS specifies the uncorrected sum of squares of the values in the time series.

The default is TOTAL.

'setmiss' specifies the missing value interpretation.

You can specify the following values within single quotation marks:

AVERAGE AVG	specifies the accumulated average value.
FIRST	specifies the accumulated first nonmissing value.
MAXIMUM MAX	specifies the accumulated maximum value.
MEDIAN MED	specifies the accumulated median value.
MINIMUM MIN	specifies the accumulated minimum value.
MISSING	specifies a missing value.
NEXT	specifies the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.
PREVIOUS PREV	specifies the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.

The default is MISSING.

'zeromiss' specifies the zero value interpretation.

You can specify the following values within single quotation marks:

BOTH	sets both beginning and ending zeros to missing.
LEFT	sets beginning zeros to missing.
NONE	leaves beginning and ending zeros unchanged.
RIGHT	sets ending zeros to missing.

The default is NONE.

Returned Values

The ACCUMULATE function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

id returns the time ID array for the accumulated time series.

z returns the accumulated time series array.

Example

This example uses the TSMODEL procedure to accumulate a time series:

```

proc tsmodel data=mycas.air outarray=mycas.outarray;
  id date interval=month;
  var air;
  outarrays qtravg new_id;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACCUMULATE(date, air, 'QTR', new_id, qtravg, 'AVERAGE', , );
  endsubmit;
run;

```

ACCUMULATE2 Method

`rc = TSA.ACCUMULATE2 (time, y, 'interval', id, z, <'accumulate'>, <'setmiss'>, <'zeromiss'>);`

The ACCUMULATE2 function accumulates a high-frequency time series to a lower frequency and expands the lower-frequency time series to the same length as the high-frequency series.

Required Arguments

You must specify the following arguments, separated by commas:

time specifies the time ID array for the time series.
y specifies the times series array to accumulate.
'interval' specifies the time interval.

You can specify the following values within single quotation marks:

DAY specifies a seasonal cycle of length 7.
MONTH specifies a seasonal cycle of length 12.
QTR specifies a seasonal cycle of length 4.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'accumulate' specifies the accumulation statistic.

You can specify the following values within single quotation marks:

AVERAGE | AVG specifies the average of the values in the time series.
CSS specifies the corrected sum of squares of the values in the time series.
FIRST specifies the first value of the time series.
LAST specifies the last value of the time series.
MAXIMUM | MAX specifies the maximum value in the time series.

MEDIAN MED	specifies the median of the values in the time series.
MINIMUM MIN	specifies the minimum value in the time series.
N	specifies the number of nonmissing observations.
NMISS	specifies the number of missing observations.
NOBS	specifies the number of observations.
STDDEV STD	specifies the standard deviation of the values in the time series.
TOTAL	specifies the total sum of the values in the time series.
USS	specifies the uncorrected sum of squares of the values in the time series.

The default is TOTAL.

'setmiss' specifies the missing value interpretation.

You can specify the following values within single quotation marks:

AVERAGE AVG	specifies the accumulated average value.
FIRST	specifies the accumulated first nonmissing value.
LAST	specifies the accumulated last nonmissing value.
MAXIMUM MAX	specifies the accumulated maximum value.
MEDIAN MED	specifies the accumulated median value.
MINIMUM MIN	specifies the accumulated minimum value.
MISSING	specifies a missing value.
NEXT	specifies the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.
PREVIOUS PREV	specifies the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.

The default is MISSING.

'zeromiss' specifies the zero value interpretation.

You can specify the following values within single quotation marks:

BOTH	sets both beginning and ending zeros to missing.
LEFT	sets beginning zeros to missing.
NONE	leaves beginning and ending zeros unchanged.
RIGHT	sets ending zeros to missing.

The default is NONE.

Returned Values

The ACCUMULATE2 function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success.
< 0	Computational failure.

id returns the time ID array for the accumulated time series.

z returns the accumulated time series array.

Example

This example uses the TSMODEL procedure to accumulate a monthly time series into a yearly time series:

```
proc tsmodel data=mycas.air outarray=mycas.outarray;
  id date interval=month;
  var air;
  outarrays yearavg_expand yearid_expand;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACCUMULATE2(date, air, 'YEAR', yearid_expand, yearavg_expand,
                    'AVERAGE', , );
  endsubmit;
run;
```

ACF Method

rc = **TSA.ACF** (*y*, *nlag*, *lags*, *df*, < *mu* >, < *acov* >, < *acf* >, < *acfstd* >, < *acf2std* >, < *acfnorm* >, < *acfprob* >
< *acflprob* >);

The ACF function computes autocorrelation and autocovariance for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

y specifies the times series array.

nlag specifies the number of the lag to use in the calculation.

Returned Values

The ACF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

lags returns the number of the lags that were used in the calculation.

df returns the number of observations used to compute *acov* and *acf*.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

mu returns the mean estimate.

acov returns an array of covariance estimates, with *nlag*+1 entries.

acf returns an array of autocorrelation estimates, with *nlag*+1 entries.

acfstd returns an array of standard errors, with *nlag*+1 entries.

acf2std returns an array of twice standard errors, with *nlag*+1 entries.

acfnorm returns an array of normalized autocorrelation, with *nlag*+1 entries.

acfprob returns an array of autocorrelation probabilities, with *nlag*+1 entries.

acflprob returns an array of autocorrelation log probabilities, with *nlag*+1 entries.

Example

This example uses the TSMODEL procedure to compute the autocorrelation of lag 3 of the time series Air:

```
proc tsmodel data=mycas.air outscalar=mycas.outscalars
      outarray=mycas.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays acf acov lags df acfstd;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.ACF(air, 3, lags, df, mu, acov, acf, acfstd, , , , );
  endsubmit;
run;
```

ARMAORDERS Method

`rc = TSA.ARMAORDERS (y, < dif>, < 'method' >, < p>, < q>, < perror>, porders, qorders) ;`

The ARMAORDERS function performs tests to tentatively identify the autoregressive and moving average orders of mixed autoregressive moving average models.

Required Arguments

You must specify the following argument:

y specifies the times series array to test.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

dif specifies either an array of positive integers or a positive integer that is used for differencing. The default value is 0.

'method' specifies the method of tentative order selection.

You can specify the following values within single quotation marks:

ESACF specifies the extended sample autocorrelation function.

MINIC specifies the minimum information criterion.

SCAN specifies the squared canonical correlations.

The default value is MINIC.

p specifies the autoregressive order range, where *p* is an array of two nonnegative integers that define the minimum and maximum values.

By default, *p* is the array [0,5].

q specifies the moving average order range, where *q* is an array of two nonnegative integers that define the minimum and maximum values.

By default, *q* is the array [0,5].

peror specifies the autoregressive orders used to estimate the error series for the MINIC method, where *peror* is an array of two nonnegative integers that define the minimum and maximum values.

By default, *peror* is the array [max(*p*), max(*p*)+max(*q*)].

Returned Values

The ARMAORDERS function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

porders returns the recommended autoregressive orders.

qorders returns the recommended moving average orders.

Example

This example uses the TSMODEL procedure to tentatively identify the autoregressive and moving average orders for the time series Air:

```
proc tsmode1 data=mycas.air outscalar=mycas.outscalars;
  id date interval=month;
  var air;
  outscalars porders qorders;
  require tsa;
  submit;
  declare object TSA(tsa);
  porders = 0;
  qorders = 0;
  Array P[2]/nosymbols; P[1]=0; P[2]=5;
  Array Q[2]/nosymbols; Q[1]=0; Q[2]=5;
  rc=TSA.ARMAORDERS(air, 0, 'SCAN', P, Q, , porders, qorders);
  rc=TSA.ARMAORDERS(air, 1, 'ESACF', P, Q, , porders, qorders);
  rc=TSA.ARMAORDERS(air, 1, 'MINIC', P, Q, ,porders, qorders);
  endsubmit;
run;
```

CCF Method

rc = **TSA.CCF** (*y*, *x*, *nlag*, *lags*, *df*, <*ymu*>, <*xmu*>, <*ccov*>, <*ccf*>, <*ccfst*>, <*ccf2std*>, <*ccfnorm*>, <*ccfprob*>, <*ccflprob*>);

The CCF function computes the cross-correlation and cross-covariance for two time series arrays.

Required Arguments

You must specify the following arguments, separated by commas:

- y* specifies one times series array.
- x* specifies the other time series array.
- nlag* specifies the number of the lag to compute.

Returned Values

The CCF function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

- lags* returns an array of lags that were computed, with $nlag+1$ entries.
- df* returns an array of number of products for which to compute the cross-correlation, with $nlag+1$ entries.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

- ymu* returns the mean estimate of input time series *y*.
- xmu* returns the mean estimate of input time series *x*.
- ccov* returns an array of cross-covariance estimates, with $2 \times nlag + 1$ entries.
- ccf* returns an array of cross-correlation estimates, with $2 \times nlag + 1$ entries.
- ccfst* returns an array of standard errors, with $2 \times nlag + 1$ entries.
- ccf2std* returns an array of double standard errors, with $2 \times nlag + 1$ entries.
- ccfnorm* returns an array of normalized cross-correlation, with $2 \times nlag + 1$ entries.
- ccfprob* returns an array of probabilities, with $2 \times nlag + 1$ entries.
- ccflprob* returns an array of log probabilities, with $2 \times nlag + 1$ entries.

Example

This example uses the TSMODEL procedure to compute the cross-correlation and cross-covariance of two time series arrays (Price and Sale) with lag 20:

```
data mycas.pricedata (replace=yes);
  set Sashelp.pricedata;
run;

proc tsmodel data=mycas.pricedata outarray=mycas.ccf_array
             outscalar=mycas.ccf_scalar;
  id date interval=month;
  var price sale;
  by region line product;
  outscalars ymu xmu;
  outarrays lags df ccov ccf ccfstd ccf2std ccfnorm ccflprob;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.CCF(price, sale, 20, lags, df, ymu, xmu, ccov, ccf,
             ccfstd, ccf2std, ccfnorm, ccflprob);
  endsubmit;
run;
```

IACF Method

```
rc = TSA.IACF (y, nlag, lags, df, <mu>, <iacf>, <iacfstd>, <iacf2std>, <iacfnorm>, <iacfprob>,
               <iacflprob>);
```

The IACF function computes the inverse autocorrelation for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array.
- nlag* specifies the number of the lag to use in the calculation.

Returned Values

The IACF function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

- lags* returns the number of the lag that was used in the calculation.

df returns the number of observations used to compute *iacf*.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

mu returns the mean estimate.
iacf returns an array of inverse autocorrelation estimates, with *nlag*+1 entries.
iacfstd returns an array of inverse autocorrelation standard errors, with *nlag*+1 entries.
iacf2std returns an array of twice standard errors, with *nlag*+1 entries.
iacfnorm returns an array of normalized inverse autocorrelation, with *nlag*+1 entries.
iacfprob returns an array of inverse autocorrelation probabilities, with *nlag*+1 entries.
iacfprob returns an array of inverse autocorrelation log probabilities, with *nlag*+1 entries.

Example

This example uses the TSMODEL procedure to compute the inverse autocorrelation of lag 3 of the time series Air:

```
proc tsmode1 data=mycas.air outscalar=mycas.outscalars
      outarray=mycas.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays iacf lags df iacfstd;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.IACF(air, 3, lags, df, mu, iacf, iacfstd, , , , );
  endsubmit;
run;
```

INTERMITTENCYTEST Method

rc = **TSA.INTERMITTENCYTEST** (*y*, *base*, *threshold*, *med*) ;

The INTERMITTENCYTEST function tests for intermittency of a univariate time series by computing the median of the length of contiguous constant periods (demand intervals).

Required Arguments

You must specify the following arguments, separated by commas:

- y* specifies the times series array to test. The test is applied to the last 100 values.
- base* specifies the base value to test. The value is typically 0.
- threshold* specifies the threshold value for intermittency. The value is typically greater than 2.

Returned Values

The INTERMITTENCYTEST function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is not intermittent
1	Time series is intermittent
< 0	Computational failure

- med* returns the median length of the contiguous constant periods.

Example

This example uses the TSMODEL procedure to test the intermittency on the time series array Air:

```
proc tsmodel data=mycas.air outscalar=mycas.outscalars;
  id date interval=month;
  var air;
  outscalars intermittent;
  require tsa;
  submit;
  declare object TSA(tsa);
  intermittent=0;
  rc=TSA.INTERMITTENCYTEST(air, 0, 2, med);
  if rc>0 then intermittent= 1;
  endsubmit;
run;
```

MOVINGSUMMARY Method

```
rc = TSA.MOVINGSUMMARY (y, 'method', k, <lead>, <w>, <'setmiss'>, <'abs'>, x, <p>, <nmiss>)
;
```

The MOVINGSUMMARY function computes statistics for a set of values within a moving time window.

Required Arguments

You must specify the following arguments, separated by commas:

<i>y</i>	specifies the input time series array.
'method'	specifies the statistic to calculate for each output array, x_t , based on the elements of the <i>y</i> input array in the <i>t</i> window. You can specify the following methods within single quotation marks:
EWMA	calculates the exponentially weighted moving average.
GMEAN	calculates the moving geometric mean.
MAX	calculates the maximum value.
MEAN	calculates the moving average.
MED	calculates the median value.
MIN	calculates the minimum value.
PROD	calculates the moving product.
RANGE	calculates the maximum value minus minimum value.
SUM	calculates the moving sum.
TVALUE	calculates the standard deviation divided by mean.
VAR	calculates the variance of the sample defined by the window around <i>t</i> .
<i>k</i>	specifies the window size, where <i>k</i> is a positive integer. When the method is EWMA, <i>k</i> is set to 1 and defaults are used for all other arguments.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

<i>lead</i>	specifies the number of leading terms, where <i>lead</i> is a nonnegative integer less than <i>k</i> . You can specify the following values:
0	specifies the backward moving summary.
<i>k</i> /2	specifies the centered moving summary.
<i>k</i> −1	specifies the forward moving summary.

The default value is 0. When the method is EWMA, *lead* is set to 0.

w specifies an array of weights that has k elements (a scalar when $k=1$). This argument is required for the EWMA method, and it must be a scalar between 0 and 1, inclusive. This argument is optional for the MEAN, PROD, TVALUE, and VAR methods and is not supported for all other methods.

'*setmiss*' specifies how missing values are interpreted.

You can specify the following values within single quotation marks:

IGNORE specifies that missing values have no effect on the summary.

MEAN specifies that missing values are replaced with the mean of the remaining nonmissing values in the window. This value is supported only for the method SUM.

MISSING specifies that if the input window contains a missing value, the output value is also missing.

The default value is IGNORE.

'*abs*' specifies how the series is transformed into nonnegative values prior to performing the moving summary.

You can specify the following values within single quotation marks:

OFF specifies no modification. This value is not supported for the GMEAN method, because the geometric mean is undefined for negative values in the series.

ON transforms each member of the series into its absolute value.

SQUARE transforms each member of the series into its square

The default value is ON when the *method* is GMEAN and is OFF for all other *methods*.

Returned Values

The MOVINGSUMMARY function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
4	Warping limits relaxed
3	Expansion limits relaxed
2	Compression limits relaxed
1	Warping limits imposed
0	Success
< 0	Computational failure

x returns the transformed series.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

- p* returns an array in which element *t* is the number of products that contributed to element *t* of *x*. The *p* argument is supported only when *method* is PROD or GMEAN.
- nmiss* returns the number of missing values that are generated.

Examples

This example uses the TSMODEL procedure to compute the five-period moving average of the time series array Air:

```
proc tsmodel data=mycas.air outscalar=mycas.scalars outarray=mycas.arrays;
  id date interval=month;
  var air;
  outarrays x p;
  outscalars rc nmiss;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.MOVINGSUMMARY(air, 'MEAN' , 5, 0 , , , , x, p, nmiss);
  endsubmit;
run;
```

This example uses the TSMODEL procedure to compute the five-period centered weighted moving product of the time series array Air:

```
proc tsmodel data=mycas.air outscalar=mycas.scalars outarray=mycas.arrays;
  id date interval=month;
  var air;
  outarrays x;
  outscalars rc;
  require tsa;
  submit;
  declare object TSA(tsa);
  Array w[5]/nosymbols; w[1]=0.3; w[2]=0.2; w[3]=0.25; w[4]=0.1; w[5]=0.15;
  rc=TSA.MOVINGSUMMARY(air, 'PROD' , 5, 2.5 , w, , , x, , );
  endsubmit;
run;
```

PACF Method

`rc = TSA.PACF (y, nlag, lags, df, < mu>, < pacf>, < pacfst>, < pacf2std>, < pacfnorm>, < pacfprob>, < pacflprob>);`

The PACF function computes the partial autocorrelation for a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array.
- nlag* specifies the number of the lag to use in the calculation.

Returned Values

The PACF function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

- lags* returns the number of the lag that was used in the calculation.
- df* returns the number of observations used to compute *pacf*.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

- mu* returns the mean estimate.
- pacf* returns an array of partial autocorrelation estimates, with *nlag*+1 entries.
- pacfst* returns an array of partial autocorrelation standard errors, with *nlag*+1 entries.
- pacf2std* returns an array of twice standard errors, with *nlag*+1 entries.
- pacfnorm* returns an array of normalized partial autocorrelation, with *nlag*+1 entries.
- pacfprob* returns an array of partial autocorrelation probabilities, with *nlag*+1 entries.
- pacflprob* returns an array of partial autocorrelation log probabilities, with *nlag*+1 entries.

Example

This example uses the TSMODEL procedure to compute the autocorrelation of lag 3 of the time series Air:

```
proc tsmodel data=mycas.air outscalar=mycas.outscalars
              outarray=mycas.outarray;
  id date interval=month;
  var air;
  outscalars mu;
  outarrays pacf lags df pacfstd;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.PACF(air, 3, lags, df, mu, pacf, pacfstd, , , , );
  endsubmit;
run;
```

SCALE Method

$rc = \text{TSA.SCALE}(y, \text{min}, \text{max}, \text{nomiss}, x, <\text{nmiss}>);$

The SCALE function scales a time series between a specified minimum value and a specified maximum value.

Required Arguments

You must specify the following arguments, separated by commas:

- | | |
|---------------|---|
| <i>y</i> | specifies the input time series array. |
| <i>min</i> | specifies the minimum value in the output array. |
| <i>max</i> | specifies the maximum value in the output array. |
| <i>nomiss</i> | specifies how missing values are treated. You can specify the following values: |
| 0 | allows missing values in the input array. |
| 1 | does not allow missing values in the input array. If missing values exist, the output array x_t becomes missing for all values of t . |

The default is 0.

Returned Values

The SCALE function returns the following values:

- | | |
|-----------|---|
| <i>rc</i> | returns one of the following scalar return codes: |
|-----------|---|

<i>rc</i>	Termination Reason
5	The input series is nearly constant
4	Missing values were found when the value 1 was specified for <i>nomiss</i>
3	One or more arguments are ignored
2	One or more arguments are set to the default value
1	The input series is all missing
-1	One or more arguments are not supported
-2	The minimum value of the transformed series is greater than its maximum value
-4	Extreme slope
-99	Bad arguments

x returns the transformed series.

Optional Returned Values

You can also specify the following argument to request an additional returned value:

nmis returns the number of missing values that are generated.

Example

This example uses the TSMODEL procedure to scale the time series array *Air* between a minimum value of 0 and a maximum value of 100:

```
proc tsmode1 data=mycas.air outarray=mycas.scale_array;
  id date interval=month;
  var air;
  outarrays t1;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.SCALE(air, 0, 100, , t1, );
  endsubmit;
run;
```

SEASONALDECOMP Method

```
rc = TSA.SEASONALDECOMP (y, s, 'mode', <lambda>, <tcc>, <sic>, <sc>, <scstd>, <tcs>,
                           <ic>, <sa>, <pcsa>, <tc>, <cc>) ;
```

The SEASONALDECOMP function computes the seasonal indices of a univariate time series by using classical decomposition.

Required Arguments

You must specify the following arguments, separated by commas:

- y* specifies the times series array to decompose.
- s* specifies the seasonality to test, where *s* must be either a positive integer or `_SEASONALITY_`, which is the length of the seasonal cycle as specified by the `SEASONALITY=` option in the `PROC TSMODEL` statement or implied by the `INTERVAL=` option in the `ID` statement.
- 'mode' specifies the type of decomposition to be used to decompose the time series.
You can specify the following values within single quotation marks:
 - ADD | ADDITIVE** specifies additive decomposition.
 - LOGADD | LOGADDITIVE** specifies log-additive decomposition.
 - MULT | MULTIPLICATIVE** specifies multiplicative decomposition.
 - MULTORADD** specifies multiplicative or additive decomposition, depending on data.
 - PSEUDOADD | PSEUDOADDITIVE** specifies pseudo-additive decomposition.

Optional Arguments

You can also specify the following argument, separated by a comma from arguments that precede it. If you want to use a default value for this argument, enter a space for it.

- lambda* specifies the Hodrick-Prescott filter parameter for trend-cycle decomposition. The default value is 1,600. Filtering applies when the trend component or the cycle component is requested. If filtering is not specified, this option is ignored.

Returned Values

The SEASONALDECOMP function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

<i>tcc</i>	specifies the trend-cycle component.
<i>sic</i>	specifies the seasonal-irregular component.
<i>sc</i>	specifies the seasonal component.
<i>scstd</i>	specifies the seasonal component standard errors.
<i>tcs</i>	specifies the trend-cycle-seasonal component.
<i>ic</i>	specifies the irregular component.
<i>sa</i>	specifies the seasonally adjusted series.
<i>pcsa</i>	specifies the percentage of change in seasonally adjusted series.
<i>tc</i>	specifies the trend component.
<i>cc</i>	specifies the cycle component.

Example

This example uses the TSMODEL procedure to compute the seasonal indices on the time series array Air:

```
proc tsmodel data=mycas.air outarray=mycas.outarray;
  id date interval=month;
  var air;
  outarrays ADJUSTED;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.SEASONALDECOMP(air, _SEASONALITY_, 'ADD', , , , , , , ADJUSTED, , , );
  endsubmit;
run;
```

SEASONALINDICES Method

rc = TSA.SEASONALINDICES (*y*, *s*, <'mode'>, <'term'>, *indices*) ;

The SEASONALINDICES function computes the seasonal indices of a univariate time series by using regression seasonal dummies.

Required Arguments

You must specify the following arguments, separated by a comma:

<i>y</i>	specifies the times series array.
<i>s</i>	specifies the seasonality to test, where <i>s</i> must be either a positive integer or <code>_SEASONALITY_</code> , which is the length of the seasonal cycle as specified by the <code>SEASONALITY=</code> option in the PROC TSMODEL statement or implied by the <code>INTERVAL=</code> option in the ID statement.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'mode' specifies the type of model to be used in the regression.
You can specify the following values within single quotation marks:

ADD | ADDITIVE uses an additive model.
MULT | MULTIPLICATIVE uses a multiplicative model.

The default method is ADD.

'term' specifies the type of terms to be used in the regression.
You can specify the following values within single quotation marks:

S uses only seasonal dummies terms.
SC uses only seasonal dummies and constant terms.
ST uses only seasonal dummies and trend terms.
STC uses seasonal dummies, trend, and constant terms.
STQ uses seasonal dummies, trend, and quadratic terms.
STQC uses seasonal dummies, trend, quadratic, and constant terms.

The default value is S. Quadratic values can be used only in the additive model.

Returned Values

The SEASONALINDICES function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

indices returns an array of seasonal indices.

Example

This example uses the TSMODEL procedure to compute the seasonal indices of the time series Air:

```
proc tsmode1 data=mycas.air outarray=mycas.outarray;
  id date interval=month;
  var air;
  outarrays sindices;
  require tsa;
  submit;
```

```

declare object TSA(tsa);
rc=TSA.SEASONALINDICES(air, _SEASONALITY_, 'ADD', 'STQC', sindices);
endsubmit;
run;

```

SEASONTEST Method

***rc* = TSA.SEASONTEST (*y*, *s*, < *dif*>, < *p*>, < *alpha*>, < *aic*>);**

The SEASONTEST function tests whether a univariate time series is seasonal by comparing two time series models: one seasonal and one nonseasonal.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the times series array to test.
- s* specifies the seasonality to test, where *s* must be either a positive integer or `_SEASONALITY_`, which is the length of the seasonal cycle as specified by the `SEASONALITY=` option in the `PROC TSMODEL` statement or implied by the `INTERVAL=` option in the `ID` statement.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

- dif* specifies an array of positive integers or a positive integer that is used for differencing. The default value is 0.
- p* specifies the autoregressive order (0 or 1). The default value is 0.
- alpha* specifies the significance level. The default value is 0.01.

Returned Values

The SEASONTEST function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is not seasonal
1	Time series is seasonal
< 0	Computational failure

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

aic returns an array of three values: Akaike's information criterion (AIC) for the nonseasonal model, AIC for seasonal model, and the p -value for the F test.

Example

The following example uses the TSMODEL procedure to test the seasonality of the time series array Air:

```
proc tsmodel data=mycas.air outscalar=mycas.outscalars
      outarray=mycas.outarray;
  id date interval=month;
  var air;
  outscalars seasonal;
  outarrays aic;
  require tsa;
  submit;
  declare object TSA(tsa);
  seasonal=0;
  rc=TSA.SEASONTTEST(air, _SEASONALITY_, 0, 1, , aic); /*- no detrending -*/
  if rc>0 then seasonal= 1;
  rc=TSA.SEASONTTEST(air, _SEASONALITY_, 1, 1, 0.05, ); /*- detrending -*/
  if rc>0 then seasonal= 1;
  endsubmit;
run;
```

SIMILARITY Method

$rc = \text{TSA.SIMILARITY}(x, y, \text{'type'}, \text{'scale'}, \text{expandpct}, \text{expandabs}, \text{compresspct}, \text{compressabs}, \text{measure});$

The SIMILARITY function analyzes the similarity between two time series.

Required Arguments

You must specify the following arguments, separated by a comma:

x specifies the input time series array to be compared to the target time series.
y specifies the target time series array to be compared to the input time series.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

'type' specifies the similarity measure.

You can specify the following values within single quotation marks:

ABSDEV	specifies the absolute deviation.
MABSDEV	specifies the mean absolute deviation.
MABSDEVINP	specifies the mean absolute deviation relative to the length of the input sequence.
MABSDEVMAX	specifies the mean absolute deviation relative to the maximum valid path length.
MABSDEVMIN	specifies the mean absolute deviation relative to the minimum valid path length.
MABSDEVTAR	specifies the mean absolute deviation relative to the length of the target sequence.
MSQRDEV	specifies the mean squared deviation.
MSQRDEVINP	specifies the mean squared deviation relative to the length of the input sequence.
MSQRDEVMAX	specifies the mean squared deviation relative to the maximum valid path length.
MSQRDEVMIN	specifies the mean squared deviation relative to the minimum valid path length.
MSQRDEVTAR	specifies the mean squared deviation relative to the length of the target sequence.
SQRDEV	specifies the squared deviation.

The default value is **SQRDEV**.

'scale' specifies how the working input sequence is scaled with respect to the working target sequence. Scaling is performed after normalization.

You can specify the following values within single quotation marks:

ABS	applies absolute scaling.
NONE	applies no scaling.
STD	applies standard scaling.

The default value is **NONE**.

expandpct specifies the warping expansion as a percentage of the length of the target sequence, where *expandpct* ranges from 0 to 100, 0 implies no compression, and 100 implies maximum allowable compression. The default value is 100.

expandabs specifies the absolute warping expansion, where *expandabs* is an integer that ranges from 0 to 10,000. The default is the maximum allowable absolute expansion.

- compresspct* specifies the warping compression as a percentage of the length of the target sequence, where *compresspct* ranges from 0 to 100, 0 implies no compression, and 100 implies maximum allowable compression. The default value is 100.
- compressabs* specifies the absolute warping compression, where *compressabs* is an integer that ranges from 0 to 10,000. The default is the maximum allowable absolute compression.

Returned Values

The SIMILARITY function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
4	Warping limits relaxed
3	Expansion limits relaxed
2	Compression limits relaxed
1	Warping limits imposed
0	Success
< 0	Computational failure

- measure* returns the similarity measure.

Example

This example uses the TSMODEL procedure to compute the similarity of two time series arrays: x and y.

```
data test;
  input i x1 x2 x3 y1 y2 y3 r;
  datalines;
  1 3 2 4 2 3 2 1
  2 5 4 5 4 5 3 1
  3 3 3 4 6 4 5 1
  4 3 6 6 7 6 7 1
  5 3 5 5 3 5 7 1
  6 6 6 6 8 8 8 1
  7 3 8 5 9 9 8 1
  8 8 9 8 3 7 3 1
  9 6 7 6 8 4 9 1
  10 7 9 8 9 6 7 1
;
run;

data mycas.testsim;
  set test;
run;

proc tsmodel data=mycas.testsim
  outscalar=mycas.sim_scalar
  nthreads=1;
  require tsa;
  id i interval=day;
```

```

var x1 y1;
outscalars measure;
submit;
declare object TSA(tsa);
rc = TSA.SIMILARITY(x1, y1, 'absdev', 'NONE', , , , measure);
endsubmit;
run;

```

STATIONARITYTEST Method

`rc = TSA.STATIONARITYTEST (y, <dif>, <d>, <p>, <'type'>, pvalue);`

The STATIONARITYTEST function tests for stationarity of a univariate time series.

Required Arguments

You must specify the following argument:

y specifies the times series array to test.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

<i>dif</i>	specifies an array of positive integers or a positive integer that is used for differencing. The default value is 0.
<i>d</i>	specifies the order of unit root ($d = 1, \dots, 12$). If the <i>type</i> is SSM, then $d = 1$. The default value is 1.
<i>p</i>	specifies the autoregressive order, where p must be a nonnegative integer. The default value is 5.
<i>'type'</i>	specifies the type of test statistic used.

You can specify the following values within single quotation marks:

SSM	specifies the studentized test statistic for the single mean (intercept) case.
STR	specifies the studentized test statistic for the deterministic time trend case.
SZM	specifies the studentized test statistic for the zero mean (no intercept) case. This value is allowed only when <i>d</i> has a value of 1.

The default value of *type* is SZM.

Returned Values

The STATIONARITYTEST function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Time series is stationary with the default significance level of 0.05
1	Time series is not stationary with the default significance level of 0.05
< 0	Computational failure

pvalue returns the probability value associated with the test.

Example

This example uses the TSMODEL procedure to test the stationarity on the time series array Air:

```
proc tsmode1 data=mycas.air outscalar=mycas.outscalars;
  id date interval=month;
  var air;
  outscalars stationary1 stationary2;
  require tsa;
  submit;
  declare object TSA(tsa);
  stationary1=1; stationary2=1;
  rc = TSA.STATIONARITYTEST(air,,,,,pvalue);
  *test with the default significant level=0.05;
  if rc =1 then stationary1 = 0;
  *test with significant level = 0.1;
  if pvalue > 0.1 then stationary2 = 0;
  endsubmit;
run;
```

TRANSFORM Method

```
rc = TSA.TRANSFORM (y, <'type'>, <inverse>, <c>, x) ;
```

The TRANSFORM function transforms a time series to another form.

Required Arguments

You must specify the following arguments:

y specifies an input time series array.

Optional Arguments

You can also specify the following arguments, separated by commas. If you want to use a default value for any of these arguments, enter a space for it.

type specifies the type of transformation. You can specify the following values within single quotation marks:

LOG	specifies logarithmic transformation.
SQRT	specifies square root transformation.
LOGIT	specifies logit transformation.
BOXCOX	specifies Box-Cox transformation.
NONE	requests that no transformation be performed.

The default value is NONE.

inverse specifies whether to perform an inverse transformation. You can specify the following values:

0	does not perform an inverse transformation.
1	returns the inverse of the specified transformation method.

The default value is 0.

c specifies a parameter to be used in the transformation. Its use depends on the transformation method as follows:

- For log transformation, *c* is bias: $x = \log(y + c)$. The default value is 0.
- For square root transformation, *c* is bias: $x = \sqrt{y + c}$. The default value is 0.
- For logit transformation, *c* is scaling: $x = \log(c \times y / (1 - (c \times y)))$. The default value is 1.
- For the Box-Cox transformation, *c* is λ : $x = c^2 + (y^c - 1)/c$. If *c* is not specified, $x=y$.

Returned Values

The TRANSFORM function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

x returns the transformed series.

Example

This example uses the TSMODEL procedure to take the log transform of the time series array Air:

```
proc tsmodel data=mycas.air outarray=mycas.trans_array;
  id date interval=month;
  var air;
  outarrays t1;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.TRANSFORM(air, 'LOG', 0, 0, t1);
  endsubmit;
run;
```

UNBIASEDNESS Method

```
rc = TSA.UNBIASEDNESS (y, predict, < siglevel>, intercept, scale, fvalue, pvalue) ;
```

The UNBIASEDNESS function tests whether a univariate time series is unbiased.

Required Arguments

You must specify the following arguments, separated by a comma:

- y* specifies the input time series array.
- predict* specifies an input array of predicted time series.

Optional Arguments

You can also specify the following argument. If you want to use a default value for this argument, enter a space for it.

- siglevel* specifies the significance level.

Returned Values

The UNBIASEDNESS function returns the following values:

- rc* returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
1	Biased predictions
0	Unbiased predictions
–1	Degree of freedom error
–2	Singular system
–3	Extreme value

<i>intercept</i>	returns the constant parameter.
<i>scale</i>	returns the scale parameter.
<i>fvalue</i>	returns the test statistic for the <i>F</i> test.
<i>pvalue</i>	returns the <i>p</i> -value for the <i>F</i> test.

Example

This example uses the TSMODEL procedure to test whether the series Actual is unbiased:

```
proc hpf data=sashelp.air out=_null_ outfor=outfor;
    id date interval=month;
    forecast air;
run;

proc reg data=outfor;
    model actual=predict;
    test intercept=0, predict=1;
run;
quit;
data mycas.outfor;
    set outfor;
run;

proc tsmodel data=mycas.outfor outscalar=mycas.bias_scalar outarray=mycas.bias_array;
    id date interval=month;
    var ACTUAL PREDICT;
    outscalars intercept scale fvalue pvalue;
    require tsa;
    submit;
    declare object TSA(tsa);
    rc=TSA.UNBIASEDNESS(ACTUAL, PREDICT, 0.05, intercept, scale, fvalue, pvalue);
    endsubmit;
run;
```

WHITENOISE Method

```
rc = TSA.WHITENOISE (y, nlag, lags, df, wn, <wnprob>, <wnlprob>);
```

The WHITENOISE function tests for white noise in a time series array.

Required Arguments

You must specify the following arguments, separated by a comma:

<i>y</i>	specifies the times series array to compute.
<i>nlag</i>	specifies the number of the lag to use in the calculation.

Returned Values

The WHITENOISE function returns the following values:

rc returns one of the following scalar return codes:

<i>rc</i>	Termination Reason
0	Success
< 0	Computational failure

lags returns the number of the lag that was used in the calculation.

df returns the number of observations that were used to test white noise.

wn returns an array of Ljung-Box white noise tests, with *nlag*+1 entries.

Optional Returned Values

You can also specify the following arguments, separated by commas to request additional returned values. If you do not want the value to be returned, enter a space for it.

wnprob returns white noise probabilities.

wnlprob returns white noise log probabilities.

Example

This example uses the TSMODEL procedure to perform the white noise test of lag 3 of the time series Air:

```
proc tsmodel data=mycas.air outarray=mycas.outarray;
  id date interval=month;
  var air;
  outarrays lags df wn wnprob wnlprob;
  require tsa;
  submit;
  declare object TSA(tsa);
  rc=TSA.WHITENOISE(air, 3, lags, df, wn, wnprob, wnlprob);
  endsubmit;
run;
```

References

Hodrick, R. J., and Prescott, E. C. (1980). "Post-war U.S. Business Cycles: An Empirical Investigation." Discussion Paper 451, Carnegie Mellon University.

Chapter 19

Time Series Model Package

Contents

Overview: TSM Package	1112
TSM Package Summary	1112
Using the TSM Package	1115
Return Codes	1117
TSM Object	1117
TSM Synopsis	1119
TSM Methods	1119
CFC Object	1124
CFC Synopsis	1126
CFC Methods	1126
ESMSPEC Object	1132
ESMSPEC Synopsis	1133
ESMSPEC Methods	1134
ARIMASPEC Object	1137
ARIMASPEC Synopsis	1139
ARIMASPEC Methods	1139
EXMSPEC Object	1145
EXMSPEC Synopsis	1147
EXMSPEC Methods	1147
IDMSPEC Object	1149
IDMSPEC Synopsis	1150
IDMSPEC Methods	1150
UCMSPEC Object	1154
UCMSPEC Synopsis	1155
UCMSPEC Methods	1156
TSMPEST Object	1161
TSMPEST Synopsis	1162
TSMPEST Methods	1162
TSMSPEC Object	1164
TSMSPEC Synopsis	1165
TSMSPEC Methods	1165
TSMFOR Object	1165
TSMFOR Synopsis	1167
TSMFOR Methods	1168
TSMSTAT Object	1169

TSMSTAT Synopsis	1173
TSMSTAT Methods	1173
TSMINEST Object	1175
TSMINEST Synopsis	1175
TSMINEST Methods	1175
TSMINSPEC Object	1176
TSMINSPEC Synopsis	1176
TSMINSPEC Methods	1176
Examples: TSM Package	1177
Using CAS Sessions and CAS Engine Librefs	1177
Example 19.1: Fitting and Forecasting with ARIMA and ESM Models	1178
Example 19.2: Fitting a Transfer Function Model	1181
Example 19.3: Replaying a Previously Fitted Model	1183
Example 19.4: Performing Time Series Imputation Using an ARIMA Model	1185
Example 19.5: Combining Forecasts	1189
References	1191

Overview: TSM Package

The time series model (TSM) package contains a set of time series modeling objects that provide a flexible way to model and forecast time series. For more information about the statistical methodology that underlies this package, see relevant chapters in *SAS/ETS User's Guide* and *SAS Forecast Server Procedures: User's Guide*.

TSM Package Summary

Table 19.1 lists the objects that are contained in the TSM package. These objects are designed to provide access to various univariate time series model families.

Table 19.1 Objects in the Time Series Model Package

Object	Description
Central Objects	
TSM	Time series model object (Note 1)
CFC	Combine Forecasts object (Note 2)
Time Series Model Specification Objects (Note 3)	
ARIMASPEC	Autoregressive integrated moving average model specification object
ESMSPEC	Exponential smoothing model specification object

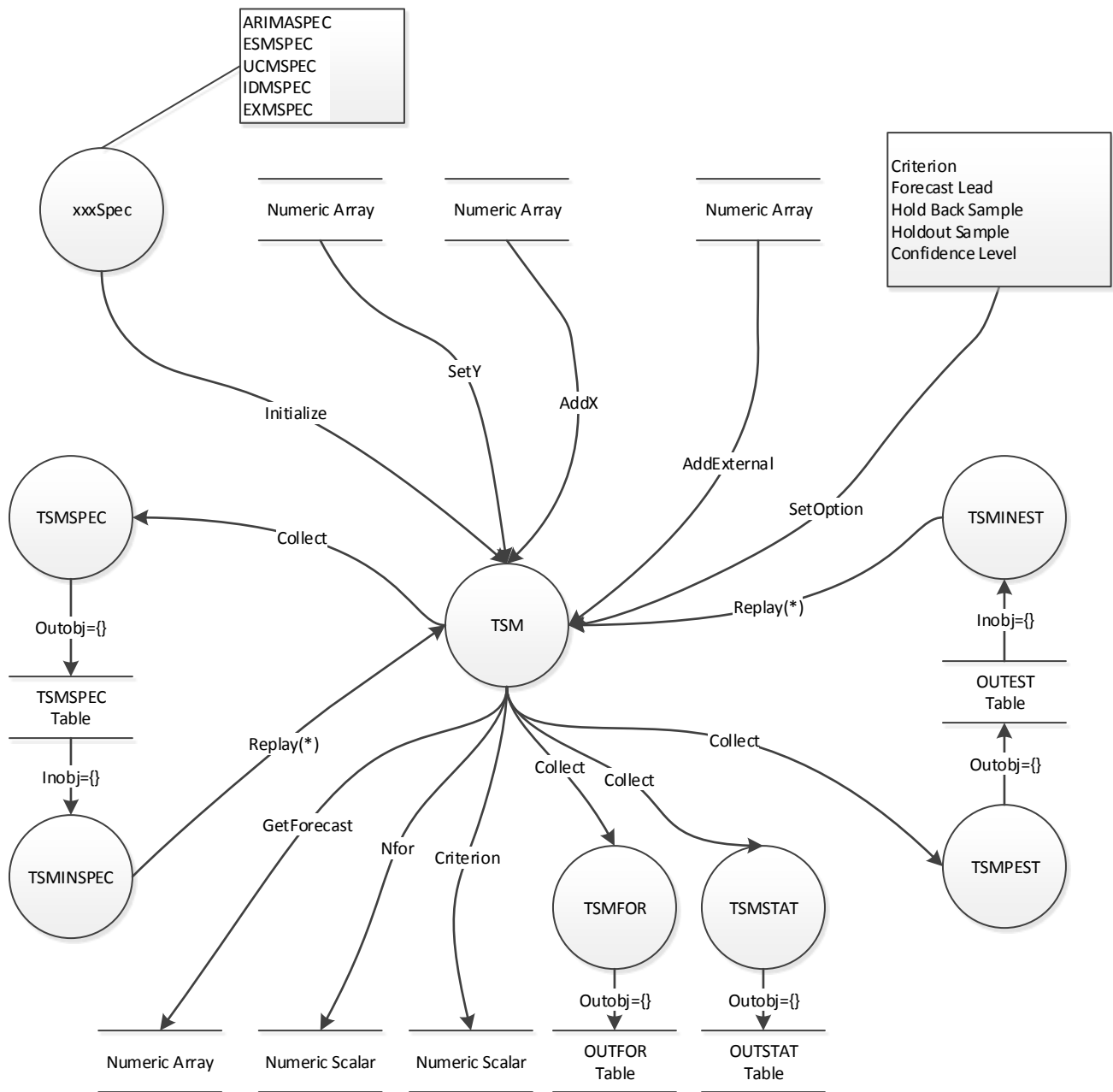
Table 19.1 *continued*

Object	Description
EXMSPEC	External model specification object
IDMSPEC	Intermittent demand model specification object
UCMSPEC	Unobserved component model (UCM) specification object
Time Series Model Collector Objects (Note 4)	
TSMFOR	Time series model forecast collector object
TSMPEST	Time series model parameter estimates collector object
TSMSPEC	Time series model specification collector object
TSMSTAT	Time series model fit statistics collector object
Time Series Model Repeater Objects (Note 5)	
TSMINEST	Time series model input parameter estimates repeater object
TSMINSPEC	Time series model input specification repeater object

NOTE:

1. The TSM object is the central hub that interacts with all other objects. It executes and encapsulates the computational services for all univariate time series models.
2. The Combine Forecasts (CFC) object allows you to perform forecast combinations.
3. The time series model specification objects enable you to specify the characteristics of a time series model to be executed by a TSM object.
4. The time series model collector objects operate on TSM objects to collect and save various results from the time series model execution (fit statistics, forecasts, parameter estimates, model specifications, and so on).
5. The time series model repeater objects act as conduits to replay collected model specifications and model parameter estimates as input to other TSM objects.

Figure 19.1 diagrams the relationships among the objects in the TSM package.

Figure 19.1 TSM Object Data Flow

Using the TSM Package

The following steps provide a general outline of how to use each type of object in the TSM package. Subsequent sections describe each step in greater detail.

- 1 Configure a model specification object.
- 2 Use the TSM object with the model specification object to generate a forecast.
- 3 Use TSM collector objects to extract results and parameter estimates from the TSM object.
- 4 Use TSM repeater objects to replay collected parameters and specifications to another TSM object.

Step1: Configure a Model Specification Object

Model specification objects define the time series model characteristics that you want to be applied to the TSM object, which then performs the model execution. Model specification objects use **Open** and **Close** methods to initialize and finalize model specifications. The basic execution pattern follows this sequence of operations:

- 1 **Declare:** Create the model specification object by using the object declaration statement. The object declaration assigns a default model specification to the specification objects.
- 2 **Open:** Initialize the specification object to a default state that is ready to accept configuration methods that shape the model and define its characteristics.
- 3 **Configure:** Use object-specific model specification methods to configure the model. For example, in an ESMSPEC, you might specify the smoothing method or model to be used, specify a functional transformation for the dependent variable to force the use of specific smoothing parameters, or specify bounds on the estimated parameters. For an ARIMASPEC, you might specify a set of autoregressive or moving average backshift operator polynomial factors, specify differencing operators, add simple and complex transfer functions, or specify a functional transformation for the dependent variable.
- 4 **Close:** Declare the model specification object to be complete and ready for use.
- 5 **Use:** Use the completed model specification object with a TSM object to directly perform the specified time series model fit and forecast computations. You can also use model specification objects with the automatic time series modeling (ATSM) package to include custom models into its automatic time series forecasting process. For more information about this use, see Chapter 3, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

Step2: Use a TSM Object with a Model Specification Object

The TSM object executes the time series model. The TSM object is configured with a time series model from one of the model specification objects. Then the TSM object applies that configuration to the time series data to produce a forecast. The basic execution pattern follows this sequence of operations:

- 1 Declare:** Create the TSM object by using the object declaration statement.
- 2 Initialize:** Add a model specification object to the TSM object.
- 3 Specify variables:** Specify the dependent series (Y) and any independent series (X) variables.
- 4 Specify options:** Specify other options and properties as appropriate.
- 5 Run:** Execute the model in the TSM object to produce its forecast.
- 6 Extract:** Extract the results by using collector objects.

Various properties (attributes) of the executed TSM object can be queried directly and saved into declared variables, or the results can be collected by the TSM collector objects for presentation and storage.

Step3: Use the TSM Collector Objects

Collector objects enable you to create a snapshot of results from TSM objects and store those results in CAS tables. Each collector object defines a table schema that is determined by the collector object's design. The TSM collector objects follow a common pattern. The basic execution follows this sequence of operations:

- 1 Declare:** Create the collector object using the object declaration statement.
- 2 Collect:** Use the Collect method with a TSM object passed in as an argument to collect results from the TSM object. For example, the TSMPEST object collects parameter estimates from a TSM object, and the TSMSPEC object collects from a model specification's XML. Rows that are collected are automatically appended to the collector's associated CAS table at the end of each BY group, and the collector object's saved row set is automatically reset. Rows that are added to the CAS table are qualified by the values of the corresponding values of the BY variable. This enables repeater objects to locate the rows that are relevant to each BY group and correctly replay that information into a TSM object. The **nrows** attribute returns the current row count in the collector. A missing value is returned if nothing has been collected. The data can now be used for reports or used by a repeater object on another model.

Step 4: Use the TSM Repeater Objects

Repeater objects read rows from a CAS table and convert their contents back into useful information that can be used by other TSM objects. This is the inverse function of the collector objects. Each repeater object defines a CAS table schema that is determined by its counterpart collector object's design. Repeater objects must be associated with an existing CAS table that has the table schema that is required by the repeater object. The basic execution follows this sequence of events:

- 1 Declare:** Create the repeater object by using the object declaration statement.

- 2 Replay:** Use the TSM.Replay method on a TSM object with the repeater object passed in as an argument to execute a new time series model. The replayed TSM object generates the forecast data that was produced by the new time series model. Then the data can be collected by using a collector object as described in Step 3.

Return Codes

Table 19.2 shows the return code (designated by *rc* in method statements) status values that are used in this package. These status code values are returned after a method that is associated with an object is called; they can help determine whether the method executed successfully.

Table 19.2 Return Codes

Status	Meaning
< 0	An unrecoverable error occurred. No result was produced.
= 0	Unconditional success. The requested action was completed, and a normal result was produced.
> 0	Conditional success or warning. A result was produced subject to conditions.

TSM Object

The TSM object generates forecasts of univariate time series.

Table 19.3 lists the time series model families that are supported.

Table 19.3 Model Families for the TSM Object

Family	Object	Description
ARIMA	ARIMASPEC	ARIMAX models
ESM	ESMSPEC	Exponential smoothing models
EXM	EXMSPEC	External model (external forecast)
IDM	IDMSPEC	Intermittent demand models (Croston's/average demand (ADEM))
UCM	UCMSPEC	Unobserved component models

Table 19.4 summarizes the methods that are associated with the TSM object.

Table 19.4 Methods of the TSM Object

Method	Description
AddX	Add an independent time series array (<i>XSeries</i>) for the TSM object
AddExternal	Add external forecast component series for the TSM object
criterion	Return the final fit statistic over a specified forecast region
GetForecast	Get the forecast series
Initialize	Initialize the TSM object
nfor	Return the forecast series length
Replay	Replay the restored model and parameter estimates
Run	Run the TSM object
SetOption	Specify the named option for the TSM object
SetY	Specify the dependent time series array (<i>YSeries</i>) for the TSM object

The basic execution pattern for using a TSM object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new TSM object.
- 2 Initialize:** The TSM.Initialize method takes a model specification object as its argument and initializes the TSM object for that specified time series model. If no model specification object is provided, the TSM object is initialized as an exponential smoothing method (ESM) that uses the best suited exponential smoothing model. This is equivalent to initializing the TSM object with an ESMSPEC object that has default option values.
- 3 SetY:** The TSM.SetY method defines the dependent time series for the TSM object.
- 4 AddX:** The TSM.AddX method defines any independent time series for the TSM object. Each call defines one predictor series. Repeat as needed for each predictor series.
- 5 SetOption:** The TSM.SetOption method specifies any options that affect the running of the model. Each call defines an option. Repeat as needed to specify all options that are required.
- 6 Run:** The TSM.Run method uses its currently configured X and Y time series data to execute the time series model that is defined by the TSM object's model specification. At completion, the model has estimated the parameters and produced a final forecast based on these parameters.

TSM Synopsis

DECLARE OBJECT *obj* (TSM) ;

Method syntax, in order of typical usage:

```
rc=obj.Initialize (< ModelSpec >) ;
rc=obj.SetY (YSeries) ;
rc=obj.AddX (XSeries < ,Required,NoDiff,ModelSymbol >) ;
rc=obj.AddExternal (Series < ,Role >) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
rc=obj.Replay (TSMINSPECObj < , TSMINESTObj >) ;
rc=obj.Run () ;
rc=obj.GetForecast (Which, Result) ;
nfor=obj.nfor () ;
criterion=obj.criterion (Region) ;
```

TSM Methods

TSM.AddExternal Method

```
rc=obj.AddExternal (Series < ,Role >) ;
```

Adds a time series array, *Series*, for use in external model computations when the TSM object is initialized from an external model specification (EXMSPEC) object. Calling this method when the TSM object is not configured with an EXMSPEC results in an error return and no further action. For more information about external model support, see the section “[EXMSPEC Object](#)” on page 1145.

No default role mapping is implied by the name of the *Series* variable that you specify in the method. Each call to the AddExternal method adds the specified series to the TSM object according to its role in the external model. That role association happens in one of the following ways:

1. You specify the *Role* argument in the method call. This takes precedence over any role mapping that is defined in the EXMSPEC that was used in the TSM.Initialize method. If you specify an invalid *Role* value, an error is generated.
2. You specify a *Series* variable that matches a role mapping in the EXMSPEC object that was used in the TSM.Initialize method. If you specify a *Series* variable that fails to match a role mapping in the EXMSPEC, an error is generated.

This method can be called as many times as needed to specify all of the external series that are required to run the external model. In all cases, if the series that you specify to add fails to resolve to a role in the EXMSPEC object, an error is generated without the series being included in the TSM object. Such failures do not cause subsequent TSM.AddExternal method calls to fail.

Input Arguments

You must specify the following input arguments:

<i>Series</i>	specifies a numeric array that contains an external forecast series for the TSM object.
<i>Role</i>	is a case-sensitive character string that specifies the role of the external forecast series in the external model. You can specify one of the following values:
ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

TSM.AddX Method

```
rc=obj.AddX (XSeries < ,Required,NoDiff,ModelSymbol> ) ;
```

Adds an independent time series array (*XSeries*) for the TSM object. Each call of the TSM.AddX method adds the specified *XSeries* array variable to the TSM object. This method can be called as many times as needed to specify all the independent variables. By default, the name of the *XSeries* variable must match the name of an input symbol in the model specification that is used to configure the TSM object. You can specify a symbol name, *ModelSymbol*, to associate an *XSeries* array to an input symbol in the model specification. Only ARIMA models support *XSeries* predictors.

Input Arguments

You must specify the following input argument:

<i>XSeries</i>	specifies a numeric array that contains an independent series for the TSM object.
----------------	---

You can also specify the following input arguments:

<i>Required</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the <i>XSeries</i> variable is required to be in the model. This might cause the model estimation to fail if the <i>XSeries</i> array variable is deemed inadmissible for inclusion into the underlying ARIMA model. The default value is 0.
<i>NoDiff</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the <i>XSeries</i> variable does not automatically follow the <i>XSeries</i> variable differencing. The default value is 0.
<i>ModelSymbol</i>	takes a character variable that specifies the name of an input symbol in the model specification to be associated with the <i>XSeries</i> variable when the model is run. By default, the TSM.AddX method adds <i>XSeries</i> variables into an ARIMA model specification if they are not already referenced as an input symbol. Specification of <i>ModelSymbol</i> defines a way to include the <i>XSeries</i> variable if the ARIMASPEC object includes a symbol that matches the <i>ModelSymbol</i> argument.

TSM.criterion Method

*criterion=*obj.criterion (*Region*) ;

Returns the fit statistic value over the specified forecast *Region* for the TSM object. The criterion is set via the 'CRITERION' argument in the TSM.SetOption method. A missing value indicates that the TSM object instance has not produced a successful forecast.

Input Arguments

You must specify the following input argument:

<i>Region</i>	is a case-insensitive character string that specifies the forecast region over which the fit statistic is computed. You can specify one of the following values:
BACK	returns the fit statistic over the time region that is subsequent to the FIT region and that did not contribute any data for estimating model parameters (that is, the model forecast region).
FIT	returns the fit statistic over the time region that supplied observations for estimating model parameters (that is, the model fit region).

TSM.GetForecast Method

*rc=*obj.GetForecast (*Which*, *Result*) ;

Gets the specified forecast series (*Which*) from the TSM object and stores it in the specified numeric array (*Result*).

Input Arguments

You must specify the following input argument:

<i>Which</i>	is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:
ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

<i>Result</i>	specifies a numeric array to receive the forecast series.
---------------	---

TSM.Initialize Method

```
rc=obj.Initialize (< ModelSpec>) ;
```

Initializes a TSM object to use the specified *ModelSpec*. This method must be called before the time series arrays (*XSeries* and *YSeries*) and other attributes for the TSM object are specified. If no *ModelSpec* object is specified, the TSM object is initialized to use the default ESM specification. This is equivalent to initializing the TSM object with an ESM SPEC object that has default option values.

Input Arguments

You can specify the following input argument:

ModelSpec specifies an optional name for a TSM model specification object that is used to configure the TSM object.

TSM.nfor Method

```
nfor=obj.nfor () ;
```

Returns the length (observation count) of the forecast series for the TSM object. A missing value indicates that the TSM object has not produced a successful forecast.

Arguments

There are no arguments associated with this method.

TSM.Replay Method

```
rc=obj.Replay (TSMINSPECObj <,TSMINESTObj>) ;
```

Uses a previously saved model specification from a *TSMINSPECObj* as input to another TSM object. Optionally, restored parameter estimates from *TSMINESTObj* are applied to the model specification.

Input Arguments

You must specify the following input argument:

TSMINSPECObj specifies the TSMINSPEC object to supply the model specification.

You can also specify the following input argument:

TSMINESTObj specifies the TSMINEST object to supply the model's parameter estimates.

TSM.Run Method

```
rc=obj.Run () ;
```

Runs the TSM object to estimate and forecast the time series model by using the specified dependent (*YSeries*) and independent (*XSeries*) series. Upon successful completion, various results can be extracted from the TSM object.

Arguments

There are no arguments associated with this method.

TSM.SetOption Method

```
rc=obj.SetOption ('Name', Value <,'Name', Value,... >);
```

Specifies the named options for the TSM object. When you invoke this method, the previous forecast produced by the [TSM.Run](#) method is discarded. You must then rerun the [TSM.Run](#) method in order to produce a new forecast using the updated configuration options.

Input Arguments

You must specify at least one of the following 'Names' and its associated *Value*:

'ALPHA'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the significance level for forecast confidence bands. The default value is 0.05.
'BACK'	takes a nonnegative integer <i>Value</i> that specifies the back region for model performance. If 'BACK'= <i>n</i> and the number of observations is <i>T</i> , then the first $T - n$ observations are used to diagnose a series. The default is value 0.
'CRITERION'	takes a string <i>Value</i> that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. For a list of valid values, see the CRITERION= option in the HPFDIAGNOSE procedure in <i>SAS Forecast Server Procedures: User's Guide</i> . The default is RMSE.
'HOLDOUT'	takes a nonnegative integer <i>Value</i> that specifies the holdout region for model selection. The holdout sample is a subset of dependent series (which you specify by using the TSM.SetY method) that ends at the last nonmissing observation. This option is relevant only when a TSM instance has been initialized via the Initialize method by using a model specification that requires model selection. Currently, this applies only to a model specification that is created by an ESMSPEC object and whose 'METHOD' option is set to either 'BEST', 'BESTN', or 'BESTS' via the ESMSPEC.SetOption method. The default value is 0.
'HORIZON'	takes a numeric <i>Value</i> that specifies the forecast horizon reference time. When set to a missing value, the forecast horizon reference time is automatically set as the first time period that follows the last nonmissing observation of the dependent series. The default value is a missing value.
'LEAD'	takes a nonnegative integer <i>Value</i> that specifies the forecast lead. The default value is the value specified in the PROC TSMODEL statement (LEAD= option). If LEAD= is not specified in the PROC TSMODEL statement, the default value of LEAD=0.

TSM.SetY Method

```
rc=obj.SetY (YSeries);
```

Specifies the dependent time series array (*YSeries*) for the TSM object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the TSM object.

CFC Object

The CFC object creates optimized combinations of time series model forecasts. The performance of combined model forecasts tends to improve over that of individual models. You can specify the model forecasts to be combined either as arbitrary time series arrays or as instances of the TSM object that have been previously run. Table 19.5 summarizes the methods that are associated with the CFC object.

Table 19.5 Methods of the CFC Object

Method	Description
AddModel	Add a forecast series from an instance of the TSM object
AddPredict	Add a time series array as a forecast series
criterion	Return the final fit statistic over a specified forecast region
GetForecast	Get the forecast series
Initialize	Initialize the CFC object
nfor	Return the forecast series length
Run	Run the CFC object to generate a combined forecast
SetOption	Specify the named option for the CFC object
SetY	Specify the dependent time series array (<i>YSeries</i>) for the CFC object

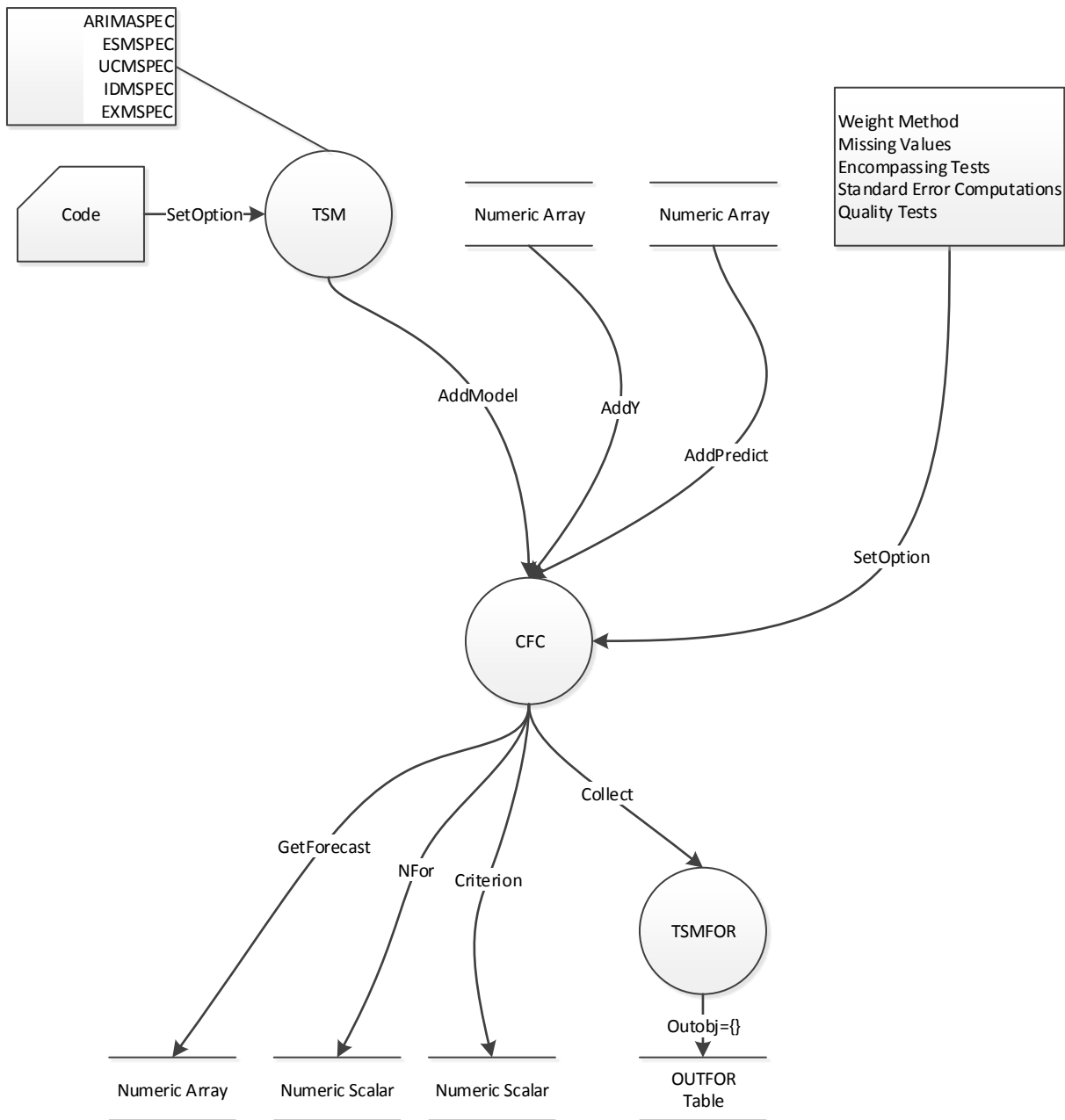
The basic execution pattern for using a CFC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new instance of the CFC object.
- 2 Initialize:** The [CFC.Initialize](#) method resets the object in preparation for a new run.
- 3 SetY:** The [CFC.SetY](#) method defines the dependent time series for the CFC object.
- 4 AddModel:** The [CFC.AddModel](#) method adds the forecast series that was produced by an instance of the TSM object. Each call adds one model forecast series to the combination. Repeat as needed for each model forecast series to be combined. Also, if you invoke this method prior to invoking the [CFC.SetY](#) method, then this method will use the dependent series of the TSM object as the dependent series of the CFC object.
- 5 AddPredict:** The [CFC.AddPredict](#) method adds a forecast series to the combination that is specified as a numeric array. Each call adds one forecast series to the combination. Repeat as needed for each forecast series to be combined.

- 6 SetOption:** The `CFC.SetOption` method specifies any options that affect the generation of the combined model forecast. Each call defines an option. Repeat as needed to specify all options that are required.
- 7 Run:** The `CFC.Run` method uses its currently configured dependent and forecast series data to generate a combined forecast.

Figure 19.2 illustrates the data flow through the CFC object and its relationship with other components of the TSM package.

Figure 19.2 CFC Object Data Flow



CFC Synopsis

DECLARE OBJECT *obj* (CFC) ;

Method syntax, in order of typical usage:

```
rc=obj.Initialize () ;
rc=obj.SetY (YSeries) ;
rc=obj.AddModel (ModelObj < , ModelName >) ;
rc=obj.AddPredict (Predict, PredictSE < ,ModelName, Nparms > ) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value,... >) ;
rc=obj.Run () ;
rc=obj.GetForecast (Which, Result) ;
nfor=obj.nfor () ;
criterion=obj.criterion (Region) ;
```

CFC Methods

CFC.AddModel Method

```
rc=obj.AddModel (ModelObj < , ModelName >) ;
```

Adds the forecast model series that were produced by an instance of the TSM object to the forecast combination process. This includes both the predicted and standard error series that were produced by the TSM object during the forecasting process. By default, the CFC object uses the name of the TSM object instance to identify the forecast component associated with its forecast series. However, you can optionally specify the forecast component via the *ModelName* argument.

Input Arguments

You must specify the following input argument:

ModelObj specifies the TSM object to use as a source of forecast model series.

You can also specify the following input argument:

ModelName is a character string that specifies the name of the forecast component that is used to identify the model forecast series in the combination. The default is the name of the TSM object instance that is used as the source of forecast model series.

CFC.AddPredict Method

```
rc=obj.AddPredict (Predict, PredictSE < , ModelName, Nparms > ) ;
```

Adds a forecast series to be included in the forecast combination for the CFC object. This includes both the *Predict* predicted series and the *PredictSE* standard errors series as numeric arrays. You can optionally specify the name of the forecast component in the combination via the *ModelName* argument. Each call to the AddPredict method adds the specified *Predict* and *PredictSE* series to the CFC object to be included in the combination process. The arrays that are used to specify the *Predict* and *PredictSE* series do not need to be unique, but the forecast component names that are specified in the *ModelName* argument must be unique if you specify them. You can call the AddPredict method repeatedly to specify multiple forecast series.

Input Arguments

You must specify the following input arguments:

- Predict* specifies a numeric array that contains the predicted series.
- PredictSE* specifies a numeric array that contains the standard errors of the predicted series.

You can also specify the following input arguments:

- ModelName* is a character string that specifies the name of the forecast component that is used to identify the model forecast series in the combination. The default is a unique name that is autogenerated by the CFC object.
- Nparms* takes a numeric value that specifies the number of parameters used by the model that generated the forecast series. Valid values are integers greater than or equal to 0. The default value is 0.

CFC.criterion Method

criterion=obj.criterion (Region) ;

Returns the fit statistic value over the specified forecast *Region* for the CFC object. The criterion is set via the 'CRITERION' argument in the CFC.SetOption method. A missing value indicates that the CFC object instance has not produced a successful forecast.

Input Arguments

You must specify the following input argument:

- Region* is a case-insensitive character string that specifies the forecast region over which the fit statistic is computed. You can specify one of the following values:
- | | |
|-------------|--|
| BACK | returns the fit statistic over the time region that is subsequent to the FIT region and that did not contribute any data to the process of estimating combination weights for the specified model forecast series. |
| FIT | returns the fit statistic over the time region that supplied observations for the process of estimating combination weights for the specified model forecast series. |

CFC.GetForecast Method

rc=obj.GetForecast (Which, Result) ;

Gets the specified forecast series (*Which*) from the CFC object and stores it in the specified numeric array (*Result*).

Input Arguments

You must specify the following input argument:

- Which* is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:

ERROR	returns prediction errors.
LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
PREDICT	returns a prediction series.
UPPER	returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

Result specifies a numeric array to receive the forecast series.

CFC.Initialize Method

```
rc=obj.Initialize () ;
```

Initializes a CFC object by discarding a previous forecast and resetting the object's configuration to the default values. This method must be called before any other attributes for the CFC object are specified.

Arguments

There are no arguments associated with this method.

CFC.nfor Method

```
nfor=obj.nfor () ;
```

Returns the length (observation count) of the forecast series for the CFC object. A missing value indicates that the CFC object has not produced a successful forecast.

Arguments

There are no arguments associated with this method.

CFC.Run Method

```
rc=obj.Run () ;
```

Runs the CFC object to perform the forecast combination process by using the dependent (*YSeries*) and candidate forecast series that have been specified for it. Upon successful completion, various results can be extracted from the CFC object.

Arguments

There are no arguments associated with this method.

CFC.SetOption Method

```
rc=obj.SetOption ('Name', Value <, 'Name', Value,... >) ;
```

Specifies the named options for the CFC object. When you invoke this method, the previous forecast combination produced by the [CFC.Run](#) method is discarded. You must then rerun the [CFC.Run](#) method in order to produce a new forecast combination using the updated configuration options.

Input Arguments

You must specify at least one of the following 'Names' and its associated *Value*:

'BACK'	takes a nonnegative integer <i>Value</i> that specifies the back region for model performance. If 'BACK'=n and the number of observations is <i>T</i> , then the first $T - n$ observations are used to diagnose a series. The default value is 0.						
'CRITERION'	takes a string <i>Value</i> that specifies the fit statistic to be computed for the final forecast. The default is RMSE.						
'ENCALPHA'	takes a numeric <i>Value</i> between 0 and 1, inclusive, that specifies the significance level of the encompassing test. The default value is 0.05.						
'ENCTEST'	takes a string <i>Value</i> that specifies the encompassing test type. The encompassing test attempts to eliminate from consideration any forecasts that fail to add significant information to the final forecast. For a detailed description of the encompassing test, see the chapter “Forecast Combination Computational Details” in <i>SAS Forecast Server Procedures: User's Guide</i> . You can specify the following <i>Values</i> : <table data-bbox="406 1050 1443 1543"> <tr> <td>HLN</td><td>uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.</td></tr> <tr> <td>NONE</td><td>performs no encompassing tests.</td></tr> <tr> <td>OLS</td><td>uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.</td></tr> </table> The default is NONE.	HLN	uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.	NONE	performs no encompassing tests.	OLS	uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.
HLN	uses the Harvey-Leybourne-Newbold (HLN) test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.						
NONE	performs no encompassing tests.						
OLS	uses an OLS-based regression test to estimate pairwise encompassing between candidate forecasts. Candidates are ranked from best to worst using the 'CRITERION' option. Iterating from best to worst, inferior candidates are tested together with the best of the untested candidates for retention in the combined set. The significance level for the test is given by the 'ENCALPHA' argument.						
'HMISSPCT'	takes a numeric <i>Value</i> between 0 and 100 that specifies a threshold for the percentage of missing forecast values in the combination horizon. This threshold is used to exclude a candidate forecast from consideration in the final combination. By default, no horizon missing percentage test is performed on candidate forecasts. The forecast horizon is the region of time in which multistep forecasts are generated.						
'LEAD'	takes a nonnegative integer <i>Value</i> that specifies the forecast lead. The default is the value that is specified in the PROC TSMODEL statement (LEAD= option). If the LEAD=						

option is not specified in the PROC TSMODEL statement, the default value of LEAD=0 is used.

'MISSMODE' takes a string *Value* that specifies a method for treating missing values in the forecast combination. In a particular time slice across the combination ensemble, one or more combination contributors can have a missing value. This setting determines the treatment of those contributors in the final combination for such time indices. You can specify the following *Values*:

MISSING generates a missing combined forecast at each time index with one or more missing contributors.

RESCALE rescales the combination weights for the nonmissing contributors at each time index to sum to 1. You cannot specify RESCALE when the value of the 'WEIGHT' argument is OLS or NRLS.

The default value depends on the combination weight method that is specified by the 'WEIGHT' argument. RESCALE is the default for simple average, user-specified weights, ranked user weights, ranked weights, and RMSE weights. MISSING is the default for all other methods.

'MISSPCT' takes a numeric *Value* between 0 and 100 that specifies a threshold for the percentage of missing forecast values in the combination estimation region. This threshold is used to exclude a candidate forecast from consideration in the final combination. By default, no missing percentage test is performed on candidate forecasts.

'RANKING' takes a string *Value* that specifies the fit statistic to be used for ranking forecast candidates. This option is used in conjunction with the 'ENCTEST' and 'WEIGHT' (when *Value* is set to RANKWGT) arguments. For a list of valid values, see the CRITERION= option in the HPFDIAGNOSE procedure in *SAS Forecast Server Procedures: User's Guide*. The default is RMSE.

'SEMODE' takes a string *Value* that specifies the method for computing the prediction error variance series. This series is used to compute the prediction standard error, which in turn is used to compute confidence bands on the combined forecast. You can specify the following *Values*:

DIAG computes the prediction error variance by assuming that the forecast errors at time t are uncorrelated so that the simple diagonal form of Σ_t is used.

ESTCORR computes the prediction error variance by using estimates of $\rho_{i,j,t}$, the sample cross-correlation between $e_{i,t}$ and $e_{j,t}$ over the time span $t = 1, \dots, T$, where t denotes the last time index of the actual series y_t . This *Value* implies that the error series $e_{i,t}$ and $e_{j,t}$ are assumed to be jointly stationary.

The default is DIAG.

'USERWEIGHTS' takes a numeric array *Value* that specifies the combination weights to be used when the value of the 'WEIGHT' argument is RANKWGT or USERDEF. If the specified weights do not sum to 1, then the weights are automatically normalized to sum to 1 and a warning message is issued. For more information, see the 'WEIGHT' argument.

'WEIGHT'

takes a string *Value* that specifies the method for determining the combination weights that are used in the weighted average of the candidate model forecasts. You can specify the following *Values*:

AICC	computes the combination weights on the basis of corrected AIC weights. By default, all AICC scored candidate forecasts are combined.
AVERAGE	computes the simple average of the forecasts that are selected for combination.
ERLS	computes the combination weights on the basis of a constrained least squares problem to minimize the ℓ_2 norm of the combined forecast residuals subject to the constraint that the weights sum to 1.
LAD	computes the weights on the basis of a least absolute deviations (LAD) measure of fit for the combined forecast. A linear program is formulated in which an objective function to be minimized is expressed in terms of the absolute values of a loss series subject to constraints that the weights sum to 1 and be nonnegative.
NERLS	computes the combination weights on the basis of a constrained least squares problem to minimize the ℓ_2 norm of the combined forecast residuals subject to the constraints that the weights sum to 1 and be nonnegative.
NRLS	is equivalent to NERLS, except that the resulting combination weights are not constrained to summing to 1.
OLS	computes the combination weights that result from the ordinary least squares problem to minimize the ℓ_2 norm of the combined forecast residuals.
RANKWGT	assigns weights by using the rank of the candidate forecasts at the time the combination is performed. You can optionally specify the weights in the 'USERWEIGHTS' argument, where the number of specified values must match the number of model forecasts that are specified in the current instance of the CFC object. The weights are assigned by ranking the candidate forecasts from best to worst on the basis of the fit statistic specified in the 'RANKING' argument. The best candidate uses the first weight, W1, and so on. If you do not specify the weights in the 'USERWEIGHTS' argument, then the weight of each candidate forecast is set to the inverse value of its rank. The set of weights that is used is normalized to account for candidate forecasts that are omitted from the final combination.
RMSEWGT	computes the combination weights on the basis of the RMSE statistic of fit for the forecast contributors. The weights are normalized to sum to 1.
USERDEF	assigns weights by using the list of user-specified values. You must specify the weights in the 'USERWEIGHTS' argument, where the number of specified values must match the number of model forecasts that are specified in the current instance of the CFC object. The weights correspond to the order of specification of the model forecasts. The

set of weights that is used is normalized to account for candidates that are omitted from the final combination.

The default is AVERAGE.

CFC.SetY Method

```
rc=obj.SetY (YSeries) ;
```

Specifies the dependent time series array (*YSeries*) for the CFC object.

Input Arguments

You must specify the following input argument:

YSeries specifies a numeric array that contains the dependent series for the CFC object.

ESMSPEC Object

The ESMSPEC object defines an exponential smoothing model specification for use with the TSM object. The basic execution pattern for defining an ESMSPEC follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new ESMSPEC object. By default, it is an ESMSPEC with METHOD=BEST.
- 2 Open:** The ESMSPEC.Open method initializes the default ESMSPEC object for a new configuration.
- 3 Configure:** The various ESMSPEC.Set methods configure the ESMSPEC object.
- 4 Close:** The ESMSPEC.Close method finalizes the ESMSPEC object.
- 5 Apply:** Add the ESMSPEC to a TSM object using the TSM.Initialize method.

Table 19.6 summarizes the methods that are associated with the ESMSPEC object.

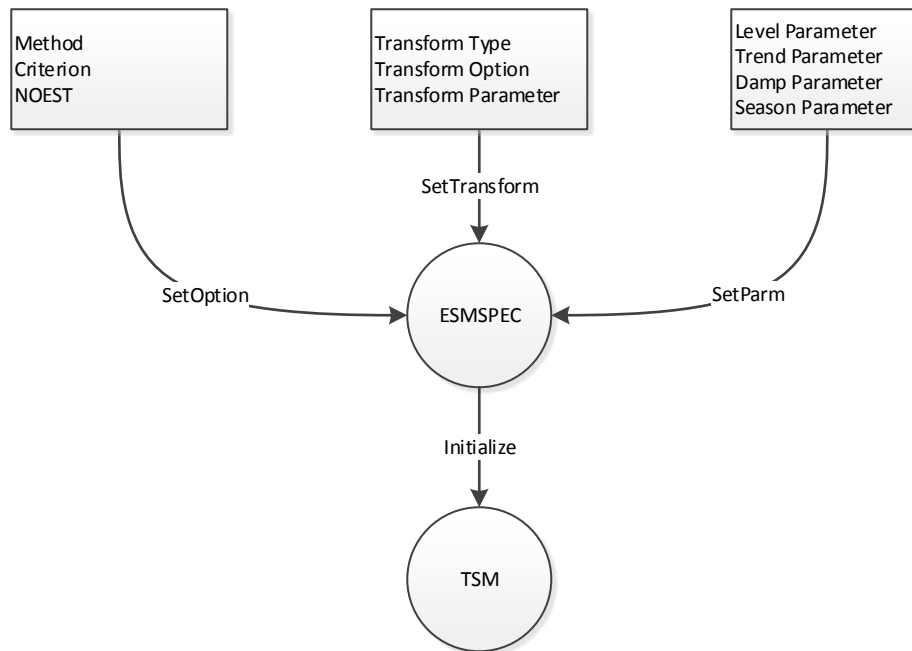
Table 19.6 Methods of the ESMSPEC Object

Method	Description
Close	Close ESM model specification
Open	Open ESM model specification
SetOption	Set option for ESM model
SetParm	Set parameters for ESM model
SetTransform	Set transform for ESM model

You can also store the XML representation of the ESMSPEC object in a CAS table. For more information, see the TSMSPEC object.

Figure 19.3 illustrates the data flow through the ESMSPEC object.

Figure 19.3 ESMSPEC Object Data Flow



ESMSPEC Synopsis

DECLARE OBJECT *obj* (ESMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.SetTransform ('Type' <,'Option',Parm>) ;
rc=obj.SetParm ('CompName', Parm, <LRest,URest>) ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.Close () ;
  
```

ESMSPEC Methods

ESMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the ESMSPEC object to prepare the ESM model to be used in a TSM object or to be imported to a TSMSPEC object for printing or for storage in a model repository catalog.

Arguments

There are no arguments associated with this method.

ESMSPEC.Open Method

```
rc=obj.Open () ;
```

Opens the ESMSPEC object.

Arguments

There are no arguments associated with this method.

ESMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
```

Specifies ESM options.

Input Arguments

You must specify at least one of the following 'Names' and its associated Value:

'CRITERION'	takes a string <i>Value</i> that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. For a list of valid values, see the CRITERION= option in the HPFDIAGNOSE procedure in <i>SAS Forecast Server Procedures: User's Guide</i> . The default is RMSE.
'METHOD'	takes a string <i>Value</i> that specifies the ESM method. You can specify the following values:
ADDWINTERS	requests the Winters additive method.
BEST	requests the best candidate smoothing model among the SIMPLE, LINEAR, DAMPTREND, SEASONAL, ADDWINTERS, or WINTERS methods.
BESTN	requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, or DAMPTREND methods.
BESTS	requests the best candidate seasonal smoothing model among the SEASONAL, ADDWINTERS, or WINTERS methods.
DAMPTREND	requests damped trend exponential smoothing.
DOUBLE	requests second order exponential smoothing.
LINEAR	requests linear (Holt) exponential smoothing.

MULTSEASONAL	requests multiplicative seasonal exponential smoothing.
SEASONAL	requests additive seasonal exponential smoothing.
SIMPLE	requests simple (single) exponential smoothing.
WINTERS	requests Winters multiplicative method.

The default is BEST.

'NOEST' takes a Boolean *Value* that indicates whether the smoothing model parameters are fixed values. By default, the smoothing model parameters are optimized. 'NOEST' requires all of the exponential smoothing model parameters to be explicitly specified via calls to the ESMSPEC.SetParm method. This argument is ignored if any of the model parameters is not specified.

ESMSPEC.SetParm Method

```
rc=obj.SetParm ('CompName', Parm, < LRest,URest> ) ;
```

Specifies parameter values and restrictions for the specified ESM model component. Optional bounds, *LRest* and *URest*, can be specified to restrict the weight value for the specified ESM component during parameter optimization. Parameter values and restrictions must be in the range from -1 to 2.

Input Arguments

You must specify the following input arguments:

'CompName' specifies a character variable that indicates the ESM component context. You can specify the following values:

DAMP	specifies that <i>Parm</i> is the initial value of a damping weight parameter.
LEVEL	specifies that <i>Parm</i> is the initial value of a level weight parameter.
SEASON	specifies that <i>Parm</i> is the initial value of a season weight parameter.
TREND	specifies that <i>Parm</i> is the initial value of a trend weight parameter.

By default, if the initial value of a parameter is not specified, then the optimizer uses a grid search to find an appropriate initial value.

Parm takes a numeric value that specifies the smoothing component weight.

You can also specify the following input arguments:

LRest is a numeric variable that specifies a lower bound on the smoothing weight.

URest is a numeric variable that specifies an upper bound on the smoothing weight.

ESMSPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' < ','Option',Parm>) ;
```

Specifies the functional transform, '*Type*', to be used by the ESM model. Optional arguments *Option* and *Parm* offer greater control over the transform.

Input Arguments

You must specify the following input argument:

<i>'Type'</i>	takes one of the following string values:
AUTO	automatically chooses between NONE and LOG based on model selection criteria.
BOXCOX (<i>value</i>)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).
LOG	requests logarithmic transformation.
LOGIT	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.

The default is NONE.

You can also specify the following input arguments:

<i>'Option'</i>	takes one of the following string values that specifies prediction semantics for the inverse transform:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.

The default is MEAN.

<i>Parm</i>	takes a numeric value between –5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transforms.
-------------	---

ARIMASPEC Object

The ARIMASPEC object generates autoregressive integrated moving average (ARIMA) model specifications for use with a TSM object.

Table 19.7 summarizes the methods that are associated with the TSM.ARIMASPEC method.

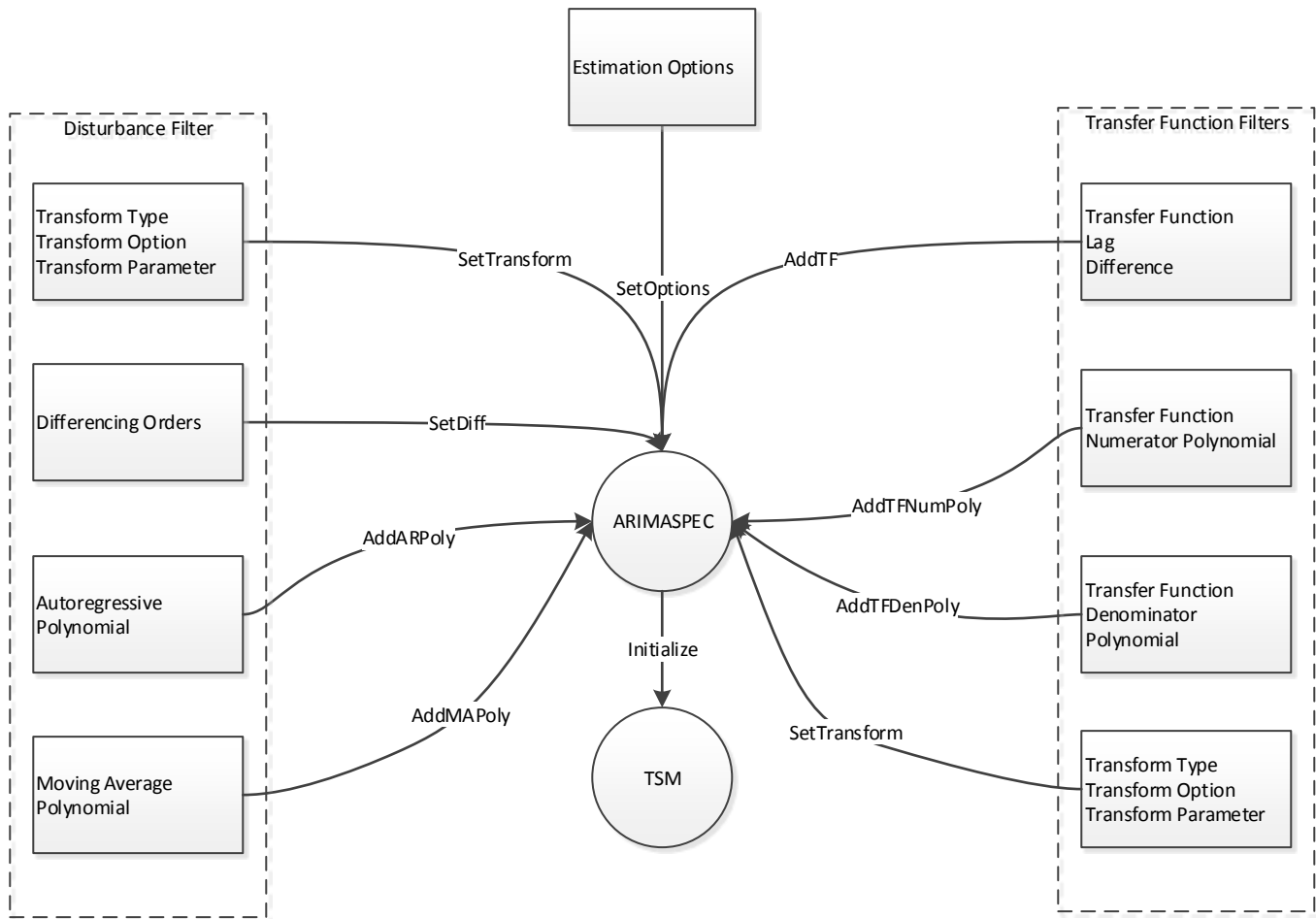
Table 19.7 Methods of the ARIMASPEC Object

Method	Description
AddARPoly	Add an autoregressive polynomial factor to the ARIMA model
AddMAPoly	Add a moving average polynomial factor to the ARIMA model
AddTF	Add a transfer function to the ARIMA model
AddTFDenPoly	Add a transfer function denominator factors to the ARIMA model
AddTFNumPoly	Add a transfer function numerator factors to the ARIMA model
Close	Close the ARIMA model specification
Open	Open the ARIMA model specification
SetDiff	Add differencing to the ARIMA model
SetOption	Specify the ARIMA option
SetTFTransform	Specify the transfer function transform
SetTransform	Specify transform

The basic execution pattern for using an ARIMASPEC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new ARIMASPEC object. At creation, its default state is ARIMA(0,1,0) with intercept (random walk with drift).
- 2 Open:** The ARIMASPEC.Open method readies the default ARIMASPEC object for new configuration.
- 3 Configure:** The various ARIMASPEC.Set methods configure the ARIMASPEC object.
- 4 Close:** The ARIMASPEC.Close method finalizes the ARIMASPEC object.
- 5 Apply:** The TSM.Initialize method adds the ARIMASPEC to a TSM object.

You can also store the XML representation of the ARIMA model in a CAS table. For more information, see the TSMSPC object. Figure 19.4 illustrates the data flow through the ARIMASPEC object.

Figure 19.4 ARIMASPEC Object Data Flow

You can include series transforms such as log or Box-Cox in the specification. Deferred seasonality wildcards for seasonal ARIMA polynomial factors and seasonal differencing lags are also supported. The following list summarizes where these features can be used. For more information, see the method descriptions.

- `ARIMASPEC.SetDiff` supports a seasonal wildcard lag in the `DiffArray`.
- `ARIMASPEC.AddARPoly` and `ARIMASPEC.AddMAPoly` support addition of seasonal ARIMA polynomial factors.
- `ARIMASPEC.AddTF` supports a seasonal wildcard lag in the `DiffArray`.
- `ARIMASPEC.AddTFNumPoly` and `ARIMASPEC.AddTFDenPoly` support addition of seasonal ARIMA polynomial factors for the transfer function numerator and denominator, respectively.
- `ARIMASPEC.SetTransform` applies transforms such as log, square root, logistic, or Box-Cox to the model.

When you specify ARIMA model coefficient values for any model components, you must specify them for all model components without regard to the value of the NOEST argument in the [ARIMASPEC.SetOption](#) method. Failing to specify a complete set of ARIMA model parameters results in an error when you call [ARIMASPEC.Close](#), and the accumulated ARIMA model specification is reset.

ARIMASPEC Synopsis

DECLARE OBJECT *obj* (**ARIMASPEC**) ;

Method syntax, in order of typical usage:

```
rc=obj.Open () ;
rc=obj.SetTransform ('Type' <,'Option', Parm>) ;
rc=obj.SetDiff (DiffArray <,NDiff>) ;
rc=obj.AddARPoly (OrderArray <,NOrder,Seasonal,CoeffArray>) ;
rc=obj.AddMAPoly (OrderArray <,NOrder,Seasonal,CoeffArray>) ;
rc=obj.AddTF (XName, < Delay,DifArray,NDiff>) ;
rc=obj.AddTFNumPoly (XName, NumArray <,NNum,Seasonal,CoeffArray>) ;
rc=obj.AddTFDenPoly (XName, DenArray <,NDen,Seasonal,CoeffArray>) ;
rc=obj.SetTFTransform (XName, 'Type', < Parm>) ;
rc = obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.Close () ;
```

ARIMASPEC Methods

ARIMASPEC.AddARPoly Method

```
rc=obj.AddARPoly (OrderArray <,NOrder,Seasonal,CoeffArray>) ;
```

Adds an autoregressive (AR) polynomial factor to ARIMA model. Additional AR polynomial factors can be added to the ARIMA model with subsequent calls to this method.

Input Arguments

You must specify the following input argument:

OrderArray is a numeric array that specifies AR polynomial lags. Valid values are integers greater than or equal to 1.

You can also specify the following input arguments:

NOrder takes a numeric variable that specifies the number of *OrderArray* values to use. By default, all *OrderArray* values are used.

Seasonal takes a Boolean value (0 or 1) that, when set to 1, specifies that the AR polynomial is seasonal. By default, the AR polynomial is not seasonal and the lags of the AR polynomial are simple.

CoeffArray is a numeric array that specifies the AR polynomial coefficients. If specified, this array must be of the same cardinality as the *OrderArray*.

ARIMASPEC.AddMAPoly Method

```
rc=obj.AddMAPoly (OrderArray < ,NOrder,Seasonal,CoeffArray> ) ;
```

Adds a moving average (MA) polynomial factor to ARIMA model. More MA polynomial factors can be added to the ARIMA model by subsequent calls to the this method.

Input Arguments

You must specify the following input argument:

OrderArray is a numeric array that specifies MA polynomial lags. Valid values are integers greater than or equal to 1.

You can also specify the following input arguments:

NOrder takes a numeric variable that specifies the number of *OrderArray* values to use. By default, all *OrderArray* values are used.

Seasonal takes a Boolean value (0 or 1) that, when set to 1, specifies that the MA polynomial is seasonal. By default, the MA polynomial is not seasonal and the lags of the MA polynomial are simple.

CoeffArray is a numeric array that specifies the MA polynomial coefficients. If specified, this array must be of the same cardinality as the *OrderArray*.

ARIMASPEC.AddTF Method

```
rc=obj.AddTF (XName, < Delay,DifArray,NDiff> ) ;
```

Adds a transfer function to the ARIMA model for the specified *XName* variable. This method adds the variable as a simple scale effect subject to any specified lag and differencing that might be applied.

Input Arguments

You must specify the following input argument:

XName is a character string that specifies the name of the X variable.

You can also specify the following input arguments:

Delay takes a numeric value that specifies the simple delay for the predictor. The default value is 0.

DifArray is a numeric array that specifies differencing orders for the X variable. Valid values are integers greater than or equal to 1. If not specified, then no differencing is applied. Values in *DifArray* are interpreted as follows:

- Negative values are not allowed and result in an error condition.
- Nonnegative values represent differencing orders.
- An .S missing value is interpreted to include seasonal difference order.
- Any other missing value is ignored.

NDiff takes a numeric value that specifies the number of *DifArray* values to use. If not specified, then all the values in *DifArray* are used.

ARIMASPEC.AddTFDenPoly Method

```
rc=obj.AddTFDenPoly (XName, DenArray < ,NDen,Seasonal, CoeffArray> ) ;
```

Adds a transfer function denominator polynomial factor for the specified *XName* variable. More polynomials can be added by subsequent calls to the ARIMASPEC.AddTFNumPoly method.

Input Arguments

You must specify the following input argument:

XName is a character string that specifies the name of the X variable.

DenArray is a numeric array that specifies denominator polynomial lags for the X variable. Valid values are integers greater than 0.

You can also specify the following input arguments:

NDen takes a numeric value that specifies the number of *DenArray* values to use. Valid values are integers greater than 0.

Seasonal takes a Boolean value (0 or 1) that, when set to 1, specifies that the denominator polynomial is seasonal. By default, the denominator polynomial is not seasonal and the lags of the denominator polynomial are simple.

CoeffArray is a numeric array that specifies denominator polynomial coefficients.

ARIMASPEC.AddTFNumPoly Method

```
rc=obj.AddTFNumPoly (XName, NumArray < ,NNum,Seasonal, CoeffArray> ) ;
```

Adds a transfer function numerator polynomial factor for the specified *XName* variable. More polynomials can be added by subsequent calls to the ARIMASPEC.AddTFNumPoly method.

Input Arguments

You must specify the following input argument:

XName is a character string that specifies the name of the X variable.

NumArray is a numeric array that specifies numerator polynomial lags for the X variable. Valid values are integers greater than 0.

You can also specify the following input arguments:

NNum takes a numeric value that specifies the number of *NumArray* values to use. By default, all *NumArray* values are used.

Seasonal takes a Boolean value (0 or 1) that, when set to 1, specifies that the numerator polynomial is seasonal. By default, the numerator polynomial is not seasonal and the lags of the numerator polynomial are simple.

CoeffArray is a numeric array that specifies numerator polynomial coefficients.

ARIMASPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the ARIMASPEC object to prepare the ARIMA model to be used in a TSM object or to be imported to a TSMSPEC object for printing or for storage in a model repository.

Arguments

There are no arguments associated with this method.

ARIMASPEC.Open Method

```
rc=obj.Open () ;
```

Initializes an empty ARIMASPEC object for configuration.

Arguments

There are no arguments associated with this method.

ARIMASPEC.SetDiff Method

```
rc=obj.SetDiff (DiffArray < ,NDiff>) ;
```

Adds differencing to the ARIMA model.

Input Arguments

You must specify the following input argument:

DiffArray is a numeric array that specifies differencing orders, where each order must be an integer greater than or equal to 1. Values in *DiffArray* are interpreted as follows:

- Negative values are not allowed and result in an error condition.
- Nonnegative integer values represent differencing orders.
- An .S missing value is interpreted to include seasonal difference order.
- Any other missing value is ignored.

You can also specify the following input argument:

NDiff takes a numeric value that specifies the number of *DiffArray* values to use. The default is to use all elements of *DiffArray*.

ARIMASPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
```

Specifies ARIMA model options. Options are ('Name', Value) pairs where 'Name' is a case-insensitive character string and Value depends on the 'Name'.

Input Arguments

You must specify at least one of the following *'Names'* and its associated *Value*:

'CONVERGE'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the convergence criterion. Convergence is assumed when the largest change in the estimate for any parameter is less than the specified <i>Value</i> . If the absolute value of the parameter estimate is greater than 0.01, the relative change is used; otherwise, the absolute change in the estimate is used. The default value is 0.001.						
'DELTA'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the perturbation value for computing numerical derivatives. The default value is 0.001.						
'MAXITER'	takes a positive integer <i>Value</i> that specifies the maximum number of iterations allowed. The default is 50.						
'METHOD'	takes a string <i>Value</i> that specifies the estimation method to use. You can specify one of the following <i>Values</i> : <table data-bbox="406 756 1136 892"> <tr> <td>CLS</td><td>specifies the conditional least squares method.</td></tr> <tr> <td>ML</td><td>specifies the maximum likelihood method.</td></tr> <tr> <td>ULS</td><td>specifies the unconditional least squares method.</td></tr> </table> The default is CLS.	CLS	specifies the conditional least squares method.	ML	specifies the maximum likelihood method.	ULS	specifies the unconditional least squares method.
CLS	specifies the conditional least squares method.						
ML	specifies the maximum likelihood method.						
ULS	specifies the unconditional least squares method.						
'MU'	takes a numeric <i>Value</i> that specifies a constant term for the ARIMA model. The default is 0.						
'NOEST'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, specifies that no estimation is performed. The default value is 0 (estimation is performed).						
'NOINT'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, specifies that no intercept is defined. This suppresses the fitting of a constant (or intercept) parameter in the model. That is, the value specified with MU value is omitted. The default value is 0 (intercept is defined).						
'NOSTABLE'	takes a Boolean <i>Value</i> (0 or 1) that, when set to 1, requests that the autoregressive and moving average parameter estimates for the noise part of the model not be restricted to the stationary and invertible regions, respectively. The default is 0 (parameter estimates for the noise part of the model are restricted).						
'SINGULAR'	takes a numeric <i>Value</i> between 0 and 1, exclusive, that specifies the criterion for checking singularity. If a pivot of a sweep operation is less than <i>Value</i> , the matrix is deemed singular. Sweep operations are performed on the Jacobian matrix during final estimation and on the covariance matrix when preliminary estimates are obtained. The default is 1E-7.						

ARIMASPEC.SetTFTransform Method

```
rc=obj.SetTFTransform (XName, 'Type', <Parm>) ;
```

Specifies a functional transform for specified *XName* variable.

Input Arguments

You must specify the following input arguments:

<i>XName</i>	is a character string that specifies the name of the X variable.
'Type'	takes a string value that specifies the functional transform to use. You can specify the following values:
BOXCOX (<i>value</i>)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).
LOG	requests logarithmic transformation.
LOGIT	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	specifies square-root transformation.

The default is NONE.

You can also specify the following input argument:

<i>Parm</i>	takes a numeric value between –5 and 5 that specifies a control argument. This parameter is allowed only for Box-Cox transforms.
-------------	--

ARIMASPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' < ','Option',Parm>);
```

Specifies the functional transform '*Type*' to be used by the ARIMA model. Optional arguments *Option* and *Parm* offer greater control over the transform.

Input Arguments

You must specify the following input argument:

'Type'	takes a string value that specifies the functional transform to use. You can specify the following values:
BOXCOX (<i>value</i>)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).
LOG	requests logarithmic transformation.
LOGIT	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	specifies square-root transformation.

The default is NONE.

You can also specify the following input arguments:

<i>'Option'</i>	takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values: MEAN requests that the inverse transform produce mean forecasts. MEDIAN requests that the inverse transform produce median forecasts. The default is MEAN.
<i>Parm</i>	takes a numeric value between −5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transforms.

EXMSPEC Object

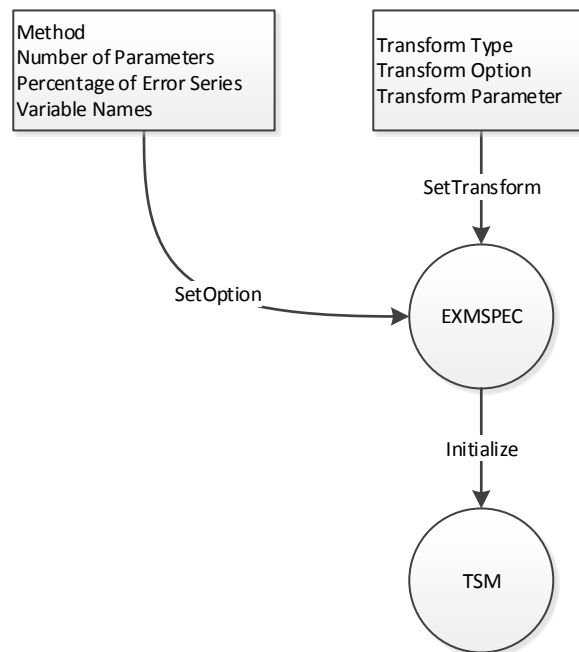
The EXMSPEC object generates an external model (EXM) specification for use with the TSM object, or with the automatic time series modeling (ATSM) package for automatic forecasting services.

Table 19.8 summarizes the methods that are associated with the EXMSPEC object.

Table 19.8 Methods of the EXMSPEC Object

Method	Description
Open	Open EXM model specification
Close	Close EXM model specification
SetTransform	Set transform for EXM model specification
SetOption	Set options for EXM model specification

Figure 19.5 illustrates the data flow through the EXMSPEC object.

Figure 19.5 EXMSPEC Object Data Flow

The basic execution pattern for using an EXMSPEC object follows this sequence of operations:

- 1 Declare:** The object declaration statement creates a new EXMSPEC object.
- 2 Open:** The EXMSPEC.Open method readies the default EXMSPEC object for new configuration.
- 3 Configure:** The various EXMSPEC.Set methods configure the EXMSPEC object.
- 4 Close:** The EXMSPEC.Close method finalizes the EXMSPEC object.
- 5 Apply:** The EXMSPEC is applied to a TSM object using the TSM.Initialize method or is used with the ATSM package. For more information about using the EXMSPEC object, see the section “[TSM Object](#)” on page 1117 or Chapter 3, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

NOTE:

1. You can also store the XML representation of the EXMSPEC to a CAS table. For more information, see the section “[TSMSPEC Object](#)” on page 1164.
2. To run the EXMSPEC in a TSM object, time series variables for the forecast series roles must be added to the TSM object via the TSM.AddExternal method. For more information, see the section “[TSM.AddExternal Method](#)” on page 1119.

3. To run the EXMSPEC in an ATSM:FORENG object, time series variables for the forecast series roles must be added to the ATSM:TSDf object that is used to specify the time series data for the ATSM:FORENG object. When the EXMSPEC object is used with the ATSM package, more sophisticated persistence mechanisms are available. For more information, see Chapter 3, “Automatic Time Series Modeling Package” (*SAS Visual Forecasting: Time Series Packages*).

EXMSPEC Synopsis

DECLARE OBJECT *obj* (EXMSPEC) ;

Method syntax, in order of typical usage:

```
rc=obj.Open () ;
rc=obj.SetTransform ('Type' < ,Option, Parm>) ;
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
rc=obj.Close () ;
```

EXMSPEC Methods

EXMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the EXM model in the EXMSPEC object. This prepares the EXM model for use in a TSM object or to be imported to a TSMsPEC object for printing or storage to a model repository catalog.

Arguments

There are no arguments associated with this method.

EXMSPEC.Open Method

```
rc=obj.Open () ;
```

Initializes an EXMSPEC object for configuration.

Arguments

There are no arguments associated with this method.

EXMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... >) ;
```

Specifies the options for the EXMSPEC object.

Input Arguments

You must specify at least one of the following 'Names' and its associated Value:

'ERROR'	takes a string <i>Value</i> that specifies the variable name for ERROR series.										
'LOWER'	takes a string <i>Value</i> that specifies the variable name for LOWER series.										
'METHOD'	takes a string <i>Value</i> that specifies the method to approximate prediction STDERR series. You can specify one of the following values: <table data-bbox="406 378 1218 630"> <tr> <td>ACF</td><td>Autocorrelation is used.</td></tr> <tr> <td>ERRORACF</td><td>Prediction error autocorrelation is used.</td></tr> <tr> <td>NONE</td><td>No prediction error autocorrelation is used.</td></tr> <tr> <td>PERFECT</td><td>Perfect autocorrelation is assumed.</td></tr> <tr> <td>WN</td><td>Prediction error autocorrelation is white noise.</td></tr> </table> The default is PERFECT.	ACF	Autocorrelation is used.	ERRORACF	Prediction error autocorrelation is used.	NONE	No prediction error autocorrelation is used.	PERFECT	Perfect autocorrelation is assumed.	WN	Prediction error autocorrelation is white noise.
ACF	Autocorrelation is used.										
ERRORACF	Prediction error autocorrelation is used.										
NONE	No prediction error autocorrelation is used.										
PERFECT	Perfect autocorrelation is assumed.										
WN	Prediction error autocorrelation is white noise.										
'NLAGPCT'	takes a numeric <i>Value</i> between 0 and 100 that specifies the percentage of error series count for ACF computations. The default is 25 (25%).										
'NPARMS'	takes a nonnegative integer <i>Value</i> that specifies the number of parameters for the external forecast. The default is 0.										
'PREDICT'	takes a string <i>Value</i> that specifies the variable name for PREDICT series.										
'SIGMA'	takes a numeric <i>Value</i> that specifies the prediction standard error for the external model. If left unspecified, then the prediction mean square error is computed from the prediction errors by using the 'NPARMS' argument. The default is missing value.										
'STDERR'	takes a string <i>Value</i> that specifies the variable name for STDERR series.										
'UPPER'	takes a string <i>Value</i> that specifies the variable name for UPPER series.										

EXMSPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' < ,Option,Parm> ) ;
```

Specifies the functional transform to be used by the EXM model.

Input Arguments

You must specify the following input argument:

'Type'	takes one of the following string values: <table data-bbox="357 1512 1429 1785"> <tr> <td>BOXCOX(value)</td><td>requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).</td></tr> <tr> <td>LOG</td><td>requests logarithmic transformation.</td></tr> <tr> <td>LOGIT</td><td>requests logistic transformation.</td></tr> <tr> <td>NONE</td><td>does not apply a transformation.</td></tr> <tr> <td>SQRT</td><td>requests square-root transformation.</td></tr> </table> The default is NONE.	BOXCOX(value)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).	LOG	requests logarithmic transformation.	LOGIT	requests logistic transformation.	NONE	does not apply a transformation.	SQRT	requests square-root transformation.
BOXCOX(value)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).										
LOG	requests logarithmic transformation.										
LOGIT	requests logistic transformation.										
NONE	does not apply a transformation.										
SQRT	requests square-root transformation.										

You can also specify the following input arguments:

<i>'Option'</i>	takes one of the following string values that specifies prediction semantics for the inverse transform: MEAN requests that the inverse transform produce mean forecasts. MEDIAN requests that the inverse transform produce median forecasts. The default is MEAN.
<i>Parm</i>	takes a numeric value between –5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transforms.

IDMSPEC Object

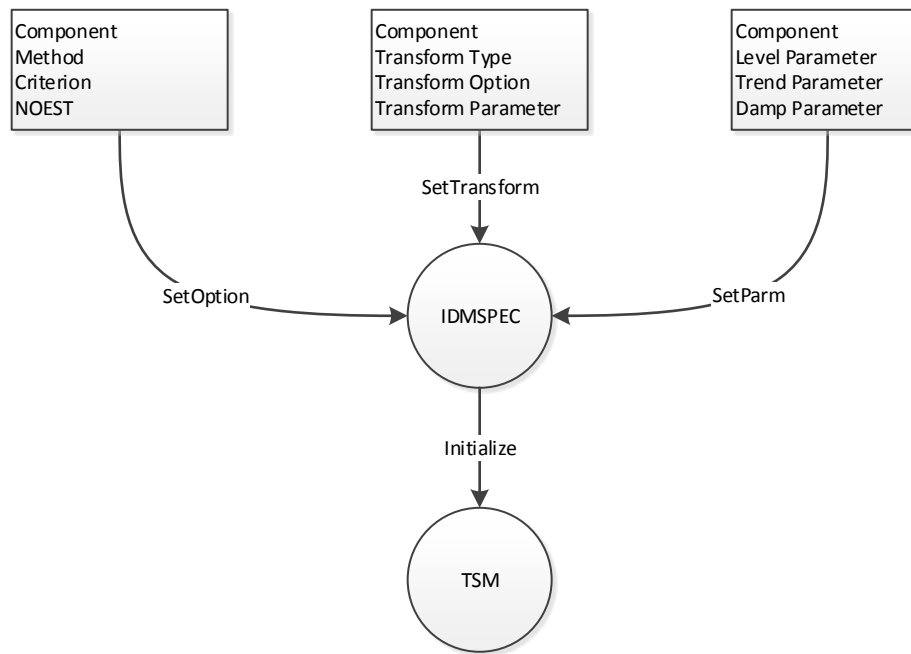
The IDMSPEC object generates intermittent demand models for use with the TSM object. You open the IDMSPEC object to begin defining a new IDM model. You call the IDMSPEC methods to configure the object with the desired settings to define the model characteristics of interest, and then you close the IDMSPEC object to ready it for use in a TSM object. For more information, see the section “[TSM.Initialize Method](#)” on page 1122 method. You can also store the XML representation of the IDM to a CAS table. For more information, see the section “[TSMSPEC Object](#)” on page 1164.

[Table 19.9](#) the methods that are associated with the IDMSPEC object.

Table 19.9 Methods of the IDMSPEC Object

Method	Description
Close	Close the IDM model specification
Open	Open the IDM model specification
SetMethod	Set component smoothing method
SetOption	Set options for IDM model specification
SetParm	Set component parameters
SetTransform	Set component transform

[Figure 19.6](#) illustrates the data flow through the IDMSPEC object.

Figure 19.6 IDMSPEC Object Data Flow

IDMSPEC Synopsis

DECLARE OBJECT *obj* (IDMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.SetTransform ('IDMComp', 'Type' <,'Option',Parm>) ;
rc=obj.SetMethod ('IDMComp', 'Method') ;
rc=obj.SetParm ('IDMComp', 'ESMComp',Parm <.,Noest,LRest, URest>) ;
rc=obj.Close () ;

```

IDMSPEC Methods

IDMSPEC.Close Method

```
rc=obj.Close () ;
```

Finalizes the IDM model in the IDMSPEC object and prepares the IDM model for use in a TSM object or to be imported to a TSMSPEC object for printing or storage to a model repository catalog.

Arguments

There are no arguments associated with this method.

IDMSPEC.Open Method

```
rc=obj.Open () ;
```

Opens the IDMSPEC object for configuration and initializes it with default options. The default options specify that the best IDM model from among the average demand and Croston's model be selected based on the RMSE criterion (see the value BEST for the '**MODEL**' argument in the SetOption method). The smoothing model for the individual components of both models is selected from among the best candidate nonseasonal smoothing method (see value BESTN for the '*Method*' argument in the SetMethod method). In addition, by default, all component transformations are automatically chosen between NONE and LOG based on the model selection criterion (see the value AUTO for the '*Type*' argument in the SetTransform method).

Arguments

There are no arguments associated with this method.

IDMSPEC.SetMethod Method

```
rc=obj.SetMethod ('IDMComp', 'Method') ;
```

Sets a smoothing method for IDM model component.

Input Arguments

You must specify the following input arguments:

'IDMComp' is a character string that specifies the IDM component model. You can specify one of the following values:

AVERAGE	specifies the average demand model.
INTERVAL	specifies demand interval model.
SIZE	specifies demand size model.

'Method' takes a string value that specifies the name of the smoothing method. You can specify one of the following values:

BESTN	requests the best candidate nonseasonal smoothing model among the SIMPLE, LINEAR, or DAMPTREND methods.
DAMPTREND	requests damped trend exponential smoothing.
LINEAR	requests linear (Holt) exponential smoothing.
NONE	disables the use of a smoothing model for the specified component.
SIMPLE	requests simple (single) exponential smoothing.

The default is BESTN.

IDMSPEC.SetOption Method

```
rc=obj.SetOption ('Name', Value < , 'Name', Value, ... > ) ;
```

Specifies the options for the IDMSPEC object.

Input Arguments

You must specify at least one of the following 'Names' and its associated *Value*:

- | | | | | | | | |
|--------------------|--|----------------|--|-------------|--|----------------|---|
| 'BASE' | takes a numeric <i>Value</i> that specifies the base demand level. A 'BASE' value that is specified as a missing numeric value is interpreted to infer the base demand automatically as the median value of the dependent series. | | | | | | |
| 'CRITERION' | takes a string <i>Value</i> that specifies the model selection criterion (statistic of fit) to be used to select from several candidate models. For a list of valid values, see the CRITERION= option in the HPFDIAGNOSE procedure in <i>SAS Forecast Server Procedures: User's Guide</i> . The default is RMSE. | | | | | | |
| 'MODEL' | is a character string that specifies a mnemonic for the IDM model to use. You can specify the following <i>Values</i> : <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 10px;">AVERAGE</td> <td>uses the single smoothing model to fit the average demand component.</td> </tr> <tr> <td style="padding-right: 10px;">BEST</td> <td>requests the best IDM model from among the AVERAGE and CROSTON models.</td> </tr> <tr> <td style="padding-right: 10px;">CROSTON</td> <td>uses the two smoothing models to fit the demand interval component and the demand size component.</td> </tr> </table> | AVERAGE | uses the single smoothing model to fit the average demand component. | BEST | requests the best IDM model from among the AVERAGE and CROSTON models. | CROSTON | uses the two smoothing models to fit the demand interval component and the demand size component. |
| AVERAGE | uses the single smoothing model to fit the average demand component. | | | | | | |
| BEST | requests the best IDM model from among the AVERAGE and CROSTON models. | | | | | | |
| CROSTON | uses the two smoothing models to fit the demand interval component and the demand size component. | | | | | | |

The default is BEST.

IDMSPEC.SetParm Method

```
rc=obj.SetParm ('IDMComp', 'ESMComp', Parm < , Noest, LRest, URest > ) ;
```

Sets parameter value and restrictions for the ESM component in the IDM component model. The ESM method-specific bounds are employed to limit or filter the smoothing method weight values.

Input Arguments

You must specify the following input arguments:

- | | | | | | | | |
|------------------|--|----------------|-------------------------------------|-----------------|----------------------------------|-------------|------------------------------|
| 'IDMComp' | is a character string that specifies the IDM component model. You can specify one of the following values: <table border="0" style="margin-left: 20px;"> <tr> <td style="padding-right: 10px;">AVERAGE</td> <td>specifies the average demand model.</td> </tr> <tr> <td style="padding-right: 10px;">INTERVAL</td> <td>specifies demand interval model.</td> </tr> <tr> <td style="padding-right: 10px;">SIZE</td> <td>specifies demand size model.</td> </tr> </table> | AVERAGE | specifies the average demand model. | INTERVAL | specifies demand interval model. | SIZE | specifies demand size model. |
| AVERAGE | specifies the average demand model. | | | | | | |
| INTERVAL | specifies demand interval model. | | | | | | |
| SIZE | specifies demand size model. | | | | | | |
| 'ESMComp' | is a character string that specifies the ESM component of the IDM component. You can specify one of the following values: | | | | | | |

DAMP	specifies that <i>Parm</i> is the initial value of a damping weight parameter.
LEVEL	specifies that <i>Parm</i> is the initial value of a level weight parameter.
SEASON	specifies that <i>Parm</i> is the initial value of a season weight parameter.
TREND	specifies that <i>Parm</i> is the initial value of a trend weight parameter.

Parm takes a numeric value that specifies the smoothing component weight.

You can also specify the following input arguments:

<i>Noest</i>	takes a Boolean value (0 or 1) that, when set to 1, specifies that the <i>Parm</i> argument is fixed. The default value is 0 (initial).
<i>LRest</i>	takes a numeric value between –1 and 2 that specifies a lower bound restriction on the ESM weight.
<i>URest</i>	takes a numeric value between –1 and 2 that specifies an upper bound restriction on the ESM weight.

IDMSPEC.SetTransform Method

```
rc=obj.SetTransform ('IDMComp', 'Type' <,'Option',Parm>);
```

Sets the functional transform to be used by the IDM model component.

Input Arguments

You must specify the following input arguments:

'IDMComp' is a character string that specifies the IDM component model. You can specify one of the following values:

AVERAGE	specifies the average demand model.
INTERVAL	specifies demand interval model.
SIZE	specifies demand size model.

'Type' takes a string value that specifies the transform to use. You can specify one of the following values:

AUTO	automatically chooses between NONE and LOG based on model selection criteria.
BOXCOX(value)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).
LOG	requests logarithmic transformation.
LOGIT	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.

The default is AUTO.

You can also specify the following input arguments:

'Option'	takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.
	The default is MEAN.
Parm	takes a numeric value between –5 and 5 that specifies a control parameter. This parameter is allowed only for Box-Cox transforms.

UCMSPEC Object

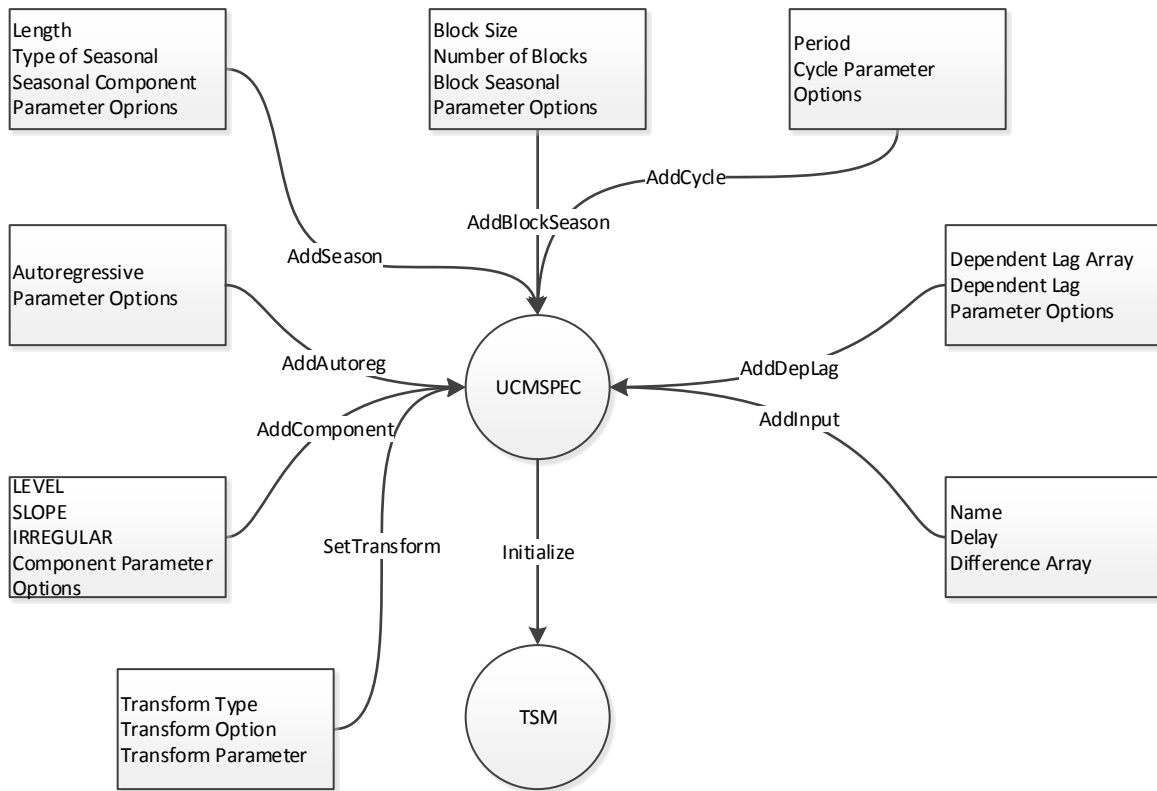
The unobserved component model (UCM) specification object generates UCM models for use with the TSM object. You open the UCMSPEC object to begin defining a new UCM model. You call the UCMSPEC methods to define the model characteristics, and then you close the UCMSPEC object to ready it for use in a TSM object. For more information, see the section “[TSM.Initialize Method](#)” on page 1122. You can also store the XML representation of the UCM in a CAS table. For more information, see the section “[TSM.SPEC Object](#)” on page 1164. Note that the UCMSPEC object is equivalent to the HPFUCMSPEC procedure in *SAS Forecast Server Procedures: User’s Guide*.

Table 19.10 lists the methods that are associated with the UCMSPEC object.

Table 19.10 Methods of the UCMSPEC Object

Method	Description
AddAutoreg	Add an autoregressive component to a UCM model
AddBlockSeason	Add a block-season component to a UCM model
AddComponent	Add a fundamental component to a UCM model
AddCycle	Add a cycle component to a UCM model
AddDeplag	Add a deplag component to a UCM model
AddInput	Add an input component to a UCM model
AddSeason	Add a seasonal component to a UCM model
Close	Close the UCM model specification
Open	Open the UCM model specification
SetTransform	Specify the transform

Figure 19.7 illustrates the data flow through the UCMSPEC object.

Figure 19.7 UCMSPEC Object Data Flow

UCMSPEC Synopsis

DECLARE OBJECT *obj* (UCMSPEC) ;

Method syntax, in order of typical usage:

```

rc=obj.Open () ;
rc=obj.AddAutoreg (< Rho, NoestRho, Variance, Noest>) ;
rc=obj.AddBlockSeason (BlockSize, NBlocks, < 'Type', Offset, Variance, Noest>) ;
rc=obj.AddComponent (Comp, < Variance, Noest>) ;
rc=obj.AddCycle (< Period, NoestPeriod, Rho, NoestRho, Variance, Noest>) ;
rc=obj.AddDeplag (LagArray, < NLag, CoeffArray, Noest>) ;
rc=obj.AddInput (XName, < Delay, DiffArray, NDiff, Transform, TransParm>) ;
rc=obj.AddSeason (Length, < 'Type', Variance, Noest>) ;
rc=obj.SetTransform (XName, < Delay, DiffArray, NDiff, 'Type', TransParm>) ;
rc=obj.Close () ;

```

UCMSPEC Methods

UCMSPEC.AddAutoreg Method

rc=obj.AddAutoreg (< Rho, NoestRho, Variance, Noest>) ;

Adds an autoregressive component to a UCM model specification. You can add at most one autoregressive component to a UCM model specification.

Input Arguments

You can specify the following input arguments:

<i>Rho</i>	takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of <i>Rho</i> must be in the interval $(-1, 1]$. The default is a missing value.
<i>NoestRho</i>	takes a Boolean value that specifies whether the AR(1) coefficient is to be estimated (0) or fixed (1) at the specified starting value. If <i>NoestRho</i> is 1, a valid nonmissing <i>Rho</i> value must be specified. The default value is 0.
<i>Variance</i>	specifies an initial value for the disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value. The default is a missing value.
<i>Noest</i>	takes a Boolean value that specifies whether the variance of the AR(1) noise process is to be estimated (0) or fixed (1). If <i>Noest</i> is 1, a valid nonmissing <i>Variance</i> value must be specified. The default value is 0.

UCMSPEC.AddBlockSeason Method

rc=obj.AddBlockSeason (BlockSize, NBlocks, <'Type', Offset, Variance, Noest>) ;

Adds a block-season component to a UCM model specification. You can use this method repeatedly to add more block-season components to a UCM model specification.

Input Arguments

You must specify the following input arguments:

<i>BlockSize</i>	takes a numeric value that specifies the block size, where <i>BlockSize</i> can be any integer larger than or equal to 2. Typical examples of block sizes are 24 (which corresponds to the hours of the day when a day is used as a block in hourly data) or 60 (which corresponds to the minutes in an hour when an hour is used as a block in data that are recorded by minutes).
<i>NBlocks</i>	takes a numeric value that specifies the number of blocks, where the <i>NBlocks</i> can be any integer greater than or equal to 2.

You can also specify the following input arguments:

<i>'Type'</i>	is a character string that specifies the type of the seasonal component. You can specify one of the following values:
---------------	---

DUMMY specifies a dummy type seasonal component.

TRIG specifies a trigonometric seasonal component.

The default is DUMMY.

Offset takes a numeric value that specifies the offset (the position of the first measurement within the block, if the first measurement is not at the start of a block). The *Offset* value must be an integer between 1 and *BlockSize*.

Variance specifies an initial value for the disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value. The default is a missing value.

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If *Noest* is 1, a valid nonmissing *Variance* value must be specified. The default value is 0.

UCMSPEC.AddComponent Method

```
rc=obj.AddComponent (Comp, < Variance,Noest>);
```

Adds a fundamental component (such as LEVEL, SLOPE, or IRREGULAR) to a UCM model specification. This method must be called separately to add each of these fundamental components. A UCM model specification cannot contain more than one instance of these fundamental components.

Input Arguments

You must specify the following input argument:

Comp is a character string that specifies the UCM component model. You can specify one of the following values:

IRREGULAR includes an irregular component in the model that corresponds to the overall random error in the model.

LEVEL includes a level component in the model. The level component, either by itself or together with a slope component, forms the trend component. If the slope component is absent, the resulting trend is a random walk (RW). If the slope component is present, then a locally linear trend (LLT) is obtained.

SLOPE includes a slope component in the model. If you specify this value, you must call the method again with the value LEVEL for the *Comp* argument.

You can also specify the following input arguments:

Variance specifies an initial value for the disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value. The default is a missing value.

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If *Noest* is 1, a valid nonmissing *Variance* value must be specified. The default value is 0.

UCMSPEC.AddCycle Method

rc=obj.AddCycle (< Period, NoestPeriod,Rho,NoestRho,Variance, Noest>) ;

Adds a cycle component to a UCM model specification. You can add multiple cycle components to a UCM model specification.

Input Arguments

You can specify the following input arguments:

<i>Period</i>	specifies a numeric value as a starting value for the cycle period during the parameter estimation process, where <i>Period</i> can be any number larger than or equal to 2. The default is a missing value.
<i>NoestPeriod</i>	takes a Boolean value that specifies whether the cycle period is to be estimated (0) or fixed (1) at the specified starting value. If <i>NoestPeriod</i> is 1, a valid nonmissing <i>Period</i> value must be specified. The default value is 0.
<i>Rho</i>	takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of <i>Rho</i> must be in the interval $(-1, 1]$. The default is a missing value.
<i>NoestRho</i>	takes a Boolean value that specifies whether the AR(1) coefficient is to be estimated (0) or fixed (1) at the specified starting value. If <i>NoestRho</i> is 1, a valid nonmissing <i>Rho</i> value must be specified. The default value is 0.
<i>Variance</i>	specifies an initial value for the disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value. The default is a missing value.
<i>Noest</i>	takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If <i>Noest</i> is 1, a valid nonmissing <i>Variance</i> value must be specified. The default value is 0.

UCMSPEC.AddDeplag Method

rc=obj.AddDeplag (LagArray, < NLag,CoeffArray,Noest>) ;

Adds a deplag component to a UCM model specification. You can add at most one deplag component to a UCM model specification.

Input Arguments

You must specify the following input argument:

<i>LagArray</i>	specifies an integer array that defines a model with specified lags of the dependent variable included as predictors.
-----------------	---

You can also specify the following input arguments:

<i>NLag</i>	takes an integer that specifies the number of <i>LagArray</i> values to use. The default is all <i>LagArray</i> values.
<i>CoeffArray</i>	is a numeric array that specifies initial coefficients for the lagged dependent variables.

Noest takes a Boolean value that specifies whether lag coefficients are to be estimated (0) or fixed (1). If *Noest* is 1, a valid nonmissing *Variance* value must be specified. The default value is 0.

UCMSPEC.AddInput Method

rc=obj.AddInput (XName, < Delay,DiffArray,NDiff,Transform, TransParm>) ;

Adds an input (predictor variable) to a UCM model specification. You can add multiple inputs to a UCM model specification.

Input Arguments

You must specify the following input argument:

XName is a character variable that specifies the name of the X-variable symbol.

You can also specify the following input arguments:

Delay takes a nonnegative integer value that specifies the lag of the X-variable in the model. If not specified, the *Delay* is 0.

DiffArray is numeric array of nonnegative integers that specifies differencing orders for the X-variable. If not specified, no differencing is applied to the predictor series.

NDiff takes a numeric variable that specifies the number of *DiffArray* values to use. If not specified, the cardinality of *DiffArray* is used.

Transform is character variable that specifies the name of a functional transform. You can specify one of the following values:

BOXCOX(value) requests Box-Cox transformation with a parameter *value* between –5 and 5. The default is BOXCOX(1).

LOG requests logarithmic transformation.

LOGIT requests logistic transformation.

NONE does not apply a transformation.

SQRT requests square-root transformation.

The default is NONE.

TransParm takes a numeric value that specifies a transform parameter. This argument is used only for Box-Cox transformations.

UCMSPEC.AddSeason Method

rc=obj.AddSeason (Length, <'Type', Variance,Noest>) ;

Adds a seasonal component to a UCM model specification. You can add multiple seasonal component to a UCM model specification.

Input Arguments

You must specify the following input argument:

Length is a numeric variable that specifies the length of a seasonal cycle.

You can also specify the following input arguments:

'*Type*' is a character string that specifies the type of the seasonal component. You can specify one of the following values:

DUMMY specifies a dummy type seasonal component.

TRIG specifies a trigonometric seasonal component.

The default is DUMMY.

Variance specifies an initial value for the disturbance variance during the parameter estimation process. Any nonnegative value, including 0, is an acceptable starting value. The default is a missing value.

Noest takes a Boolean value that specifies whether the disturbance variance is to be estimated (0) or fixed (1). If *Noest* is 1, a valid nonmissing *Variance* value must be specified. The default value is 0.

UCMSPEC.Close Method

```
rc=obj.Close ();
```

Finalizes the UCM model in the UCMSPEC object. This method prepares the UCM model for use in a TSM object or to be imported to a TSMSPEC object for printing or storage to a model repository catalog.

Arguments

There are no arguments associated with this method.

UCMSPEC.Open Method

```
rc=obj.Open ();
```

Opens the UCMSPEC object for configuration. This method initializes an empty UCM model.

Arguments

There are no arguments associated with this method.

UCMSPEC.SetTransform Method

```
rc=obj.SetTransform ('Type' <,'Option',Parm>);
```

Specifies the functional transform to be used by the UCM model.

Input Arguments

You must specify the following input argument:

'Type'	takes a string value that specifies the transform to use. You can specify one of the following values:
BOXCOX (<i>value</i>)	requests Box-Cox transformation with a parameter <i>value</i> between –5 and 5. The default is BOXCOX(1).
LOG	requests logarithmic transformation.
LOGIT	requests logistic transformation.
NONE	does not apply a transformation.
SQRT	requests square-root transformation.
	The default is NONE.
'Option'	takes a string value that specifies prediction semantics for the inverse transform. You can specify the following values:
MEAN	requests that the inverse transform produce mean forecasts.
MEDIAN	requests that the inverse transform produce median forecasts.
	The default is MEAN.
<i>Parm</i>	takes a numeric value between –5 and 5 that specifies a control parameter for the functional transform. This argument is allowed only for Box-Cox transforms.

TSMPEST Object

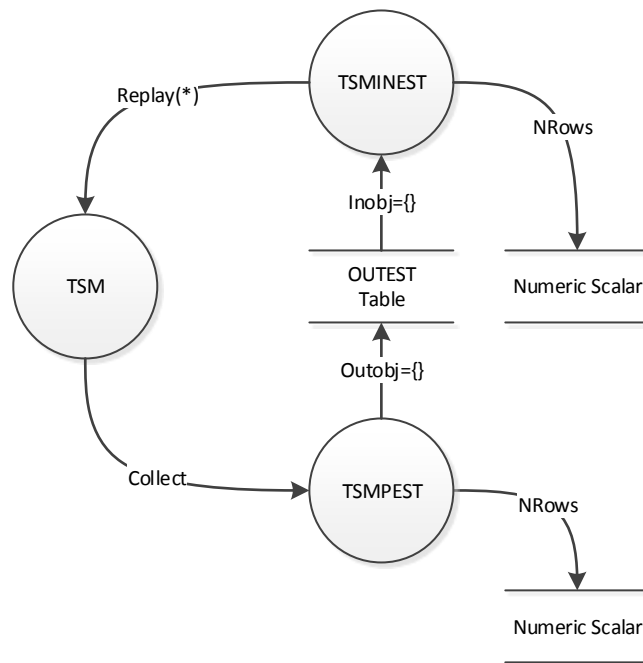
The time series model parameter estimates (TSMPEST) collector object collects parameter estimates from a TSM object and stores them in a CAS table for printing or archiving or for use by a repeater object on another TSM object.

Table 19.11 summarizes the methods that are associated with the TSMPEST object.

Table 19.11 Methods of the TSMPEST Object

Method	Description
Collect	Collect parameter estimates from TSM objects
nrows	Return the number of rows that are collected
SetOption	Set options for the TSMPEST collector object

Figure 19.8 illustrates the data flow through the TSMPEST object.

Figure 19.8 TSMPEST Object Data Flow

TSMPEST Synopsis

DECLARE OBJECT *obj* (TSMPEST) ;

Method syntax, in order of typical usage:

```

rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;

```

TSMPEST Methods

TSMPEST.Collect Method

rc=**obj.Collect** (TSMObj) ;

Collects time series model parameter estimates from a TSM object, *TSMObj*, and stores them in a CAS table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of time series model parameter estimates.

TSMPEST.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

TSMPEST.SetOption Method

```
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
```

Specifies the options for the TSMPEST object.

Input Arguments

You must specify at least one of the following *'Names'* and its associated *Value*:

'MODELNAME' takes a string *Value* that specifies the name of the model to be stored into the CAS table along with the parameter estimates of the model. By default, the model name is the name of the TSM object instance that is collected by the [TSMPEST.Collect](#) method.

'FORCEMODELNAME' takes a boolean *Value* that specifies whether the model name should be stored into the CAS table even when the model failed to run. You can specify one of the following values:

DISABLE|NO|N|FALSE|F|OFF Do not store the model name into the CAS table if the model failed to run.

ENABLE|YES|Y|TRUE|T|ON Always store the model into the CAS table.

The default is DISABLE.

TSMSPEC Object

The time series model specification (TSMSPEC) collector object collects the model specification from a TSM object, *TSMObj*, and stores it in a CAS table for printing or archiving or for use by a repeater object on another TSM object.

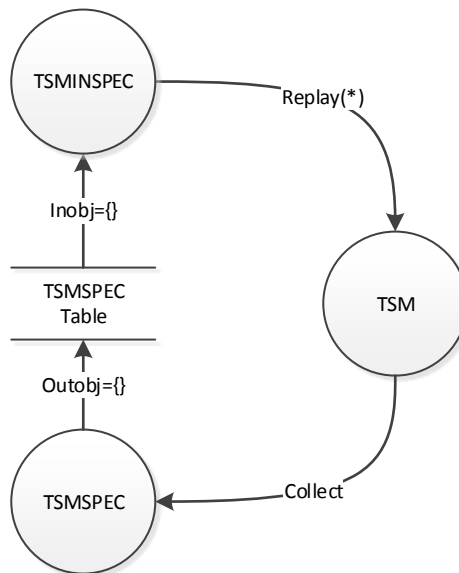
Table 19.12 summarizes the methods that are associated with the TSMSPEC object.

Table 19.12 Methods of the TSMSPEC Object

Method	Description
Collect	Collect model specification from TSM objects
nrows	Return the number of rows that are collected

Figure 19.9 illustrates the data flow through the TSMSPEC object.

Figure 19.9 TSMSPEC Object Data Flow



TSMSPEC Synopsis

DECLARE OBJECT *obj* (TSMSPEC) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;
```

TSMSPEC Methods

TSMSPEC.Collect Method

```
rc=obj.Collect (TSMObj) ;
```

Collects time series model specifications from a TSM object, *TSMObj*, and stores them in a CAS table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of time series model parameter estimates.

TSMSPEC.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

TSMFOR Object

The TSMFOR object collects forecast series from a TSM object, *TSMObj*, and stores them in a CAS table. The CAS table schema that is used for storing the set of forecast series variables is compatible with the schema that the HPFENGINE procedure uses for the data set that is specified in the TSMFOR= option. Alternatively, the TSMFOR object can be configured to collect from a TSM object the smoothed (imputed) values for the dependent series and their associated standard errors and confidence limits.

Table 19.13 shows the contents of the TSMFOR object.

Table 19.13 Contents of the TSMFOR Object

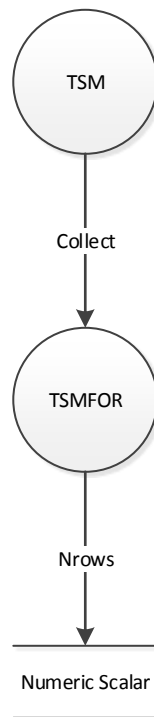
Column	Type	Description
NAME	String	Name of the dependent variable
MODEL	String	(Optional) Model name
TIMEID	Numeric	Uniform time ID values for series
ACTUAL	Numeric	Accumulated values of dependent variable
ERROR	Numeric	Residuals
LOWER	Numeric	Lower confidence limit
PREDICT	Numeric	Forecasts of the dependent variable
STD	Numeric	Prediction standard error
UPPER	Numeric	Upper confidence limit

Table 19.14 summarizes the methods that are associated with the TSMFOR object.

Table 19.14 Methods of the TSMFOR Object

Method	Description
Collect	Collect forecasts estimates from the TSM object
nrows	Return the number of rows collected
SetOption	Specifies options for the TSMFOR instance

Figure 19.10 illustrates the data flow through the TSMFOR object.

Figure 19.10 TSMFOR Object Data Flow

TSMFOR Synopsis

DECLARE OBJECT *obj* (TSMFOR <('Name', Value)>) ;

Method syntax, in order of typical usage:

```

rc=obj.SetOption (Name, Value) ;
rc=obj.Collect (ModelObj <,Region>) ;
rc=obj.nrows () ;

```

Input Arguments

You can optionally specify one or more of the following '*Names*' and its associated *Value*:

'MODELNAME' takes a string *Value* that specifies whether to add a model name column to the output CAS table. The default model name is the TSM instance name. You can override the default model name via the 'MODELNAME' option of the [SetOption](#) method. You can specify one of the following *Values*:

YES	includes a model name column in the output CAS table.
NO	does not include a model name column in the output CAS table.

The default is NO.

'SMOOTH' takes a string *Value* that specifies whether to output the smoothed (imputed) dependent series values and their associated standard errors and confidence limits. If you set this option to 'YES', the PREDICT column holds the values of the smoothed dependent series instead of the forecast series. You can specify one of the following *Values*:

YES outputs the smoothed dependent series and its associated statistics from a TSM instance to a CAS table.

NO outputs the forecast series and its associated statistics from a TSM instance to a CAS table.

The default is NO.

TSMFOR Methods

TSMFOR.Collect Method

```
rc=obj.Collect (ModelObj<,Region>) ;
```

Collects the forecast series from a TSM object, *ModelObj*. An optional *Region* argument can be specified to indicate the forecast region to be collected. If you specify the 'SMOOTH' option in the declaration of the TSMFOR instance, then the smoothed (imputed) dependent series is collected instead.

Input Arguments

You must specify the following input argument:

ModelObj is a character string that specifies the name of the TSM object to use as the source of time series model forecasts.

You can also specify the following input argument:

Region specifies the time region over which to collect the forecast series. You can specify the following values for *Region*:

string specifies the collection region. You can specify the following *strings*:

ALL collects over the entire time span of the available data.

FIT collects over the time region that supplied observations to estimate model parameters (that is, the model fit region).

FORECAST collects over the time region that is subsequent to the FIT region and that did not contribute any data to the model parameter estimation process (that is, the model forecast region).

The default is ALL.

numeric is a two-valued numeric array, in which the first value specifies the starting time ID and the second value specifies the ending time ID of the time region over which the forecast series are to be collected. Either the starting time ID or the ending time ID can be a missing value. If both are missing values, then the default value ALL is used.

TSMFOR.nrows Method

```
rc=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

TSMFOR.SetOption Method

```
rc=obj.SetOption ('Name', Value) ;
```

Specifies the named options for the TSMFOR instance.

Input Arguments

You must specify the following '*Name*' and its associated *Value*:

'MODELNAME' takes a string *Value* that is copied to the `_MODEL_` column of the output CAS table. This option is relevant only if you set the 'MODELNAME' option to 'YES' in the declaration of the TSMFOR instance. The default value is the TSM instance name.

TSMSTAT Object

The TSMSTAT object collects model statistics of fit from a TSM object, *TSMObj*, and stores them in a CAS table. This information is useful for evaluating how well the model fits the dependent series. The CAS table schema that is used for storing the fit statistics is compatible with the schema that the HPFENGINE procedure uses for its OUTSTAT= data set.

Table 19.15 shows the contents of the TSMSTAT object.

Table 19.15 Contents of the TSMSTAT Object

Column	Type	Description
NAME	String	Name of the dependent variable
REGION	String	Region in which the fit statistics are calculated. You can specify the following values: FIT indicates that fit statistics were calculated over the fit region. FORECAST indicates that fit statistics were calculated over the forecast (back) region. HOLDOUT indicates that fit statistics were calculated over the holdout region.
MODEL	String	Model name
AADJRSQ	Numeric	Amemiya's adjusted R-square
ADJRSQ	Numeric	Adjusted R-square
AIC	Numeric	Akaike's information criterion
AICC	Numeric	Finite sample corrected AIC
APC	Numeric	Amemiya's prediction criterion
DFE	Numeric	Degrees of freedom error
GMAPE	Numeric	Geometric mean absolute percentage error
GMAPES	Numeric	Geometric mean absolute error as a percentage of standard deviation
GMAPPE	Numeric	Geometric mean absolute predictive percentage error
GMRAE	Numeric	Geometric mean relative absolute error
GMASPE	Numeric	Geometric mean absolute symmetric percentage error
MAE	Numeric	Mean absolute error
MAPE	Numeric	Mean absolute percentage error
MAPES	Numeric	Mean absolute error as a percentage of standard deviation
MAPPE	Numeric	Mean absolute predictive percentage error
MASE	Numeric	Mean absolute scaled error
MAXAPES	Numeric	Maximum absolute error as a percentage of standard deviation
MAXERR	Numeric	Maximum error
MAXPE	Numeric	Maximum percentage error

Table 19.15 *continued*

Column	Type	Description
MAXPPE	Numeric	Maximum predictive percentage error
MAXRE	Numeric	Maximum relative error
MAXSPE	Numeric	Maximum symmetric percentage error
MDAPE	Numeric	Median absolute percentage error
MDAPES	Numeric	Median absolute error as a percentage of standard deviation
MDAPPE	Numeric	Median absolute predictive percentage error
MDASPE	Numeric	Median absolute symmetric percentage error
MDRAE	Numeric	Median relative absolute error
ME	Numeric	Mean error
MINAPES	Numeric	Minimum absolute error as a percentage of standard deviation
MINERR	Numeric	Minimum error
MINPE	Numeric	Minimum percentage error
MINPPE	Numeric	Minimum predictive percentage error
MINRE	Numeric	Minimum relative error
MINSPE	Numeric	Minimum symmetric percentage error
MPE	Numeric	Mean percentage error
MPPE	Numeric	Mean predictive percentage error
MRAE	Numeric	Mean relative absolute error
MRE	Numeric	Mean relative error
MSE	Numeric	Mean square error
MSPE	Numeric	Mean symmetric percentage error
N	Numeric	Number of observations that were used
NMISSA	Numeric	Number of missing actual values
NMISSP	Numeric	Number of missing predicted values
NOBS	Numeric	Number of observations
NPARMS	Numeric	Number of parameters
RMSE	Numeric	Root mean square error
RSQUARE	Numeric	R-square
RWRSQ	Numeric	Random walk R-square

Table 19.15 *continued*

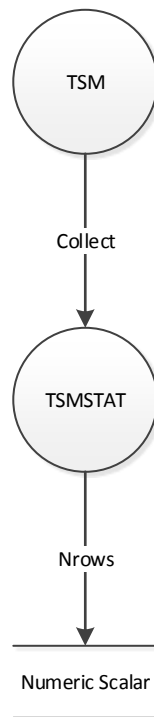
Column	Type	Description
SBC	Numeric	Schwarz Bayesian information criterion
SMAPE	Numeric	Symmetric mean absolute percentage error
SSE	Numeric	Sum of square error
SST	Numeric	Corrected total sum of squares
TSS	Numeric	Total sum of squares
UMSE	Numeric	Unbiased mean square error
URMSE	Numeric	Unbiased root mean square error

Table 19.16 summarizes the methods that are associated with the TSMSTAT object.

Table 19.16 Methods of the TSMSTAT Object

Method	Description
Collect	Collect model statistics of fit from the TSM object
nrows	Return the number of rows collected
SetOption	Set options for the TSMSTAT collector object

Figure 19.11 illustrates the data flow through the TSMSTAT object.

Figure 19.11 TSMSTAT Object Data Flow

TSMSTAT Synopsis

DECLARE OBJECT *obj* (TSMSTAT) ;

Method syntax, in order of typical usage:

```

rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
rc=obj.Collect (TSMObj) ;
nrows=obj.nrows () ;

```

TSMSTAT Methods

TSMSTAT.Collect Method

```
rc=obj.Collect (TSMObj) ;
```

Collects model statistics of fit from a TSM object, *TSMObj*, and stores them in a CAS table.

Input Arguments

You must specify the following input argument:

TSMObj specifies the TSM object to use as the source of model fit statistics.

TSMSTAT.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

TSMSTAT.SetOption Method

```
rc=obj.SetOption ('Name', Value <,'Name',Value,... >) ;
```

Specifies the options for the TSMSTAT object.

Input Arguments

You must specify at least one of the following '*Names*' and its associated *Value*:

'MODELNAME' takes a string *Value* that specifies the name of the model to be stored into the CAS table along with the parameter estimates of the model. By default, the model name is the name of the TSM object instance that is collected by the [TSMSTAT.Collect](#) method.

'FORCEMODELNAME' takes a boolean *Value* that specifies whether the model name should be stored into the CAS table even when the model failed to run. You can specify one of the following values:

DISABLE|NO|N|FALSE|F|OFF Do not store the model name into the CAS table if the model failed to run.

ENABLE|YES|Y|TRUE|T|ON Always store the model into the CAS table.

The default is DISABLE.

TSMINEST Object

The time series model input estimates (TSMINEST) repeater object imports parameter estimates from a CAS table for use in a TSM object. The TSMINEST table schema required for the parameter estimates is compatible with that used by the TSMPEST collector object. The TSMINEST object must be used in conjunction with a TSMINSPEC specification object and a TSM model object to replay a model by using its saved parameters (for an example, see [Example 19.3](#)).

Table 19.17 summarizes the methods of the TSMINEST object.

Table 19.17 Methods of the TSMINEST Object

Method	Description
<code>nrows</code>	Return number of rows in the TSMINEST object

TSMINEST Synopsis

```
DECLARE OBJECT obj (TSMINEST) ;
```

Method syntax:

```
rc=obj.nrows () ;
```

TSMINEST Methods

TSMINEST.nrows Method

```
rc=obj.nrows () ;
```

Returns the number of rows in the TSMINEST object, *obj*. A returned missing value indicates that the object has not been successfully configured.

Arguments

There are no arguments associated with this method.

TSMINSPEC Object

The time series model input specification (TSMINSPEC) repeater object imports model specifications from a CAS table for use in a TSM object. The TSMINSPEC table schema required for the model specification is compatible with that used by the TSMSPEC collector object.

Table 19.18 summarizes the methods of the TSMINSPEC object.

Table 19.18 Methods of the TSMINSPEC Object

Method	Description
<code>nrows</code>	Return number of rows in TSMINSPEC object

TSMINSPEC Synopsis

DECLARE OBJECT *obj* (TSMINSPEC) ;

Method syntax:

rc obj.nrows () ;

TSMINSPEC Methods

TSMINSPEC.nrows Method

rc=obj.nrows () ;

Returns the number of rows that have been collected and stored in the CAS table. A returned missing value indicates that the TSMINSPEC object has not been successfully configured.

Arguments

There are no arguments associated with this method.

Examples: TSM Package

Throughout this section it is assumed that you have already started a CAS session and the data tables that are used in this section are stored in `mycas`, a CAS library that you have necessary permissions to work with. This section assumes that you are familiar with the general workings of the TSMODEL procedure; for more information, see Chapter 11, “The TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named `mysess`, and the LIBNAME statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Example 19.1: Fitting and Forecasting with ARIMA and ESM Models

The airline passenger data, given as Series G in Box and Jenkins (1976), have been used in time series analysis literature as an example of a nonstationary seasonal time series; for more information about ARIMA modeling of Series G, see “Example 7.2 Seasonal Model for the Airline Series” in the ARIMA chapter of *SAS/ETS User’s Guide*.

This example shows how you can use the objects in the TSM package to fit the airline model, $ARIMA(0,1,1) \times (0,1,1)_{12}$ NOINT and to fit the Winters exponential smoothing model to the airline series. In particular, it shows how you can do the following:

1. Create ARIMA and ESM specifications by using the ARIMASPEC and ESMSPEC specification objects.
2. Use these specification objects to initialize the TSM model objects.
3. Use these TSM model objects to fit the airline model and Winters model to the airline series, and to forecast the series according to these models.
4. Postprocess the forecast results to compute an ad hoc statistic.
5. Output the computed results to CAS tables.

The broad outline of the code in this example is as follows:

1. The PROC TSMODEL statement specifies the input data set (mycas.air), a variety of output tables (mycas.airFor, mycas.airEst, and so on), and the forecast lead (12).
2. The ID statement specifies date as the time index variable, and the INTERVAL= option indicates that the data are monthly.
3. The VAR statement specifies the input data set variable, air, which contains the airline series.
4. The OUTARRAYS and OUTSCALARS statements declare some output arrays and scalars that are used to store the analysis results, which are subsequently saved as CAS tables.
5. The REQUIRE statement specifies the TSM package, which is needed for the analysis.
6. The statements between the SUBMIT and ENDSUBMIT statements use the TSM package objects to perform the actual analysis in your CAS session.
7. These statements are grouped in three sections:
 - The first section does the specification, fitting, and forecasting according to the airline model. The airSpec object contains the airline specification, and the airModel object is a TSM model object that is initialized by using airSpec. After the airModel is run, the parameter estimates are collected in airEst and the forecasts are collected in airFor. The model residuals are stored in the airErr array for later processing.

- The second section does the specification, fitting, and forecasting according to the Winters exponential smoothing model. The `esmSpec` object contains the Winters specification, and the `esmModel` object is a TSM model object that is initialized by using `esmSpec`. The `esmModel` is run, and the model residuals are stored in the `esmErr` array for later processing.
- The third section computes an ad hoc statistic called `nbetter`, which counts the number of times the airline model residuals are smaller (in absolute size) than the Winters model. This section is included to illustrate how you can write your own custom postprocessing code to analyze the results that are produced by the TSM objects.

```
proc tsmmodel data=mycas.air
    outobj=(airFor=mycas.airFor airEst=mycas.airEst)
    outscalar=mycas.nbetter outarray=mycas.out lead=12;
id date interval=month;
var air;
outarrays esmErr airErr;    **Store residuals of ESM and ARIMA models;
outscalars nbetter nfor;
require tsm;
submit;

    **Temporary work arrays used in ARIMA spec;
    array diff[2]/nosymbols;
    array ma[1]/nosymbols;

    *** Analysis based on the airline model ***;
    declare object airModel(tsm);
    declare object airSpec(arimaspec);

    **Set up the airline model spec:**;
    ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint *;
    rc = airSpec.Open( );
    *** Specify differencing orders ***;
    diff[1] = 1;
    diff[2] = 12;
    rc = airSpec.SetDiff(diff,2);
    *** Specify moving average orders: q = (1)(12) ***;
    *** Use AddMAPoly twice for the two factors ***;
    ma[1] = 1;
    rc = airSpec.AddMAPoly(ma);
    ma[1] = 12;
    rc = airSpec.AddMAPoly(ma);
    *** Specify NOINT ***;
    rc = airSpec.SetOption('noint',1);
    *** Specify the log transform ***;
    rc = airSpec.SetTransform('log');
    *** Done setting up the ARIMA model ***;
    rc = airSpec.Close( );

    *** Set up and run the airModel TSM object ***;
    rc = airModel.Initialize(airSpec);
    rc = airModel.SetY(Air);
    rc = airModel.SetOption('lead',12);
    rc = airModel.Run( );
```

```

*** Output the airline model forecasts and estimates ***;
declare object airFor(tsmfor);
declare object airEst(tsmpest);
rc = airFor.Collect(airModel);
rc = airEst.Collect(airModel);

***Put the airline model residuals in airErr array;
rc = airModel.getForecast('error',airErr);

*** Analysis based on ESM model ***;
declare object esmModel(tsm);
declare object esmSpec(esmspec);
rc = esmspec.open( );
rc = esmspec.SetOption('method', 'winters');
rc = esmspec.close( );

***Set up and run the TSM object ***;
rc = esmModel.Initialize(esmspec);
rc = esmModel.SetY(Air);
rc = esmModel.SetOption('lead',12);
rc = esmModel.Run( );

**Put the ESM model residuals in esmErr array;
rc = esmModel.getForecast('error',esmErr);

***Compute an ad hoc statistic based on airErr and esmErr arrays;
nbetter = 0;
nfor = esmModel.nfor( );
do t=1 to nfor;
    if airErr[t] ^= . & esmErr[t] ^= . then do;
        if abs(airErr[t]) < abs(esmErr[t])
            then nbetter = nbetter + 1;
    end;
end;
endsubmit;
quit;

```

Output 19.1.1 shows the predictions, and Output 19.1.2 shows the parameter estimates for the airline model.

Output 19.1.1 Airline Model Predictions (Partial Output)**Airline Model Predictions**

DATE	PREDICT	STD	UPPER	LOWER
JAN1961	450.4	16.9215	484.5	418.2
FEB1961	426.1	18.8590	464.2	390.3
MAR1961	480.1	24.0408	528.9	434.7
APR1961	492.8	27.2405	548.3	441.6
MAY1961	509.5	30.5863	572.0	452.1
JUN1961	584.2	37.6514	661.4	513.9
JUL1961	670.7	45.9957	765.3	585.1
AUG1961	668.2	48.4237	768.0	578.3
SEP1961	559.6	42.6271	647.7	480.7
OCT1961	498.3	39.7181	580.6	425.0
NOV1961	431.2	35.8240	505.5	365.2
DEC1961	478.9	41.3484	565.0	403.0

Output 19.1.2 Airline Model Parameter Estimates (Partial Output)**Airline Model Parameter Estimates**

<u>_EST_</u>	<u>_STDERR_</u>	<u>_TVALUE_</u>	<u>_PVALUE_</u>
0.3773	0.0820	4.6033	9.828E-6
0.5724	0.0780	7.3361	2.17E-11

Output 19.1.3 shows the number of times the airline model residuals are smaller than the Winters model residuals.

Output 19.1.3 Number of Times the Airline Model Residuals Are Smaller Than the Winters Model Residuals

<u>nbetter</u>
71.0000

Example 19.2: Fitting a Transfer Function Model

This example uses the gas furnace data from Box and Jenkins (1976). The data, called Series J by Box and Jenkins, contain sequentially recorded measurements of two variables: x , the input gas rate, and y , the output CO_2 . The data also include an index variable, time, which keeps track of the sequence number of each observation (essentially the row index). The TSMODEL procedure requires an ID variable that has a valid time interval associated with it to index the observations. In order to satisfy this requirement, the time variable is assigned as the time ID variable in the ID statement and its interval is specified as SECOND using the INTERVAL= option. The value INTERVAL=SECOND is one of the simplest interval types for sequential indexing. As shown in “Example 7.3 Model for Series J Data from Box and Jenkins” in the ARIMA chapter of *SAS/ETS User's Guide*, a reasonable ARIMAX model for y turns out to be $y = \text{TFinput}(x) + \text{AR}(2)$, where $\text{TFinput}(x)$ is a transfer function term in x with a delay of 3, numerator polynomial of order 2, and

denominator polynomial of order 1, and where AR(2) is an error term of autoregressive order 2. The following statements show how to fit this model by using the objects in the TSM package:

```
proc tsmodel data=mycas.seriesj
  outobj=(jEst=mycas.jEst);
  id time interval=second;
  var x y;
  require tsm;
  submit;

  *** Transfer function modeling for seriesJ ***;
  declare object jModel(tsm);
  declare object jSpec(arimaspec);
  declare object jEst(tsmpest);

  array num[2]/nosymbols;
  array den[1]/nosymbols;
  array ar[2]/nosymbols;

  *** Set up the transfer function model spec: ***;
  rc = jSpec.Open( );

  *** Specify AR orders: p = (1 2) ***;
  ar[1] = 1;
  ar[2] = 2;
  rc = jSpec.AddARPoly(ar);

  rc = jSpec.AddTF('x', 3); *delay=3;
  num[1] = 1;
  num[2] = 2;
  rc = jSpec.AddTFNumPoly('x', num);
  den[1] = 1;
  rc = jSpec.AddTFDenPoly('x', den);
  *** done setting up ARIMA model ***;
  rc = jSpec.Close( );

  *** Set up and run the TSM object ***;
  rc = jModel.Initialize(jSpec);
  rc = jModel.SetY(y);
  rc = jModel.AddX(x);
  rc = jModel.Run( );

  *** output gas furnace model forecasts and estimates ***;
  rc = jEst.Collect(jModel);
endsubmit;
quit;
```

Output 19.2.1 shows the parameter estimates for the transfer function model.

Output 19.2.1 Parameter Estimates for the Transfer Function Model (Partial Output)

<u>_EST_</u>	<u>_STDERR_</u>	<u>_TVALUE_</u>	<u>_PVALUE_</u>
53.2630	0.1193	446.5	0
1.5329	0.0475	32.2472	6.25E-97
-0.6330	0.0501	-12.6434	2.27E-29
-0.5352	0.0748	-7.1534	7.21E-12
0.3760	0.1029	3.6553	0.000306
0.5189	0.1078	4.8124	2.425E-6
0.5484	0.0382	14.3499	1.72E-35

Example 19.3: Replaying a Previously Fitted Model

In some cases, it is useful to save the model specification and parameter estimates that are computed during an analysis for later use. For example, you can use the saved model specification and parameter estimates to produce model forecasts at a later stage (possibly with new measurements appended to the original data). This example shows how you can do the following:

1. Save the model specification and parameter estimates for later use by using the Collect method of the TSMSPec and TSMPEst objects, respectively.
2. Reuse the previously saved model specification and parameter estimates to configure a TSM object by using the Replay method.
3. Produce the model forecasts by using this TSM object.

The following statements fit the airline model to the airline series (see [Example 19.1](#) for more information about the airline series and the airline model). The model specification and parameter estimates are stored in CAS tables mycas.airOSpec and mycas.airEst, respectively.

```
proc tsmmodel data=mycas.air
  outobj=(airEst=mycas.airEst airOSpec=mycas.airOSpec) ;
  id date interval=month;
  var air;
  require tsm;
  submit;

  *** Analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);
  declare object airEst(tsmpest);
  declare object airOSpec(tsmspec);

  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Set up the airline model spec: ***;
  ** Model: log(air) ~ (0,1,1) (0,1,1)12 noint ***;
  rc = airSpec.Open();
```

```

*** Specify differencing orders ***;
diff[1] = 1;
diff[2] = 12;
rc = airSpec.SetDiff(diff,2);
*** Specify moving average orders: q = (1)(12) ***;
*** Use AddMAPoly twice for the two factors ***;
ma[1] = 1;
rc = airSpec.AddMAPoly(ma);
ma[1] = 12;
rc = airSpec.AddMAPoly(ma);
*** Specify NOINT ***;
rc = airSpec.SetOption('noint',1);
*** Specify the log transform ***;
rc = airSpec.SetTransform('log');
*** Done setting up the ARIMA model ***;
rc = airSpec.Close();

*** Set up and run the TSM object ***;
rc = airModel.Initialize(airSpec);
rc = airModel.SetY(Air);
rc = airModel.SetOption('lead',12);
rc = airModel.Run();

*** Output airline model spec and estimates ***;
rc = airEst.Collect(airModel);
rc = airOSpec.Collect(airModel);
endsubmit;
quit;

```

Output 19.3.1 shows the parameter estimates that are saved in mycas.airEst.

Output 19.3.1 Parameter Estimates for the Airline Model (Partial Output)

Airline Model Parameter Estimates

<u>_EST_</u>	<u>_STDERR_</u>	<u>_TVALUE_</u>	<u>_PVALUE_</u>
0.3773	0.0820	4.6033	9.828E-6
0.5724	0.0780	7.3361	2.17E-11

The following statements show how to forecast the airline series by using the previously saved model specification (mycas.airOSpec) and parameter estimates (mycas.airEst).

```

proc tsmode1 data=mycas.air
    outobj=(airFor=mycas.airFor)
    inobj=(airEst=mycas.airEst airSpec=mycas.airOSpec) ;
id date interval=month;
var air;
require tsm;
submit;

*** Analysis based on the airline model ***;
declare object airModel(tsm);
declare object airSpec(tsm1nspec);
declare object airEst(tsm1nest);

```

```

declare object airFor(tsmfor);

*** Set up and run the TSM object ***;
rc = airModel.Initialize();
rc = airModel.SetY(Air);
rc = airModel.SetOption('lead',12);
rc = airModel.Replay(airSpec,airEst);
rc = airModel.Run();

*** Output the airline model forecasts ***;
rc = airFor.Collect(airModel);
endsubmit;
quit;

```

Output 19.3.2 shows the forecasts that are produced according to the fitted model.

Output 19.3.2 Replayed Forecasts (Partial Output)

Airline Model Predictions

DATE	PREDICT	STD	UPPER	LOWER
JAN1961	450.4	16.9215	484.5	418.2
FEB1961	426.1	18.8590	464.2	390.3
MAR1961	480.1	24.0408	528.9	434.7
APR1961	492.8	27.2405	548.3	441.6
MAY1961	509.5	30.5863	572.0	452.1
JUN1961	584.2	37.6514	661.4	513.9
JUL1961	670.7	45.9957	765.3	585.1
AUG1961	668.2	48.4237	768.0	578.3
SEP1961	559.6	42.6271	647.7	480.7
OCT1961	498.3	39.7181	580.6	425.0
NOV1961	431.2	35.8240	505.5	365.2
DEC1961	478.9	41.3484	565.0	403.0

Example 19.4: Performing Time Series Imputation Using an ARIMA Model

Time series imputation is the process of replacing missing values in a time series with reasonable values that reflect the existing pattern in the available data (such as trend, seasonal variations, or long-term cyclical variations). It is a popular technique that is used across various domains of science. This example shows how you can do the following:

- Fit an ARIMA model to a time series that contains missing values by using a TSM object.
- Store the imputed values of the time series in a table.

For the purposes of this example, the following statements introduce two artificial missing values into the airline series (see [Example 19.1](#) for more information about the airline series and the airline model). The modified series is stored in the mycas.airMiss CAS table. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```

data mycas.airmiss;
  set mycas.air;
  airmiss = air;
  if date = '01JUL1955'd then airmiss = .;
  if date = '01AUG1955'd then airmiss = .;
run;

```

The following statements plot the modified airline series along with the two artificial missing values that were introduced. The results are shown in [Output 19.4.1](#).

```

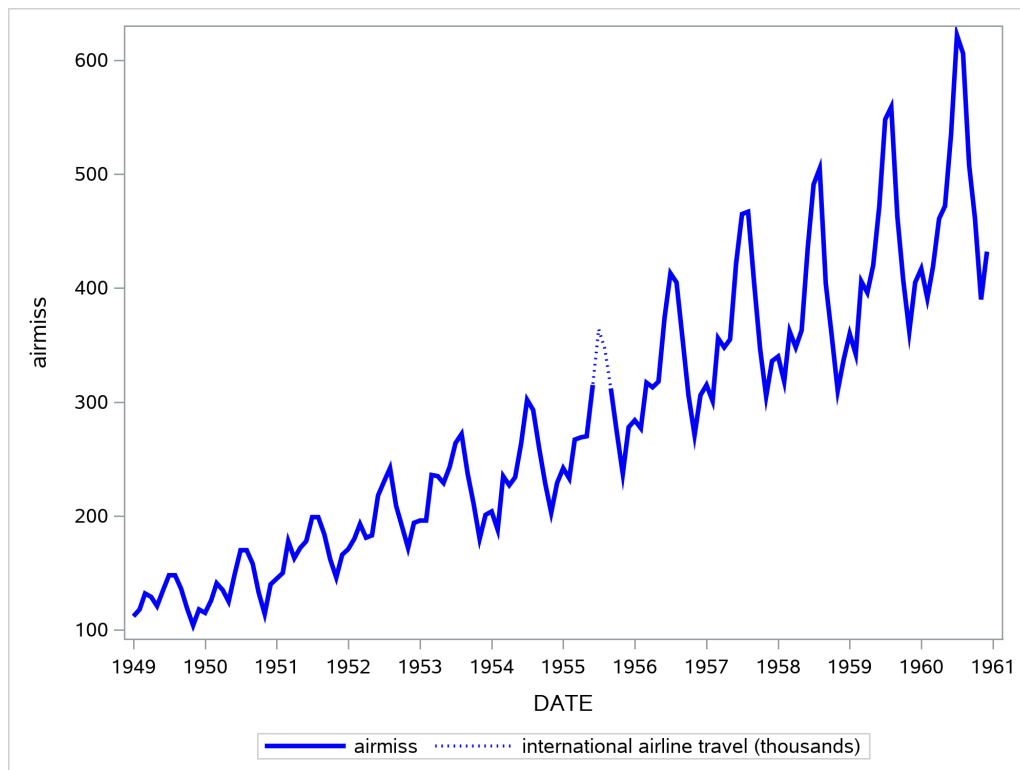
proc sort data=mycas.airmiss out=airmiss;
  by date;
run;

proc sgplot data=airmiss;
  series x=date y = airmiss / break lineattrs=(color=blue thickness=3);
  series x=date y = air / lineattrs=(thickness=2 pattern=dot color=blue);
run;

```

In [Output 19.4.1](#), the solid blue line represents all the nonmissing data values in the modified airline series. The dotted blue line between the years 1955 and 1956 represents the two actual values that were set to missing values.

Output 19.4.1 Airline Passenger Time Series with Artificial Missing Values



The following statements fit the airline model to the modified airline series and store the imputed time series in the mycas.airImpute CAS table:

```

proc tsmodel data=mycas.airmiss
  outobj=(airImpute=mycas.airImpute) ;
  id date interval=month;
  var airmiss;
  require tsm;
  submit;

  *** Analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);
  declare object airImpute(tsmfor('SMOOTH','YES'));

  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Set up the airline model spec: ***;
  ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint ***;
  rc = airSpec.Open();
  *** Specify differencing orders ***;
  diff[1] = 1;
  diff[2] = 12;
  rc = airSpec.SetDiff(diff,2);
  *** Specify moving average orders: q = (1)(12) ***;
  *** Use AddMAPoly twice for the two factors ***;
  ma[1] = 1;
  rc = airSpec.AddMAPoly(ma);
  ma[1] = 12;
  rc = airSpec.AddMAPoly(ma);
  *** Specify NOINT ***;
  rc = airSpec.SetOption('noint',1);
  *** Specify the log transform ***;
  rc = airSpec.SetTransform('log');
  *** Done setting up the ARIMA model ***;
  rc = airSpec.Close();

  *** Set up and run the TSM object ***;
  rc = airModel.Initialize(airSpec);
  rc = airModel.SetY(Airmiss);
  rc = airModel.Run();

  *** Output the imputed airline time series ***;
  rc = airImpute.Collect(airModel);
endsubmit;
quit;

```

The following statements plot the modified airline series along with its imputed version. The results are shown in [Output 19.4.2](#).

```

data mycas.airimpute;
  merge mycas.airimpute mycas.airmiss;
  by date;
run;

```

```

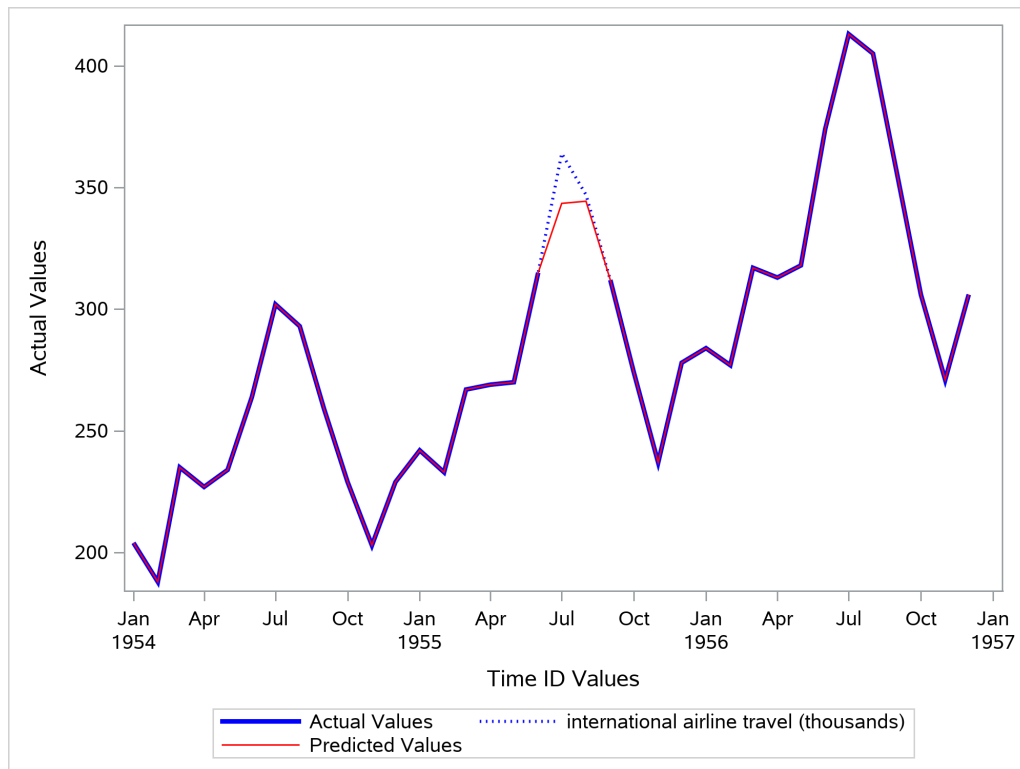
proc sort data=mycas.airimpute out=airimpute;
  by date;
run;

proc sgplot data=airimpute;
  series x=date y = actual / break lineattrs=(thickness=3 color=blue);
  series x=date y = air / lineattrs=(thickness=2 pattern=dot color=blue);
  series x=date y = predict / lineattrs=(thickness=1 color=red);
  where year(date) >= 1954 and year(date) <= 1956;
run;

```

In [Output 19.4.2](#), the thick blue line represents the modified airline series, and the thin red line represents its imputed version. The time axis range is restricted to the dates between the years 1954 and 1956 in order to facilitate the visualization. Notice how the thin red line (imputed values) closely mimics the dotted blue line (original actual values) over the region between the years 1955 and 1956 where the artificial missing values were introduced.

Output 19.4.2 Imputed Airline Passenger Time Series



Example 19.5: Combining Forecasts

This example shows how you can use the objects in the TSM package to combine multiple model forecasts to generate an improved forecast. In particular, it shows how you can do the following:

1. Fit the airline model, $\text{ARIMA}(0,1,1) \times (0,1,1)_{12}$ NOINT, to the airline series.
2. Fit the Winters exponential smoothing model to the airline series.
3. Use the CFC object to produce a combined forecast that is more accurate than the individual forecasts.
4. Output fit statistics for all models to a CAS table for later comparison.

```
proc tsmmodel data=mycas.air outobj=(fitstats=mycas.fitstats) lead=12;
  id date interval=month;
  var air;
  require tsm;
  submit;

  **Temporary work arrays used in ARIMA spec;
  array diff[2]/nosymbols;
  array ma[1]/nosymbols;

  *** Declare a collector object for fit statistics ***;
  declare object fitstats(tsmstat);

  *** Perform analysis based on airline model ***;
  declare object airModel(tsm);
  declare object airSpec(arimaspec);

  **Set up the airline model spec:**;
  ** Model: log(air) ~ (0,1,1)(0,1,1)12 noint *;
  rc = airSpec.Open( );
  *** Specify differencing orders ***;
  diff[1] = 1;
  diff[2] = 12;
  rc = airSpec.SetDiff(diff,2);
  *** Specify moving average orders: q = (1)(12) ***;
  *** Use AddMAPoly twice for the two factors ***;
  ma[1] = 1;
  rc = airSpec.AddMAPoly(ma);
  ma[1] = 12;
  rc = airSpec.AddMAPoly(ma);
  *** Specify NOINT ***;
  rc = airSpec.SetOption('noint',1);
  *** Specify the log transform ***;
  rc = airSpec.SetTransform('log');
  *** Done setting up the ARIMA model ***;
  rc = airSpec.Close( );

  *** Set up and run the TSM object ***;
  rc = airModel.Initialize(airSpec);
```

```

rc = airModel.SetY(Air);
rc = airModel.SetOption('lead', 3);
rc = airModel.SetOption('back', 3);
rc = airModel.Run( );

*** Output airline model statistics of fit ***;
rc = fitstats.Collect(airModel);

*** Perform analysis based on ESM model ***;
declare object esmModel(tsm);
declare object esmspec(esmspec);
rc = esmspec.Open( );
rc = esmspec.SetOption('method', 'winters');
rc = esmspec.Close( );

*** Set up and run the TSM object ***;
rc = esmModel.Initialize(esmspec);
rc = esmModel.SetY(Air);
rc = esmModel.SetOption('lead', 3);
rc = esmModel.SetOption('back', 3);
rc = esmModel.Run( );

*** Output ESM model statistics of fit ***;
rc = fitstats.Collect(esmModel);

declare object airCFC(cfc);
rc = airCFC.Initialize( );
rc = airCFC.SetY(Air);
rc = airCFC.AddModel(airModel);
rc = airCFC.AddModel(esmModel);
rc = airCFC.SetOption('weight', 'ols');
rc = airCFC.SetOption('lead', 3);
rc = airCFC.SetOption('back', 3);
rc = airCFC.Run( );

*** Output combined model statistics of fit ***;
rc = fitstats.Collect(airCFC);
endsubmit;
quit;

```

Output 19.5.1 shows the value of the root mean square error (RMSE) fit statistic for the airline, ESM, and forecast combination models over both the fit and forecast regions. Lower RMSE values indicate a more accurate model. Notice how the combined model forecast improves in accuracy over both of the independent forecasts in both the fit and forecast regions.

Output 19.5.1 Fit Statistics for the Airline, ESM, and Forecast Combination Models**Root Mean Square Error (RMSE) Statistic**

<u>_MODEL_</u>	<u>_REGION_</u>	<u>RMSE</u>
airModel	FIT	10.7047
airModel	FORECAST	8.7732
esmModel	FIT	10.5826
esmModel	FORECAST	10.6018
airCFC	FIT	10.3719
airCFC	FORECAST	8.6551

References

Box, G. E. P., and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Rev. ed. San Francisco: Holden-Day.

Chapter 20

Utility Package

Contents

Overview: UTL Package	1194
UTL Package Summary	1194
Using the UTL Package	1195
Common Argument Types	1195
Return Codes	1196
CLIMITS Object	1196
CLIMITS Synopsis	1197
CLIMITS Methods	1198
OUTNVP Object	1199
OUTNVP Synopsis	1200
OUTNVP Methods	1200
UTLSTAT Object	1201
UTLSTAT Synopsis	1204
UTLSTAT Methods	1205
Examples: UTL Package	1206
Using CAS Sessions and CAS Engine Librefs	1206
Example 20.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into CAS Tables	1207
Example 20.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc External Forecast	1210

Overview: UTL Package

This chapter describes the various utility object classes that are contained in the UTL package. The purpose of the UTL package is to provide a means for performing basic statistical computations on pairs of actual and predicted time series. The following types of computations are currently supported by the UTL package:

1. Computation of prediction standard errors and confidence limits for specified actual and predicted time series.
2. Computation and storing of model forecast fit statistics into CAS tables for specified actual and predicted time series.
3. Storing of ad hoc numeric variables that are defined in a user program into CAS tables.

The UTL package is object-oriented. To use the UTL package, you must declare instances of the object classes that are contained in the package. Declaring an object instance is the object-oriented equivalent of declaring a program variable. As with simple program variables, the declaration assigns the instance a name of your choosing and a type, which is defined by the object's class. Unlike simple program variables, the object instance requires a different syntax for interacting with it and offers different functions (methods) that are contextual to the object. The object can offer very sophisticated capabilities with a simple-to-use interface.

UTL Package Summary

Table 20.1 summarizes the single object class in the SFS package.

Table 20.1 Objects in the Utility Package

Object	Description
CLIMITS	Compute prediction standard errors and confidence limits for specified actual and predicted time series.
OUTNVP	Ad hoc name, value pair collector object for storing numeric scalar or array variables that are found in a user program into a CAS table.
UTLSTAT	Collector object for computing forecast fit statistics for specified actual and predicted time series and storing those statistics in a CAS table.

Using the UTL Package

The objects in the UTL package are subdivided into two different categories:

1. Stateful computational objects (the CLIMITS object)
2. Collector objects (the OUTNVP and UTLSTAT objects)

Collector objects provide a mechanism to create a snapshot of results (either from stateful objects or from plain program variables) and store those results into CAS tables. Each collector object defines a CAS table schema that is determined by the collector object's design. The collector objects in the UTL package (OUTNVP and UTLSTAT) follow a common method pattern. The basic execution follows this sequence of operations:

1. **Declare:** Create the collector object by using the object declaration statement.
2. **Collect:** Use the Collect method to store results into a CAS table. The input arguments of the Collect method are specific to the collector object. For example, the UTLSTAT collector object's Collect method requires an actual and predicted time series as arguments. It then uses the specified series to compute forecast fit statistics and stores the results in a CAS table. In contrast, the OUTNVP collector object's Collect method takes in an ad hoc numerical scalar or array variable from the user program and stores it into a CAS table. Rows that are collected are automatically appended to the collector's associated CAS table at the end of each BY group, and the collector object's saved row set is automatically reset. The Nrows attribute returns the current row count in the collector. A missing value is returned if nothing has been collected. The data, now stored in CAS tables, can then be used to produce reports or be used in further computations.

Common Argument Types

Table 20.2 defines the common argument types that are used in this chapter. The symbol *x* corresponds to the variable name.

Table 20.2 Common Argument Types

SAS Data Type	Declaration Syntax
String	LENGTH <i>x</i> \$ <i>n</i> ;
Numeric	<i>x</i> or LENGTH <i>x</i> 8;
Numeric array	ARRAY <i>x</i> [<i>n</i>]/NOSYMBOLS;
Status	<i>x</i> or LENGTH <i>x</i> 8;

Return Codes

Table 20.3 shows the return code (*rc* in method statements) status values that are used in this package. These status code values are returned after a method that is associated with an object is called; they can help determine whether the method executed successfully.

Table 20.3 Return Codes

Status	Meaning
< 0	An unrecoverable error occurred. No result was produced.
= 0	Unconditional success. The requested action completed and a normal result was produced.
> 0	Conditional success or warning. A result was produced subject to conditions.

Upon returning a negative status code, most methods in the SFS package objects also write a message to the output log that explains the causes of the related failure. These messages provide useful information during the process of debugging a user program. In the TSMODEL procedure, the output log is stored in the CAS table that is specified in the OUTLOG= option in the PROC TSMODEL statement. For more information about how to enable and configure logging and about how to access the output log after an invocation of the TSMODEL procedure, see Chapter 11, “The TSMODEL Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

CLIMITS Object

The CLIMITS object provides a mechanism for computing both the prediction standard errors and confidence limits of an *external model* (that is, a user-defined model) forecast, which is described by a pair of actual and predicted time series. The first stage of the computational process involves validating both input series. This validation is accomplished by ascertaining that the actual and predicted series have nonmissing observation values under at least one matching index. In addition, the predicted series is checked for the presence of extreme values. Next, the prediction standard errors are computed from the prediction errors (that is, the model residuals). Finally, the confidence limits are computed from the prediction standard errors. You can optionally supply the value of the confidence level that is used to compute the confidence limits. Note that the CLIMITS object retains all computed results in its internal memory. Individual forecast series can be queried via the [GetForecast](#) method, which is a method in the CLIMITS object.

Table 20.4 summarizes the methods that are associated with the UTL object.

Table 20.4 Methods of the CLIMITS Object

Method	Description
Compute	Compute the prediction standard errors and confidence limits for specified actual and predicted time series
GetForecast	Retrieve a computed forecast series by name

CLIMITS Synopsis

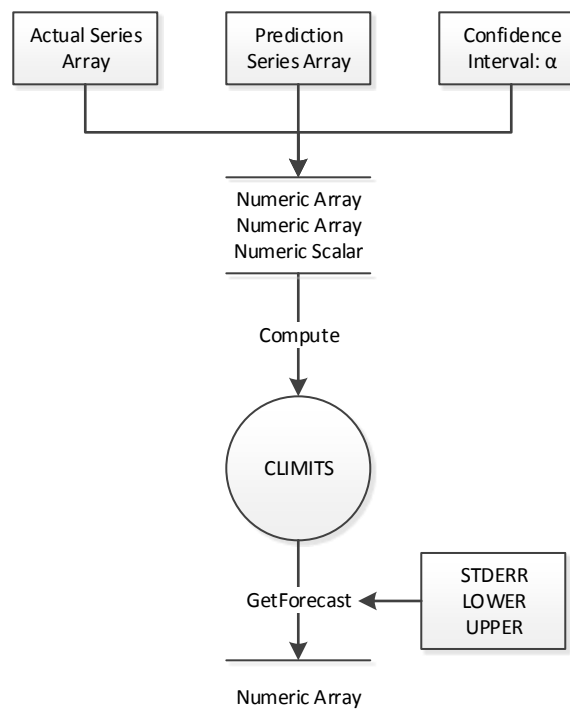
DECLARE OBJECT *obj* (CLIMITS) ;

Method syntax, in order of typical usage:

rc=obj.Compute (*Actual,Predicted,< Alpha>*) ;

rc=obj.GetForecast (*Which,Result*) ;

Figure 20.1 outlines the programmatic data flow through the CLIMITS object; each arrow represents a different object method.

Figure 20.1 CLIMITS Object Data Flow

CLIMITS Methods

CLIMITS.Compute Method

rc=obj.Compute (Actual, Predicted,< Alpha>) ;

Computes the prediction standard errors and confidence limits for specified actual and predicted time series. Both input series are validated by ascertaining the presence of nonmissing observation values under at least one matching index. Also, the predicted series is checked for the presence of extreme values. The computed forecast series are stored in the object's internal memory and can be individually queried via the [GetForecast](#) method into a numeric array defined in the user program. A negative return code indicates that the validation of an input series failed (for example, the predicted series has extreme values or all missing values), you specified an out-of-range *Alpha* argument value, or a computational failure occurred (for example, out-of-memory error).

Input Arguments

You must specify the following input arguments:

Actual specifies a numeric array that corresponds to the actual time series.
Predicted specifies a numeric array that corresponds to the predicted time series.

You can also specify the following input argument:

Alpha takes a numeric value between 0 and 1, exclusive, that specifies the significance level for forecast confidence bands. The default value is 0.05.

CLIMITS.GetForecast Method

rc=obj.GetForecast (Which,Result) ;

Places the specified forecast series (*Which*) from the CLIMITS object into the specified numeric array (*Result*). Forecast series have the same length as the predicted series that is supplied to the [Compute](#) method via its *Predicted* argument. The GetForecast method returns a negative status code if the [Compute](#) method returned a non zero value or if it was not yet executed (that is, no results exist to be queried).

Input Arguments

You must specify the following input argument:

Which is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:

LOWER	returns a lower confidence limit series.
STDERR	returns a prediction standard error series.
UPPER	returns an upper confidence limit series.

Output Arguments

You must specify the following output argument:

Result specifies a numeric array to receive the forecast series. If the array length is longer than the requested series, it is padded with missing values.

OUTNVP Object

The OUTNVP object collects any ad hoc numeric variables that are defined in the user program into CAS tables. The OUTNVP collector object accepts any of the following numeric types: scalar literal, scalar variable, and array variable.

Table 20.5 shows the contents of the OUTNVP object.

Table 20.5 Contents of the OUTNVP Object

Column	Type	Description
NAME	String	Name of the dependent variable
CALL	Numeric	Call count within the BY group
UTAG	Numeric	User-defined numeric tag
VIX	Numeric	Value index (1-based) for the row
VALUE	Numeric	Actual value for variable's row

Table 20.6 summarizes the methods that are associated with the OUTNVP object.

Table 20.6 Methods of the OUTNVP Object

Method	Description
<code>Collect</code>	Collect value for numeric data type
<code>nrows</code>	Get the OUTNVP instance row count

OUTNVP Synopsis

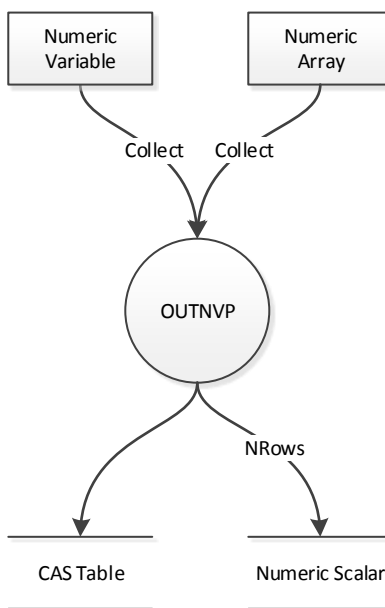
DECLARE OBJECT *obj* (OUTNVP) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect ( Variable,< Utag> ) ;  
nrows=obj.nrows () ;
```

Figure 20.2 outlines the programmatic data flow through the OUTNVP object.

Figure 20.2 OUTNVP Object Data Flow



OUTNVP Methods

OUTNVP.Collect Method

```
rc=obj.Collect ( Variable,< Utag> ) ;
```

Stores a numeric type, either scalar literal or variable or array variable in the OUTNVP table. When the *Variable* parameter is a numeric scalar literal or variable, this method collects a single row into the OUTNVP table. When the *Variable* parameter is a numeric array variable, this method collects a sequence of rows for the span of indices in the array. If the optional parameter *Utag* is specified, its value is included in the `_UTAG_` column of each collected OUTNVP row. The name of the collected variable is also included in the `_NAME_` column of each collected OUTNVP row. Similarly, the value of a counter that counts the number of calls to this method that are made within a BY group is also included in column `_CALL_` of each OUTNVP row. A negative return value indicates that an error occurred while storing results into a CAS table.

Input Arguments

You must specify the following input arguments:

Variable specifies a numeric array to be collected into the OUTNVP table.

You can also specify the following input argument:

Utag takes a numeric value that is included in the _UTAG_ column of the OUTNVP table. The default value is a missing value.

OUTNVP.nrows Method

```
nrows=obj.nrows () ;
```

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

UTLSTAT Object

The UTLSTAT object conveniently computes a number of forecast fit statistics for an ad hoc pair of user-specified actual and predicted time series. The computed forecast fit statistics are automatically stored in a CAS table. For each ad hoc pair of actual and predicted time series that is input into a UTLSTAT collector object, a single row of forecast fit statistics is added to the underlying CAS table. The CAS table schema that is used by the UTLSTAT object is compatible with the schema used by the HPFENGINE procedure for its OUTSTAT data set.

Table 20.7 shows the contents of the UTLSTAT object. For more information about the HPFENGINE procedure, see *SAS Forecast Server Procedures: User's Guide*.

Table 20.7 Contents of the UTLSTAT Object

Column	Type	Description
NAME	String	Variable name of actual series
MODEL	String	Variable name of predicted series
DFE	Numeric	Degrees of freedom error
N	Numeric	Number of observations
NOBS	Numeric	Number of observations used
NMISSA	Numeric	Number of missing actuals
NMISSP	Numeric	Number of missing predicted values
NPARMS	Numeric	Number of model parameters

Table 20.7 *continued*

Column	Type	Description
TSS	Numeric	Total sum of squares
SST	Numeric	Corrected total sum of squares
SSE	Numeric	Sum of square error
MSE	Numeric	Mean square error
RMSE	Numeric	Root mean square error
UMSE	Numeric	Unbiased mean square error
URMSE	Numeric	Unbiased root mean square error
MAPE	Numeric	Mean absolute percentage of error
MAE	Numeric	Mean absolute error
RSQUARE	Numeric	R-square
ADJRSQ	Numeric	Adjusted R-square
AADJRSQ	Numeric	Amemiya's adjusted R-square
RWRSQ	Numeric	Random walk R-square
AIC	Numeric	Akaike's information criterion
AICC	Numeric	Finite sample corrected Akaike's information criterion
SBC	Numeric	Schwarz Bayesian information criterion
APC	Numeric	Amemiya's prediction criterion
MAXERR	Numeric	Maximum error
MINERR	Numeric	Minimum error
MAXPE	Numeric	Maximum percentage of error
MINPE	Numeric	Minimum percentage of error
ME	Numeric	Mean error
MPE	Numeric	Mean percentage of error
MDAPE	Numeric	Median absolute percentage of error
GMAPE	Numeric	Geometric mean absolute percentage of error
MINPPE	Numeric	Minimum predicted percentage of error
MAXPPE	Numeric	Maximum predicted percentage of error
MPPE	Numeric	Mean predicted percentage of error
MAPPE	Numeric	Mean absolute predicted percentage of error

Table 20.7 *continued*

Column	Type	Description
MDAPPE	Numeric	Median absolute predicted percentage of error
GMAPPE	Numeric	Geometric mean absolute predicted percentage of error
MINSPE	Numeric	Minimum symmetric percentage of error
MAXSPE	Numeric	Maximum symmetric percentage of error
MSPE	Numeric	Mean symmetric percentage of error
SMAPE	Numeric	Mean absolute symmetric percentage of error
MDASPE	Numeric	Median absolute symmetric percentage of error
GMASPE	Numeric	Geometric mean absolute symmetric percentage of error
MINRE	Numeric	Minimum relative error
MAXRE	Numeric	Maximum relative error
MRE	Numeric	Mean relative error
MRAE	Numeric	Mean relative absolute error
MDRAE	Numeric	Median relative absolute error
GMRAE	Numeric	Geometric mean relative absolute error
MASE	Numeric	Mean absolute scaled error
MINAPES	Numeric	Minimum absolute error percentage of standard deviation
MAXAPES	Numeric	Maximum absolute error percentage of standard deviation
MAPES	Numeric	Mean absolute error percentage of standard deviation
MDAPES	Numeric	Median absolute error percentage of standard deviation
GMAPES	Numeric	Geometric mean absolute error percentage of standard deviation

Table 20.8 summarizes the methods that are associated with the UTLSTAT object.

Table 20.8 Methods of the UTLSTAT Object

Method	Description
Collect	Compute and collect forecast fit statistics for a specified ad hoc pair of actual and predicted time series.
nrows	Get the UTLSTAT instance row count

UTLSTAT Synopsis

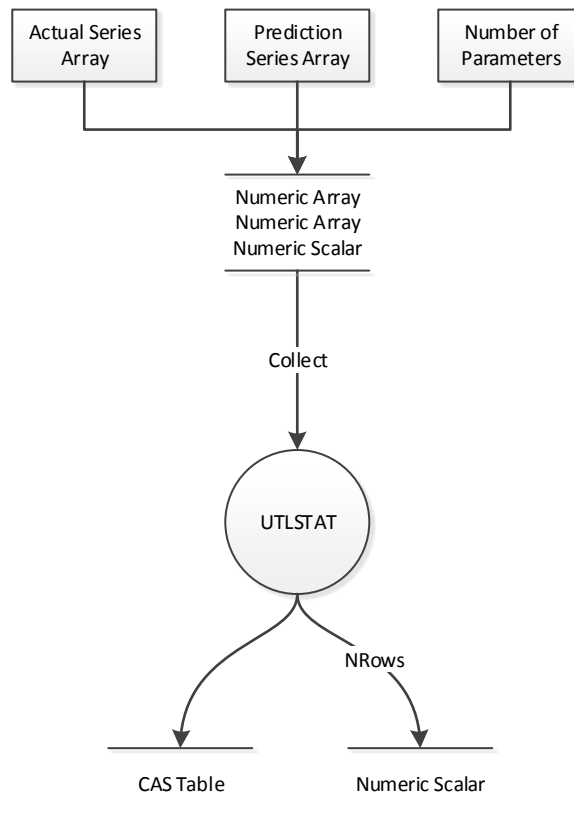
DECLARE OBJECT *obj* (UTLSTAT) ;

Method syntax, in order of typical usage:

```
rc=obj.Collect (Actual, Predicted, <Nparms>) ;  
nrows=obj.nrows () ;
```

Figure 20.3 outlines the programmatic data flow through the UTLSTAT object.

Figure 20.3 UTLSTAT Object Data Flow



UTLSTAT Methods

UTLSTAT.Collect Method

utlstatCollect *rc=obj.Collect* ;

(*Actual, Predicted, < Nparms >*)

Computes and collects forecast fit statistics for an ad hoc pair of specified actual time series and predicted time series. Each call collects a single row into the UTLSTAT CAS table, which contains all the forecast fit statistics that are listed in [Table 20.7](#). A negative return value indicates that an error occurred either during the computation of the forecast fit statistics or while storing results into a CAS table. The values of the forecast fit statistics computed by this method are sensitive to the value of optional parameter *Nparms*, which specifies the number of parameters that were used by the model that generated the predicted time series.

Input Arguments

You must specify the following input arguments:

Actual specifies a numeric array that corresponds to the actual time series.

Predicted specifies a numeric array that corresponds to the predicted time series.

You can also specify the following input argument:

Nparms takes a numeric value that specifies the number of parameters used by the model that generated the predicted series. The default value is 0.

UTLSTAT.nrows Method

nrows=obj.nrows () ;

Returns the number of rows that have been collected and stored in the CAS table.

Arguments

There are no arguments associated with this method.

Examples: UTL Package

Throughout this section, it is assumed that you have already started a CAS session and that the data tables that are used in this section are in `mycas`, a CAS library that you have necessary permissions to work with. This section assumes that you are familiar with the general workings of the `TSMODEL` procedure; for more information, see Chapter 11, “The `TSMODEL` Procedure” (*SAS Visual Forecasting: Forecasting Procedures*).

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);  
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;  
libname mycas cas sessref=mysess;
```

The `CAS` statement creates the CAS session named `mysess`, and the `LIBNAME` statement creates the `mycas` CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the `CASHOST` and `CASPORT` of the CAS server in the `CAS` statement, because these values are retrieved from the corresponding SAS option values.

If you have created the `mysess` session, you can terminate it by using the `TERMINATE` option in the `CAS` statement as follows:

```
cas mysess terminate;
```

For more information about the `CAS` statement and the `LIBNAME` statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Example 20.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into CAS Tables

This example demonstrates the capabilities of the UTLSTAT and OUTNVP collector objects from the UTL package. The UTLSTAT collector object is used to compute and collect forecast fit statistics for a specified pair of actual and predicted time series. In contrast, the OUTNVP collector object is used to store ad hoc numeric variables that are found in the user-defined program into CAS tables. The example starts by using a DATA step to load a sample time series data set called Sashelp.Air into a CAS table. The TSMODEL procedure is then invoked and the Time Series Model (TSM) object (available in the TSM package) is used to generate a model for the Air series in the Sashelp.Air data set. Next, both the actual Air series and its predicted counterpart (generated by the TSM object) are input into the UTLSTAT object to compute and collect numerous forecast fit statistics (see [Table 20.7](#)) into a CAS table. Because this example processes only a single time series (the Air series) and a single BY group, a single row of output is stored in the UTLSTAT table. The example then uses the OUTNVP collector object to store into a CAS table various forecast series that are retrieved from the TSM object. Each forecast series is first queried into a numeric array via the GetForecast method in the TSM object. Each array is then input directly, one by one, into the OUTNVP collector object. Finally, some results are retrieved from the resulting UTLSTAT and OUTNVP CAS tables, sorted, and printed for further inspection.

The following DATA step loads the Sashelp.Air data set onto the CAS server. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air (replace=yes);
  set Sashelp.Air;
run;
```

The following statements use the TSMODEL procedure to perform time series modeling on a single BY group. Because no ACCUMULATE= option is specified in the ID or VAR statements, its default value of TOTAL is used, which accumulates observations within a time period as a total sum of the nonmissing values.

```
proc tsmode1 data=mycas.air
  outarray = mycas.outarray (replace=yes)
  outscalar = mycas.outscalar (replace=yes)
  outobj=(
    utlstatobj = mycas.utlstat (replace = YES)
    outnvpmodelobj = mycas.outnvpmodel (replace = YES)
  )
  outlog = mycas.outlog (replace=yes)
  lead=12;
id date interval=month start='01jan1949'd end='01dec1960'd;
outarray predict error stderr lcl ucl;
outscalar rc1 rc2 rc3 rc4 rc5 rc6 rc7 rc8;
var air;
require tsm utl;
submit;
  /* Declare the "Time Series Model" (TSM) object and perform fit */
  declare object esm(tsm);
  rc1 = esm.Initialize();
  if rc1 < 0 then do; stop; end;
  rc1 = esm.SetY(air);
  if rc1 < 0 then do; stop; end;
```

```

rc1 = esm.Run();
if rc1 < 0 then do; stop; end;

/* Retrieve forecast series computed internally by the TSM object */
rc2 = esm.GetForecast('predict',predict); /*Predicted series*/
if rc2 < 0 then do; stop; end;
rc2 = esm.GetForecast('error',error);      /*Forecast error series*/
if rc2 < 0 then do; stop; end;
rc2 = esm.GetForecast('stderr',stderr);    /*Prediction std. errors series*/
if rc2 < 0 then do; stop; end;
rc2 = esm.GetForecast('lower',lcl);        /*Lower conf. limits series*/
if rc2 < 0 then do; stop; end;
rc2 = esm.GetForecast('upper',ucl);        /*Upper conf. limits series*/
if rc2 < 0 then do; stop; end;

/* Collect forecast series computed by the TSM object into a CAS table */
declare object outnvpmodelobj(outnvp);
rc3 = outnvpmodelobj.Collect(air,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(predict,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(error,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(stderr,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(lcl,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(ucl,_SERIES_);
if rc3 < 0 then do; stop; end;

/* Compute and collect a vast number of forecast fit statistics */
declare object utlstatobj(utlstat);
rc4 = utlstatobj.Collect(air, predict);
if rc4 < 0 then do; stop; end;
endsubmit;
run;

```

You can use the PRINT procedure to display a small subset of the 55 different forecast fit statistics that are collected by the UTLSTAT object. The PRINT procedure can access CAS tables directly; thus, there is no need to retrieve the UTLSTAT table back from CAS and into a local data set prior to display.

```

/* Print a few forecast fit statistics for the single BY group */
proc print data=mycas.utlstat;
  var _NAME_ _MODEL_ NOBS RMSE MAPE MAE RSQUARE AIC;
run;

```

Output 20.1.1 shows that a single row of data was collected. This row corresponds to the forecast fit statistics that were collected for a single forecast (that is, one pair of actual and predicted series) within the single BY group that was processed by the TSMODEL procedure call.

Output 20.1.1 Sample of the Forecast Fit Statistics Computed and Collected by the UTLSTAT Object

Obs	_NAME_	_MODEL_	NOBS	RMSE	MAPE	MAE	RSQUARE	AIC
1	AIR	predict	144	10.579085435	3.0845016398	8.0064787209	0.9921692375	679.35714621

You can use the PRINT procedure again to display a small subset of observations from the following six different series that were collected by the OUTNVP object into the mycas.outnvpmodel CAS table:

- The actual series (that is, the Air series).
- The predicted series that was generated by the TSM object.
- The forecast error series that was generated by the TSM object.
- The prediction standard errors series that was generated by the TSM object.
- The lower confidence limits series that was generated by the TSM object.
- The upper confidence limits series that was generated by the TSM object.

You can print in sequence the values of the first three observations in each of these six series, for a total of 18 rows. To accomplish this, you must sort the mycas.outnvpmodel CAS table in a manner that sequentially aligns all rows that correspond to each unique observation index in all collected series (that is, all rows that correspond to the first observation in all six series, followed by all rows that correspond to the second observation in all six series, and so on). You can use the SORT procedure to simultaneously sort a CAS table and retrieve the results into a local data set as follows:

```
/* Sort OUTNVP table by "row index" and "BY group Collect() call count" */
/* (that is, _VIX_ and _CALL_ columns). Transfer the sorted table rows */
/* back from CAS and into a local data set. */
proc sort data=mycas.outnvpmodel out=outnvpmodel;
    by _VIX_ _CALL_;
run;

/* Print the values of the first 18 rows, which correspond to the values */
/* of the first 3 observations in the six collected series. These 18 rows */
/* correspond to the condition "1 <= _VIX_ <= 3" in the CAS table called */
/* "MYCAS.OUTNVPMODEL". */
proc print data=outnvpmodel(obs=18);
run;
```

Output 20.1.2 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

Obs	_VAR_	_CALL_	_UTAG_	_VIX_	_VALUE_
1	AIR	1	0	1	112
2	predict	2	0	1	111.44700541
3	error	3	0	1	0.5529945856
4	stderr	4	0	1	10.691036547
5	lcl	5	0	1	90.492958825
6	ucl	6	0	1	132.401052
7	AIR	1	0	2	118
8	predict	2	0	2	119.94581467
9	error	3	0	2	-1.945814668
10	stderr	4	0	2	10.691036547
11	lcl	5	0	2	98.991768079
12	ucl	6	0	2	140.89986126
13	AIR	1	0	3	132
14	predict	2	0	3	135.17351431
15	error	3	0	3	-3.173514311
16	stderr	4	0	3	10.691036547
17	lcl	5	0	3	114.21946772
18	ucl	6	0	3	156.1275609

The sequence that is displayed in [Output 20.1.2](#) was obtained by using the SORT procedure to sort the data in the OUTNVP table in increasing order of the columns _VIX_ (series row index) and _CALL_ (that is, the OUTNVP object's Collect method call count within BY group). For example, rows 1–6 contain the values of the first observation in each of the six collected series, which correspond to the rows in the mycas.outnvpmodel CAS table where column _VIX_ = 1: row 1 is the value of the first observation in the Air series (that is, column _VAR_ = 'AIR') and rows 2–6 are the values of the first observations in the Predict, Error, StdErr, LCL, and UCL forecast series that were retrieved from the TSM object via its GetForecast method. The same applies for the second observation in each of the six collected series as described by rows 7–12 (that is, rows where column _VIX_ = 2) and for the third observation in each of the six collected series as described by rows 13–18 (rows where column _VIX_ = 3).

Example 20.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc External Forecast

This example uses the TSMODEL procedure to compute the prediction standard errors and confidence limits of an ad hoc forecast that includes an actual and predicted time series. The example starts by using a DATA step to create a synthetic data set called ExternalModel. The synthetic data set contains three time series that make up an ad hoc *external model* (that is, a user-defined model) forecast: an actual series called Air, a simulated forecast error series called Error, and a simulated predicted series called Predict. This synthetic, ad hoc external forecast lacks the prediction standard errors and confidence limits, which will be computed by the CLIMITS object. Notice from the DATA step code that the actual time series is simply a copy of the Air series taken from the SasHELP.Air data set. The predicted time series is generated by simply adding a small amount of noise to the actual Air series. The added noise corresponds to random samples taken of the uniform distribution (which ranges from 0.0 to 1.0) multiplied by a factor of 10. Thus, the added noise samples range from 0.0 to 10.0 and have an expected value of 5.0. This expected value is important because

it corresponds to the average forecast error of the simulated predicted series, a value that over many samples should approximate the prediction standard errors that will be computed by the CLIMITS object for this synthetic, ad hoc external forecast. Next, the synthetic data set is uploaded to a CAS table and the TSMODEL procedure is invoked. The actual and predicted time series are input to the CLIMITS object's [Compute](#) method to compute the prediction standard errors and confidence limits of the ad hoc external forecast. The CLIMITS object's [GetForecast](#) method is then used to retrieve the resulting three forecast series into numeric arrays that are defined in the user program. Finally, the OUTNVP collector object collects a total of six series into a CAS table: the actual Air series, the simulated predicted and forecast error series, and the three forecast series that were computed by the CLIMITS object. This was similarly done in [Example 20.1](#) to store into a CAS table the actual series plus the five forecast series that were retrieved from the TSM object via its own GetForecast method. Finally, some results are retrieved from CAS tables, sorted, and printed for further inspection:

The following DATA step creates a synthetic data set that contains actual and simulated predicted time series:

```
data ExternalModel (replace=yes);
    set Sashelp.Air;                /* The actual series: Sashelp.Air */
    error = floor(10*ranuni(246));  /* Simulated forecast error series */
    predict = air + error;          /* Simulated predicted series */
run;
```

The following DATA step loads the ExternalModel data set onto the CAS server. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.ExternalModel;
    set ExternalModel;
run;
```

The following statements use the TSMODEL procedure to compute the prediction standard errors and confidence limits of the ad hoc external forecast that resides in the synthetic data set:

```
proc tsmodel data = mycas.externalmodel
    outarray = mycas.outarray (replace=yes)
    outscalar = mycas.outscalar (replace=yes)
    outlog = mycas.outlog (replace=yes)
    outobj=( outnvpmodelobj = mycas.outnvpmodel (replace = YES) );
id date interval=month start='01jan1949'd end='01dec1960'd;
var air predict error;
outarrays stderr lcl ucl;
outscalar rc1 rc2 rc3;
require utl;
submit;
    /* Compute the prediction standard errors and confidence limits */
    declare object clim(CLIMITS);
    rc1 = clim.Compute(air,predict,0.05); /* Confidence level: 0.05 */
    if rc1 < 0 then do; stop; end;

    /* Retrieve the forecast series stored internally in the CLIMITS object */
    rc2 = clim.GetForecast('stderr',stderr); /* Prediction sdt. errors series */
    if rc2 < 0 then do; stop; end;
    rc2 = clim.GetForecast('lower',lcl);      /* Lower conf. limits series */
    if rc2 < 0 then do; stop; end;
```

```

rc2 = clim.GetForecast('upper',ucl);      /* Upper conf. limits series      */
if rc2 < 0 then do; stop; end;

/* Collect the actual, predicted, and forecast error series, in addition */
/* to the forecast series computed by the CLIMITS object into a CAS table*/
declare object outnvpmodelobj(outnvp);
rc3 = outnvpmodelobj.Collect(air,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(predict,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(error,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(stderr,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(lcl,_SERIES_);
if rc3 < 0 then do; stop; end;
rc3 = outnvpmodelobj.Collect(ucl,_SERIES_);
if rc3 < 0 then do; stop; end;
endsubmit;
run;

```

Following what was done in [Example 20.1](#), you can use the PRINT procedure to display a small subset of observations from the following six different series that were collected by the OUTNVP object into the mycas.outnvpmodel CAS table:

- The actual series (Air series) of the ad hoc external forecast.
- The simulated predicted series of the ad hoc external forecast.
- The simulated forecast error series of the ad hoc external forecast.
- The prediction standard errors series that were computed by the CLIMITS object for the ad hoc external forecast.
- The lower confidence limits series that were computed by the CLIMITS object for the ad hoc external forecast.
- The upper confidence limits series that were computed by the CLIMITS object for the ad hoc external forecast.

You can print in sequence the values of the first three observations in each of these six series, for a total of 18 rows. To accomplish this, you must sort the mycas.outnvpmodel CAS table in a manner that sequentially aligns all rows that correspond to each unique observation index in all collected series (all rows that correspond to the first observation in all six series, followed by all rows that correspond to the second observation in all six series, and so on). You can use the SORT procedure to simultaneously sort a CAS table and retrieve the results into a local data set as follows:

```

/* Sort OUTNVP table by "row index" and "BY group Collect() call count" */
/* (that is, _VIX_ and _CALL_ columns). Transfer the sorted table rows */
/* back from CAS and into a local data set. */
proc sort data=mycas.outnvpmodel out=outnvpmodel;
    by _VIX_ _CALL_;
run;

```

```

/* Print the value of the first 18 rows, which corresponds to the values */
/* of the first 3 observations in the six collected series. These 18 rows */
/* correspond to the condition "1 <= _VIX_ <= 3" in the CAS table called */
/* "MYCAS.OUTNVPMODEL". */
proc print data=outnvpmodel(obs=18);
run;

```

Output 20.2.1 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

Obs	_VAR_	_CALL_	_UTAG_	_VIX_	_VALUE_
1	AIR	1	0	1	112
2	predict	2	0	1	117
3	error	3	0	1	5
4	stderr	4	0	1	5.0408057116
5	lcl	5	0	1	107.12020235
6	ucl	6	0	1	126.87979765
7	AIR	1	0	2	118
8	predict	2	0	2	124
9	error	3	0	2	6
10	stderr	4	0	2	5.0408057116
11	lcl	5	0	2	114.12020235
12	ucl	6	0	2	133.87979765
13	AIR	1	0	3	132
14	predict	2	0	3	135
15	error	3	0	3	3
16	stderr	4	0	3	5.0408057116
17	lcl	5	0	3	125.12020235
18	ucl	6	0	3	144.87979765

The sequence displayed in [Output 20.2.1](#) was obtained by using the SORT procedure to sort the data in the OUTNVP table in increasing order of the columns `_VIX_` (that is, the series row index) and `_CALL_` (that is, the OUTNVP object's Collect method call count within BY group). For example, rows 1–6 contain the values of the first observation in each of the six collected series, which correspond to the rows in the mycas.outnvpmodel CAS table where column `_VIX_` = 1: row 1 is the value of the first observation in the actual Air series (that is, column `_VAR_` = 'AIR'), rows 2–3 are the values of the first observations in the simulated Predict and Error series of the ad hoc external forecast, and rows 4–6 are the values of the first observations in the StdErr, LCL, and UCL forecast series that were computed by the CLIMITS object for the ad hoc external forecast and retrieved via its [GetForecast](#) method. The same applies for the second observation in each of the six collected series as described by rows 7–12 (that is, rows where column `_VIX_` = 2) and for the third observation in each of the six collected series as described by rows 13–18 (rows where column `_VIX_` = 3).

Notice also in [Output 20.2.1](#) the value of 5.0408057116, which is reported for the first three observations of the prediction standard errors series, as shown by rows where column `_VAR_` = 'stderr'. This reported value should be close to the expected value of the simulated forecast error of the synthetic external forecast (5.0).

Subject Index

- Amemiya-MaCurdy estimation, 382
 - CPANEL procedure, 382
- an Example to illustrate the LABELSWITCH= option
 - HMM procedure, 694
- analytic store technology
 - HMM procedure, 706
- ANCOVA effects
 - Shared Concepts, 52
- ANOVA effects
 - Shared Concepts, 52
- AR tests
 - CPANEL procedure, 409
- ARIMA model
 - Box-Jenkins model, 72
- at sign (@) operator
 - Shared Concepts, 50, 54
- B-spline
 - spline basis (Shared Concepts), 32
- B-spline basis
 - CARIMA procedure, 32
 - CCDM procedure, 32
 - CCOPULA procedure, 32
 - CESM procedure, 32
 - CNTSELECT procedure, 32
 - CPANEL procedure, 32
 - CQLIM procedure, 32
 - CSPATIALREG procedure, 32
 - HMM procedure, 32
 - SEVSELECT procedure, 32
- backward elimination
 - Shared Concepts, 61
- bar (|) operator
 - Shared Concepts, 50, 53, 54
- between-groups regression, 372
 - CPANEL procedure, 372
 - instrumental variables, 385
- bounds on parameter estimates, 447
- BOUNDS statement, 447
- Box-Jenkins model, *see* ARIMA model
- BY groups
 - CARIMA procedure, 77
 - CCDM procedure, 128
 - CESM procedure, 255
 - CNTSELECT procedure, 298
 - CSPATIALREG procedure, 492
 - SEVSELECT procedure, 842
 - TSINFO procedure, 989
 - TSMODEL procedure, 1009
- CAL, 237
- calibration estimation, 237
- canonical maximum likelihood estimation, 236
- CARIMA procedure
 - B-spline basis, 32
 - BY groups, 77
 - computational method, 65
 - convergence criterion, 42–44, 46
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 76
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS table names, 35, 88
 - optimization techniques, 45, 65
 - parameter-based convergence criteria, 43, 46
 - splines and spline bases, 30
 - truncated power function (TPF) basis, 31
- CASLIB= option
 - naming a CAS library where the table containing Moody's Analytics data is stored, 786
 - SASEMOOD engine, 786
- CASOUT= option
 - naming a CAS table containing Moody's Analytics data, 786
 - SASEMOOD engine, 786
- CCDM procedure
 - B-spline basis, 32
 - BY groups, 128
 - computational method, 65
 - convergence criterion, 42–44, 46
 - descriptive statistics, 166
 - Fixed Parameter-Value Continuous Distributions, 163
 - Fixed Parameter-Value Discrete Distributions, 165
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 119
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS graph names, 173
 - ODS table names, 35, 172
 - optimization techniques, 45, 65
 - parameter perturbation analysis, 161
 - parameter-based convergence criteria, 43, 46

- scenario analysis, 138
- simulating aggregate adjusted loss distribution, 151
- simulating aggregate loss distribution, 139, 147
- splines and spline bases, 30
- truncated power function (TPF) basis, 31
- CCOPULA procedure
 - B-spline basis, 32
 - computational method, 65
 - convergence criterion, 42–44, 46
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 213
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS graph names, 241
 - ODS table names, 35
 - optimization techniques, 45, 65
 - output table names, 240
 - overview, 208
 - parameter-based convergence criteria, 43, 46
 - splines and spline bases, 30
 - syntax, 212
 - truncated power function (TPF) basis, 31
- censored regression models
 - CQLIM procedure, 461
- censoring and truncation
 - SEVSELECT procedure, 873
- CESM procedure
 - B-spline basis, 32
 - BY groups, 255
 - computational method, 65
 - convergence criterion, 42–44, 46
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 253
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS table names, 35
 - optimization techniques, 45, 65
 - parameter-based convergence criteria, 43, 46
 - splines and spline bases, 30
 - truncated power function (TPF) basis, 31
- class level
 - CPANEL procedure, 407
- CLASS statement
 - Shared Concepts, 46
 - syntax (Shared Concepts), 14, 46
- classification variables
 - Shared Concepts, 46
- Clayton copula, 231
- CMLE, 236
- CNTSELECT procedure
 - B-spline basis, 32
 - bounds on parameter estimates, 297
 - BY groups, 298
 - computational method, 65
 - convergence criterion, 42–44, 46
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 295
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS table names, 35
 - optimization techniques, 45, 65
 - output table names, 333
 - parameter-based convergence criteria, 43, 46
 - restrictions on parameter estimates, 306
 - splines and spline bases, 30
 - syntax, 292
 - truncated power function (TPF) basis, 31
- colon (:) operator
 - Shared Concepts, 51
- comparison tables
 - CPANEL procedure, 350, 410
- compound distribution modeling
 - CCDM procedure, 94
- computational method
 - CARIMA procedure, 65
 - CCDM procedure, 65
 - CCOPULA procedure, 65
 - CESM procedure, 65
 - CNTSELECT procedure, 65
 - CPANEL procedure, 65
 - CQLIM procedure, 65
 - CSPATIALREG procedure, 65
 - HMM procedure, 65
 - SEVSELECT procedure, 65
- conjugate-gradient optimization method, 296
- continuous-by-class effects
 - Shared Concepts, 55
- continuous-nesting-class effects
 - Shared Concepts, 54
- CONV= option
 - SASEMOOD engine, 786
- convergence criterion
 - CARIMA procedure, 42–44, 46
 - CCDM procedure, 42–44, 46
 - CCOPULA procedure, 42–44, 46
 - CESM procedure, 42–44, 46
 - CNTSELECT procedure, 42–44, 46
 - CPANEL procedure, 42–44, 46
 - CQLIM procedure, 42–44, 46
 - CSPATIALREG procedure, 42–44, 46
 - HMM procedure, 42–44, 46
 - SEVSELECT procedure, 42–44, 46
 - Shared Concepts, 42–44, 46

- converting the native frequency to a desired frequency
 - for Moody's Analytics data, *see* CONV=
 - option
- copulas
 - Clayton, 231
 - Frank, 231
 - Gumbel, 231
 - normal, 227
 - Student's *t*, 228
- correlated variables
 - CPANEL procedure, 352
- correlation matrix
 - CPANEL procedure, 410
- covariance matrix
 - CPANEL procedure, 409
- covariates
 - heteroscedasticity models, 452
- CPANEL procedure, 340
 - Amemiya-MaCurdy estimation, 382
 - AR tests, 409
 - B-spline basis, 32
 - between-groups regression, 372
 - class level, 407
 - comparison tables, 350, 410
 - computational method, 65
 - convergence criterion, 42–44, 46
 - correlated variables, 352
 - correlation matrix, 410
 - covariance matrix, 409
 - displayed output, 407
 - dynamic panel estimation, 389
 - endogenous variables, 354
 - fit statistics, 408
 - fixed-effects test, 408
 - Fuller-Battese variance components, 376, 380
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - HAC option, 404
 - Hausman test, 399, 408
 - Hausman-Taylor estimation, 381
 - HCCME= option, 401
 - ID variables, 355
 - input data tables, 348
 - instrumental variables, 356, 383
 - instruments table, 409
 - linear hypothesis testing, 400
 - model comparisons, 350
 - multithreading, 65
 - natural cubic spline basis, 34
 - Nerlove variance components, 377, 381
 - ODS table names, 35, 410
 - one-way fixed-effects by first differencing, 373
 - one-way fixed-effects model, 372
 - one-way random-effects model, 374
 - optimization techniques, 45, 65
 - output data tables, 366
 - parameter estimates, 409
 - parameter-based convergence criteria, 43, 46
 - partitioning data, 349
 - poolability test, 408
 - poolability test, fixed effects, 398
 - pooled regression, 372
 - predetermined variables, 367
 - predicted values, 366
 - printing options, 349
 - R-square measure, 406
 - residuals, 366
 - Sargan test, 409
 - selecting variance component methods, 377
 - splines and spline bases, 30
 - structural summary, 409
 - Swamy-Arora variance components, 375, 378
 - tests of linear hypotheses, 410
 - truncated power function (TPF) basis, 31
 - two-way fixed-effects model, 373
 - two-way random-effects model, 377
 - variance components, 408
 - Wallace-Hussain variance components, 376, 379
 - Wansbeek-Kapteyn variance components, 376, 380
- CQLIM procedure, 438
 - B-spline basis, 32
 - bivariate censored dependent variable modeling, 464
 - BY groups, 447
 - censored regression models, 461
 - computational method, 65
 - convergence criterion, 42–44, 46
 - frontier, 461
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - heteroscedasticity, 463
 - input data tables, 445
 - limited dependent variable models, 461
 - multithreading, 65
 - multivariate limited dependent models, 465
 - natural cubic spline basis, 34
 - ODS table names, 35
 - optimization techniques, 45, 65
 - ordinal discrete choice modeling, 460
 - output, 466
 - output data tables, 456
 - output table names, 471
 - parameter-based convergence criteria, 43, 46
 - splines and spline bases, 30
 - syntax, 443
 - tests on parameters, 465
 - truncated power function (TPF) basis, 31

- truncated regression models, 461
- creating a CAS table containing Moody's Analytics data, *see* GOCAS= option, *see* SASEMOOD engine
- creating a Moody's Analytics Data Buffet view, *see* SASEMOOD engine
- crossed effects
 - Shared Concepts, 53
- CSPATIALREG procedure
 - B-spline basis, 32
 - bounds on parameter estimates, 491
 - BY groups, 492
 - computational method, 65
 - convergence criterion, 42–44, 46
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 487
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS table names, 35
 - optimization techniques, 45, 65
 - output data tables, 496
 - output table names, 524
 - parameter-based convergence criteria, 43, 46
 - restrictions on parameter estimates, 497
 - splines and spline bases, 30
 - syntax, 485
 - truncated power function (TPF) basis, 31
- dash (-) operator
 - Shared Concepts, 51
- data
 - fit, 226
- data processing and fitting, 226
- data table output
 - HMM procedure, 711
- DEBUG= option
 - SASEMOOD engine, 786
- defining a custom objective function
 - SEVSELECT procedure, 920
- defining a custom probability distribution
 - SEVSELECT procedure, 895
- dependence measures, 225
- descriptive statistics
 - CCDM procedure, 166
- DIAG function
 - creating a diagonal matrix from a vector, 692
 - extracting the diagonal elements of a matrix, 692
- DISPLAY statement
 - syntax (Shared Concepts), 18
- displayed output
 - CPANEL procedure, 407
- DISPLAYOUT statement
 - syntax (Shared Concepts), 19
- double dash (--) operator
 - Shared Concepts, 51
- double-dogleg optimization method, 296
- dummy parameterization
 - Shared Concepts, 52
- dynamic panel estimation, 389
 - CPANEL procedure, 389
- ECM procedure
 - input data tables, 559
 - ODS table names, 576
- economic capital modeling
 - ECM procedure, 550
- effect parameterization
 - Shared Concepts, 56
- EFFECT statement
 - collection effect (Shared Concepts), 22
 - multimember effect (Shared Concepts), 22
 - polynomial effect (Shared Concepts), 24
 - spline effect (Shared Concepts), 27
 - syntax (Shared Concepts), 20
- empirical distribution function
 - SEVSELECT procedure, 883
- END= option
 - SASEMOOD engine, 786
- endogenous variables
 - CPANEL procedure, 354
- exact maximum likelihood estimation, 237
- example of Gaussian hidden Markov model
 - HMM procedure, 600
- example of Gaussian hidden Markov model for cross-sectional time series data
 - HMM procedure, 612
- example of Gaussian mixture hidden Markov model
 - HMM procedure, 616
- example of Gaussian mixture hidden Markov model for cross-sectional time series data
 - HMM procedure, 616
- example of Gaussian mixture model and Gaussian mixture regressions
 - HMM procedure, 717
- example of regime-switching autoregression model
 - HMM procedure, 642
- example of regime-switching regression model
 - HMM procedure, 636
- example of regression and vector autoregression
 - HMM procedure, 722
- examples, CARIMA
 - multimember effect, 23
- examples, CCDM
 - multimember effect, 23
- examples, CCOPULA
 - multimember effect, 23
- examples, CESM

- multimember effect, 23
- examples, CNTSELECT
 - multimember effect, 23
- examples, CPANEL
 - multimember effect, 23
- examples, CQLIM
 - multimember effect, 23
- examples, CSPATIALREG, 524
 - multimember effect, 23
- examples, HMM
 - multimember effect, 23
- examples, SEVSELECT
 - multimember effect, 23
- fit statistics
 - CPANEL procedure, 408
- fitting custom probability distributions
 - SEVSELECT procedure, 845
- Fixed Parameter-Value Continuous Distributions
 - CCDM procedure, 163
- Fixed Parameter-Value Discrete Distributions
 - CCDM procedure, 165
- fixed-effects model
 - poolability test, 398
- fixed-effects test
 - CPANEL procedure, 408
- forward selection
 - Shared Concepts, 59
- forward swap selection
 - Shared Concepts, 63
- Frank copula, 231
- FREQ= option
 - SASEMOOD engine, 787
- Fuller-Battese variance components
 - CPANEL procedure, 376, 380
- function-based convergence criteria
 - CARIMA procedure, 42–44
 - CCDM procedure, 42–44
 - CCOPULA procedure, 42–44
 - CESM procedure, 42–44
 - CNTSELECT procedure, 42–44
 - CPANEL procedure, 42–44
 - CQLIM procedure, 42–44
 - CSPATIALREG procedure, 42–44
 - HMM procedure, 42–44
 - SEVSELECT procedure, 42–44
 - Shared Concepts, 42–44
- Gaussian hidden Markov model
 - HMM procedure, 671
- Gaussian mixture hidden Markov model
 - HMM procedure, 672
- general effects
 - Shared Concepts, 55
- generalized method of moments (GMM), 389
- generalized two-stage least squares
 - instrumental variables, 388
- GLM parameterization
 - Shared Concepts, 52
- GOCAS= option
 - creating a CAS table containing Moody's
 - Analytics data, 789
 - SASEMOOD engine, 789
- gradient-based convergence criteria
 - CARIMA procedure, 42, 44
 - CCDM procedure, 42, 44
 - CCOPULA procedure, 42, 44
 - CESM procedure, 42, 44
 - CNTSELECT procedure, 42, 44
 - CPANEL procedure, 42, 44
 - CQLIM procedure, 42, 44
 - CSPATIALREG procedure, 42, 44
 - HMM procedure, 42, 44
 - SEVSELECT procedure, 42, 44
 - Shared Concepts, 42, 44
- Gumbel copula, 231
- HAC option
 - CPANEL procedure, 404
- Hausman specification test, 399
 - random-effects model, 399
- Hausman test
 - CPANEL procedure, 399, 408
- Hausman-Taylor estimation, 381
 - CPANEL procedure, 381
- HCCME= option
 - CPANEL procedure, 401
- heteroscedasticity models
 - covariates, 452
- heteroscedasticity- and autocorrelation-consistent
 - covariance matrices, 404
- hidden Markov model
 - HMM procedure, 668
- HMM procedure
 - an Example to illustrate the LABELSWITCH=
 - option, 694
 - analytic store technology, 706
 - B-spline basis, 32
 - computational method, 65
 - convergence criterion, 42–44, 46
 - data table output, 711
 - example of Gaussian hidden Markov model, 600
 - example of Gaussian hidden Markov model for
 - cross-sectional time series data, 612
 - example of Gaussian mixture hidden Markov
 - model, 616

- example of Gaussian mixture hidden Markov model for cross-sectional time series data, 616
- example of Gaussian mixture model and Gaussian mixture regressions, 717
- example of regime-switching autoregression model, 642
- example of regime-switching regression model, 636
- example of regression and vector autoregression, 722
- function-based convergence criteria, 42–44
- Gaussian hidden Markov model, 671
- Gaussian mixture hidden Markov model, 672
- gradient-based convergence criteria, 42, 44
- hidden Markov model, 668
- information criteria, 710
- input data tables, 650
- label switching problem, 694
- matrix expression, 683
- missing values, 710
- multithreading, 65
- natural cubic spline basis, 34
- ODS table names, 35, 715
- optimization algorithms, 683
- optimization techniques, 45, 65
- parameter estimation methods, 678
- parameter-based convergence criteria, 43, 46
- regime-switching autoregression model, 674
- regime-switching mean-adjusted autoregression model, 676
- regime-switching regression model, 672
- several types of models as special cases of hidden Markov models, 717
- splines and spline bases, 30
- truncated power function (TPF) basis, 31
- I function
 - creating an identity matrix, 693
- ID variables
 - CPANEL procedure, 355
- inferring time intervals and alignments
 - TSINFO procedure, 990
- information criteria
 - HMM procedure, 710
- instrumental variables
 - between-groups regression, 385
 - CPANEL procedure, 356, 383
 - generalized two-stage least squares, 388
 - one-way fixed effects, 385
 - one-way random effects, 387, 388
 - one-way time fixed effects, 386
 - pooled regression, 385
 - regression, 383
 - two-way fixed effects, 387
 - two-way random effects, 388
- instruments table
 - CPANEL procedure, 409
- interaction effects
 - Shared Concepts, 53
- intercept
 - Shared Concepts, 52
- J function
 - creating a matrix of identical values, 693
- JSON format
 - SASEMOOD engine, 793
- Kaplan-Meier's EDF estimator
 - SEVSELECT procedure, 885
- label switching problem
 - HMM procedure, 694
- Lagrange multiplier test
 - nonlinear hypotheses, 466
- levelization
 - Shared Concepts, 46
- LIBNAME interface engine for Moody's Analytics data, *see* SASEMOOD engine
- LIBNAME libref SASEMOOD statement
 - SASEMOOD engine, 785
- LIBNAME statement
 - SASEMOOD engine, 782
- limited dependent variable models
 - CQLIM procedure, 461
- linear hypothesis testing, 400
 - CPANEL procedure, 400
- LOGSELECT procedure
 - output data tables, 305
- loss distribution modeling
 - SEVSELECT procedure, 808
- main effects
 - Shared Concepts, 53
- matrix expression
 - HMM procedure, 683
- measure
 - dependence, 225
- missing values
 - HMM procedure, 710
- MLE, 237
- MNEMONIC= option
 - SASEMOOD engine, 789
- model comparisons
 - CPANEL procedure, 350
- modified Kaplan-Meier's EDF estimator
 - SEVSELECT procedure, 885
- Moody's Analytics data, *see* SASEMOOD engine
- Moody's Analytics data files, *see* SASEMOOD engine

- multithreading
 - CARIMA procedure, 65
 - CCDM procedure, 65
 - CCOPULA procedure, 65
 - CESM procedure, 65
 - CNTSELECT procedure, 65
 - CPANEL procedure, 65
 - CQLIM procedure, 65
 - CSPATIALREG procedure, 65
 - HMM procedure, 65
 - SEVSELECT procedure, 65
- naming a CAS library where the table containing Moody's Analytics data is stored, *see* CASLIB= option
- naming a CAS table containing Moody's Analytics data, *see* CASOUT= option
- natural cubic spline basis
 - CARIMA procedure, 34
 - CCDM procedure, 34
 - CCOPULA procedure, 34
 - CESM procedure, 34
 - CNTSELECT procedure, 34
 - CPANEL procedure, 34
 - CQLIM procedure, 34
 - CSPATIALREG procedure, 34
 - HMM procedure, 34
 - SEVSELECT procedure, 34
 - Shared Concepts, 34
- Nerlove variance components
 - CPANEL procedure, 377, 381
- nested effects
 - Shared Concepts, 54
- nested versus crossed effects
 - Shared Concepts, 54
- Newton-Raphson
 - optimization methods, 446
- Newton-Raphson method, 446
- Newton-Raphson optimization method, 296, 491
- Newton-Raphson Ridge optimization method, 296
- nonlinear hypotheses
 - Lagrange multiplier test, 466
- nonsingular parameterization
 - Shared Concepts, 56
- normal copula, 227
- ODS graph names
 - CCDM procedure, 173
 - CCOPULA procedure, 241
 - SEVSELECT procedure, 932
- ODS table names
 - CARIMA procedure, 35, 88
 - CCDM procedure, 35, 172
 - CCOPULA procedure, 35
 - CESM procedure, 35
 - CNTSELECT procedure, 35
 - CPANEL procedure, 35
 - CQLIM procedure, 35
 - CSPATIALREG procedure, 35
 - ECM procedure, 576
 - HMM procedure, 35, 715
 - SEVSELECT procedure, 35, 929
- one-way fixed effects
 - instrumental variables, 385
- one-way fixed-effects by first differencing, 373
 - CPANEL procedure, 373
- one-way fixed-effects model, 372
 - CPANEL procedure, 372
- one-way random effects
 - instrumental variables, 387, 388
- one-way random-effects model, 374
 - CPANEL procedure, 374
- one-way time fixed effects
 - instrumental variables, 386
- optimization algorithms
 - HMM procedure, 683
- optimization methods
 - conjugate-gradient, 296
 - double-dogleg, 296
 - Newton-Raphson, 296, 446
 - Newton-Raphson Ridge, 296
 - none, 296
 - quasi-Newton, 296
 - trust region, 296, 446
- optimization techniques
 - CARIMA procedure, 45, 65
 - CCDM procedure, 45, 65
 - CCOPULA procedure, 45, 65
 - CESM procedure, 45, 65
 - CNTSELECT procedure, 45, 65
 - CPANEL procedure, 45, 65
 - CQLIM procedure, 45, 65
 - CSPATIALREG procedure, 45, 65
 - HMM procedure, 45, 65
 - SEVSELECT procedure, 45, 65
 - Shared Concepts, 45
- options summary
 - CLASS statement, 14
 - EFFECT statement, 21, 301, 848
 - MODEL statement (CPANEL), 357
- ordering
 - of class levels (Shared Concepts), 47
- ordinal discrete choice modeling
 - CQLIM procedure, 460
- ordinal parameterization
 - Shared Concepts, 57
- ORTHEFFECT parameterization
 - Shared Concepts, 57

- ORTHORDINAL parameterization
 - Shared Concepts, 58
- ORTHOTHERM parameterization
 - Shared Concepts, 58
- ORTHPOLY parameterization
 - Shared Concepts, 58
- ORTHREF parameterization
 - Shared Concepts, 58
- OUT= option
 - SASEMOOD engine, 789
- OUT= option, SAS JSON data
 - SASEMOOD engine, 793
- output table names
 - CCOPULA procedure, 240
 - CNTSELECT procedure, 333
 - CQLIM procedure, 471
 - CSPATIALREG procedure, 524
- parameter estimates
 - CPANEL procedure, 409
- parameter estimation methods
 - HMM procedure, 678
- parameter perturbation analysis
 - CCDM procedure, 161
- parameter-based convergence criteria
 - CARIMA procedure, 43, 46
 - CCDM procedure, 43, 46
 - CCOPULA procedure, 43, 46
 - CESM procedure, 43, 46
 - CNTSELECT procedure, 43, 46
 - CPANEL procedure, 43, 46
 - CQLIM procedure, 43, 46
 - CSPATIALREG procedure, 43, 46
 - HMM procedure, 43, 46
 - SEVSELECT procedure, 43, 46
 - Shared Concepts, 43, 46
- parameterization
 - dummy (Shared Concepts), 52
 - effect (Shared Concepts), 56
 - GLM (Shared Concepts), 52
 - nonsingular (Shared Concepts), 56
 - ordinal (Shared Concepts), 57
 - ORTHEFFECT (Shared Concepts), 57
 - ORTHORDINAL (Shared Concepts), 58
 - ORTHOTHERM (Shared Concepts), 58
 - ORTHPOLY (Shared Concepts), 58
 - ORTHREF (Shared Concepts), 58
 - polynomial (Shared Concepts), 57
 - reference (Shared Concepts), 56
 - Shared Concepts, 49
 - thermometer (Shared Concepts), 57
- partitioning data
 - CPANEL procedure, 349
- PASS= option
 - SASEMOOD engine, 789
- polynomial effects
 - Shared Concepts, 53
- polynomial parameterization
 - Shared Concepts, 57
- poolability test, 398
 - CPANEL procedure, 408
 - fixed-effects model, 398
- poolability test, fixed effects
 - CPANEL procedure, 398
- pooled regression, 372
 - CPANEL procedure, 372
 - instrumental variables, 385
- predetermined variables
 - CPANEL procedure, 367
- predicted values
 - CPANEL procedure, 366
- printing options
 - CPANEL procedure, 349
- probability of observability
 - SEVSELECT procedure, 874
- quasi-Newton method, 446
- quasi-Newton optimization method, 296, 491
- R-square measure
 - CPANEL procedure, 406
- random-effects model
 - Hausman specification test, 399
- reference parameterization
 - Shared Concepts, 56
- regime-switching autoregression model
 - HMM procedure, 674
- regime-switching mean-adjusted autoregression model
 - HMM procedure, 676
- regime-switching regression model
 - HMM procedure, 672
- regression effects
 - Shared Concepts, 52, 53
- residuals
 - CPANEL procedure, 366
- Sargan test
 - CPANEL procedure, 409
- SAS JSON data, OUT= option
 - SASEMOOD engine, 793
- SAS JSON format
 - SASEMOOD engine, 793
- SAS output data set
 - SASEMOOD engine, 791
- SASEMOOD engine
 - CASLIB= option, 786
 - CASOUT= option, 786
 - CONV= option, 786

- creating a CAS table containing Moody's Analytics data, 804
- creating a Moody's Analytics Data Buffet view, 781
- DEBUG= option, 786
- END= option, 786
- FREQ= option, 787
- GOCAS= option, 789
- JSON format, 793
- LIBNAME interface engine for Moody's Analytics Data Buffet, 781
- LIBNAME libref SASEMOOD statement, 785
- LIBNAME statement, 782
- MNEMONIC= option, 789
- Moody's Analytics Data Buffet data, 781
- Moody's Analytics data files, 781
- OUT= option, 789
- OUT= option, SAS JSON data, 793
- PASS= option, 789
- SAS JSON data, OUT= option, 793
- SAS JSON format, 793
- SAS output data set, 791
- START= option, 790
- TRANS= option, 790
- USER= option, 790
- using PROC CASUTIL to save a CAS table containing Moody's Analytics data, 804
- viewing Moody's Analytics data, 781
- scale regression model
 - SEVSELECT procedure, 878
- scenario analysis
 - CCDM procedure, 138
- scoring functions
 - SEVSELECT procedure, 913
- selecting variance component methods
 - CPANEL procedure, 377
- SELECTION statement
 - syntax (Shared Concepts), 36
- several types of models as special cases of hidden Markov models
 - HMM procedure, 717
- SEVSELECT procedure
 - B-spline basis, 32
 - BY groups, 842
 - censoring and truncation, 873
 - computational method, 65
 - convergence criterion, 42–44, 46
 - defining a custom objective function, 920
 - defining a custom probability distribution, 895
 - empirical distribution function, 883
 - function-based convergence criteria, 42–44
 - gradient-based convergence criteria, 42, 44
 - input data tables, 831
 - multithreading, 65
 - natural cubic spline basis, 34
 - ODS graph names, 932
 - ODS table names, 35, 929
 - optimization techniques, 45, 65
 - output data tables, 852
 - parameter-based convergence criteria, 43, 46
 - predefined distributions, 863
 - predefined utility functions, 908
 - probability of observability, 874
 - scale regression model, 878
 - scoring functions, 913
 - splines and spline bases, 30
 - statistics of fit, 890
 - truncated power function (TPF) basis, 31
 - Tweedie distribution, 865
- SHAPE function
 - creating a matrix with different dimension, 693
- Shared Concepts
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - ANCOVA effects, 52
 - ANOVA effects, 52
 - at sign (@) operator, 50, 54
 - backward elimination, 61
 - bar (|) operator, 50, 53, 54
 - CLASS statement, 14, 46
 - classification variables, 46
 - collection effect (EFFECT statement), 22
 - colon (:) operator, 51
 - continuous-by-class effects, 55
 - continuous-nesting-class effects, 54
 - convergence criterion, 42–44, 46
 - crossed effects, 53
 - dash (-) operator, 51
 - DISPLAY statement, 18
 - DISPLAYOUT statement, 19
 - double dash (- -) operator, 51
 - dummy parameterization, 52
 - effect parameterization, 56
 - EFFECT statement, 20
 - FCONV option, 43
 - FCONV2 option, 43
 - forward selection, 59
 - forward swap selection, 63
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - function-based convergence criteria, 42–44

- GCONV option, 44
- GCONV2 option, 44
- general effects, 55
- GLM parameterization, 52
- gradient-based convergence criteria, 42, 44
- GTOL option, 44
- GTOL2 option, 44
- interaction effects, 53
- intercept, 52
- levelization, 46
- main effects, 53
- MAXFUNC= option, 45
- MAXITER= option, 45
- MAXTIME= option, 45
- MINITER= option, 45
- missing values, CLASS variables, 48
- multimember effect (EFFECT statement), 22
- nested effects, 54
- nested versus crossed effects, 54
- nonsingular parameterization, 56
- optimization techniques, 45
- ORDER= option, 47
- ordering of class levels, 47
- ordinal parameterization, 57
- ORTHEFFECT parameterization, 57
- ORTHORDINAL parameterization, 58
- ORTHOTHERM parameterization, 58
- ORTHPOLY parameterization, 58
- ORTHREF parameterization, 58
- parameter-based convergence criteria, 43, 46
- parameterization, 49
- polynomial effect (EFFECT statement), 24
- polynomial effects, 53
- polynomial parameterization, 57
- reference parameterization, 56
- regression effects, 52, 53
- SELECTION statement, 36
- singular parameterization, 53
- sort order of class levels, 47
- spline basis, B-spline, 32
- spline basis, natural cubic spline, 34
- spline basis, truncated power function, 31
- spline effect (EFFECT statement), 27
- stepwise selection, 62
- TECHNIQUE= option, 45
- thermometer parameterization, 57
- XCONV option, 46
- XSIZE option, 46
- XTOL option, 46
- simulating aggregate adjusted loss distribution
 - CCDM procedure, 151
- simulating aggregate loss distribution
 - CCDM procedure, 139, 147
- singular parameterization
 - Shared Concepts, 53
- Sklar's theorem, 224
- sort order
 - of class levels (Shared Concepts), 47
- splines and spline bases
 - CARIMA procedure, 30
 - CCDM procedure, 30
 - CCOPULA procedure, 30
 - CESM procedure, 30
 - CNTSELECT procedure, 30
 - CPANEL procedure, 30
 - CQLIM procedure, 30
 - CSPATIALREG procedure, 30
 - HMM procedure, 30
 - SEVSELECT procedure, 30
- START= option
 - SASEMOOD engine, 790
- statistics of fit
 - SEVSELECT procedure, 890
- stepwise selection
 - Shared Concepts, 62
- structural summary
 - CPANEL procedure, 409
- Student's *t* copula, 228
- Swamy-Arora variance components
 - CPANEL procedure, 375, 378
- tests of linear hypotheses
 - CPANEL procedure, 410
- thermometer parameterization
 - Shared Concepts, 57
- time series information, *see also* TSINFO procedure
- TPF basis, *see* truncated power function (TPF) basis
- TRANS= option
 - SASEMOOD engine, 790
- truncated power function (TPF) basis
 - CARIMA procedure, 31
 - CCDM procedure, 31
 - CCOPULA procedure, 31
 - CESM procedure, 31
 - CNTSELECT procedure, 31
 - CPANEL procedure, 31
 - CQLIM procedure, 31
 - CSPATIALREG procedure, 31
 - HMM procedure, 31
 - SEVSELECT procedure, 31
 - Shared Concepts, 31
- truncated regression models
 - CQLIM procedure, 461
- trust region
 - optimization methods, 446
- trust region method, 446
- trust region optimization method, 296, 491
- TSINFO procedure

- BY groups, 989
- TSMODEL procedure
 - BY groups, 1009
 - input data tables, 1004
 - time series analysis, 995
- Turnbull's EDF estimator
 - SEVSELECT procedure, 886
- Tweedie distribution
 - SEVSELECT procedure, 865
- two-stage least squares (2SLS)
 - regression, 381–383
- two-way fixed effects
 - instrumental variables, 387
- two-way fixed-effects model, 373
 - CPANEL procedure, 373
- two-way random effects
 - instrumental variables, 388
- two-way random-effects model, 377
 - CPANEL procedure, 377
- USER= option
 - SASEMOOD engine, 790
- using PROC CASUTIL to save a CAS table containing
 - Moody's Analytics data, *see* SASEMOOD engine
- variance components
 - CPANEL procedure, 408
- viewing Moody's Analytics data, *see* SASEMOOD engine
- Wallace-Hussain variance components
 - CPANEL procedure, 376, 379
- Wansbeek-Kapteyn variance components
 - CPANEL procedure, 376, 380

Syntax Index

ABSCONV option

- PROC CARIMA statement, 42
- PROC CCDM statement, 42
- PROC CCOPULA statement, 42
- PROC CESM statement, 42
- PROC CNTSELECT statement, 42
- PROC CPANEL statement, 42
- PROC CQLIM statement, 42
- PROC CSPATIALREG statement, 42
- PROC HMM statement, 42
- PROC SEVSELECT statement, 42

ABSFCONV option

- PROC CARIMA statement, 42
- PROC CCDM statement, 42
- PROC CCOPULA statement, 42
- PROC CESM statement, 42
- PROC CNTSELECT statement, 42
- PROC CPANEL statement, 42
- PROC CQLIM statement, 42
- PROC CSPATIALREG statement, 42
- PROC HMM statement, 42
- PROC SEVSELECT statement, 42

ABSFTOL option

- PROC CARIMA statement, 42
- PROC CCDM statement, 42
- PROC CCOPULA statement, 42
- PROC CESM statement, 42
- PROC CNTSELECT statement, 42
- PROC CPANEL statement, 42
- PROC CQLIM statement, 42
- PROC CSPATIALREG statement, 42
- PROC HMM statement, 42
- PROC SEVSELECT statement, 42

ABSGCONV option

- PROC CARIMA statement, 42
- PROC CCDM statement, 42
- PROC CCOPULA statement, 42
- PROC CESM statement, 42
- PROC CNTSELECT statement, 42
- PROC CPANEL statement, 42
- PROC CQLIM statement, 42
- PROC CSPATIALREG statement, 42
- PROC HMM statement, 42
- PROC SEVSELECT statement, 42

ABSGTOL option

- PROC CARIMA statement, 42
- PROC CCDM statement, 42
- PROC CCOPULA statement, 42

- PROC CESM statement, 42

- PROC CNTSELECT statement, 42

- PROC CPANEL statement, 42

- PROC CQLIM statement, 42

- PROC CSPATIALREG statement, 42

- PROC HMM statement, 42

- PROC SEVSELECT statement, 42

ABSTOL option

- PROC CARIMA statement, 42

- PROC CCDM statement, 42

- PROC CCOPULA statement, 42

- PROC CESM statement, 42

- PROC CNTSELECT statement, 42

- PROC CPANEL statement, 42

- PROC CQLIM statement, 42

- PROC CSPATIALREG statement, 42

- PROC HMM statement, 42

- PROC SEVSELECT statement, 42

ABSXCONV option

- PROC CARIMA statement, 43

- PROC CCDM statement, 43

- PROC CCOPULA statement, 43

- PROC CESM statement, 43

- PROC CNTSELECT statement, 43

- PROC CPANEL statement, 43

- PROC CQLIM statement, 43

- PROC CSPATIALREG statement, 43

- PROC HMM statement, 43

- PROC SEVSELECT statement, 43

ABSXTOL option

- PROC CARIMA statement, 43

- PROC CCDM statement, 43

- PROC CCOPULA statement, 43

- PROC CESM statement, 43

- PROC CNTSELECT statement, 43

- PROC CPANEL statement, 43

- PROC CQLIM statement, 43

- PROC CSPATIALREG statement, 43

- PROC HMM statement, 43

- PROC SEVSELECT statement, 43

ACCUMULATE function

- TSA package, 1079

ACCUMULATE2 Method

- TSA package, 1081

ACCUMULATE= option

- FORECAST statement (CESM), 255

- ID statement (CARIMA), 79

- ID statement (CESM), 257

- ID statement (TSMODEL), 1012
- VAR statement (TSMODEL), 1018
- ACF function
 - TSA package, 1083
- ADAPTIVEDECAY option
 - OPTIMIZATION statement, 662
- AddARPoly method
 - TSM Package, 1139
- AddBlockSeason method
 - TSM Package, 1156
- AddComponent method
 - TSM Package, 1157
- AddCycle method
 - TSM Package, 1158
- AddDeplag method
 - TSM Package, 1158
- AddExternal method
 - TSM Package, 1119
- AddInput method
 - TSM Package, 1159
- AddMAPoly method
 - TSM Package, 1140
- AddModel method
 - TSM Package, 1126
- AddPredict method
 - TSM Package, 1126
- AddSeason method
 - TSM Package, 1159
- AddTF method
 - TSM Package, 1140
- AddTFDenPoly method
 - TSM Package, 1141
- AddTFNumPoly method
 - TSM Package, 1141
- AddX method
 - TSM Package, 1120
- ADJSAMPLEVAR= option
 - OUTPUT statement (CCDM), 132
- ADJUSTEDSEVERITY= option
 - PROC CCDM statement, 118
- ALGORITHM= option
 - OPTIMIZE statement (HMM), 661
- ALIGN= option
 - ID statement (CARIMA), 80
 - ID statement (CESM), 258
 - ID statement (TSINFO), 989
 - ID statement (TSMODEL), 1013
- ALL option
 - TEST statement (CNTSELECT), 310
 - TEST statement (CPANEL), 368
 - TEST statement (CQLIM), 458
 - TEST statement (CSPATIALREG), 499
- ALPHA= number
 - FORECAST statement (CARIMA), 79
- ALPHA= option
 - FORECAST statement (CESM), 255
 - FORECAST statement (HMM), 656
- AMACURDY option
 - MODEL statement (CPANEL), 359
- ANNEALINGRATE=option
 - OPTIMIZATION statement, 662
- APPROXIMATION= option
 - PROC CSPATIALREG statement, 488
- ARIMASPEC object
 - TSM Package, 1137
- ARMAORDERS function
 - TSA package, 1085
- ARMEAN= option
 - MODEL statement (HMM), 658
- Autoreg method
 - TSM Package, 1156
- AUXDATA= option
 - PROC TSMODEL statement, 1004
- BACK= option
 - FORECAST statement (CESM), 255
 - FORECAST statement (HMM), 656
- BASIS option
 - EFFECT statement, spline (CARIMA), 27
 - EFFECT statement, spline (CCDM), 27
 - EFFECT statement, spline (CCOPULA), 27
 - EFFECT statement, spline (CESM), 27
 - EFFECT statement, spline (CNTSELECT), 27
 - EFFECT statement, spline (CPANEL), 27
 - EFFECT statement, spline (CQLIM), 27
 - EFFECT statement, spline (CSPATIALREG), 27
 - EFFECT statement, spline (HMM), 27
 - EFFECT statement, spline (SEVSELECT), 27
- BOUNDS statement
 - CNTSELECT procedure, 297
 - CQLIM procedure, 447
 - CSPATIALREG procedure, 491
- BTWNG option
 - MODEL statement (CPANEL), 359
- BUFFER=option
 - OPTIMIZATION statement, 663
- BY statement
 - CARIMA procedure, 77
 - CCDM procedure, 128
 - CCOPULA procedure, 214
 - CESM procedure, 255
 - CNTSELECT procedure, 298
 - CPANEL procedure, 349
 - CQLIM procedure, 447
 - CSPATIALREG procedure, 492
 - SEVSELECT procedure, 842
 - TSINFO procedure, 989
 - TSMODEL procedure, 1009

- CARIMA procedure, 74
 - syntax, 74
- CARIMA procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CARIMA procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CARIMA procedure, PROC CARIMA statement
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 76
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - TECHNIQUE= option, 45
 - XCONV option, 46
 - XSIZE option, 46
 - XTOL option, 46
- CARIMA procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- CASESENSITIVE option
 - DISPLAY statement (CCDM), 18, 130
 - DISPLAY statement (CCOPULA), 18, 216
 - DISPLAY statement (CNTSELECT), 18, 299
 - DISPLAY statement (CPANEL), 18, 353
 - DISPLAY statement (CQLIM), 18, 449
 - DISPLAY statement (CSPATIALREG), 18, 493
 - DISPLAY statement (ECM), 18, 565
 - DISPLAY statement (HMM), 18, 653
 - DISPLAY statement (SEVSELECT), 18, 843
- CASLIB= option
 - LIBNAME statement (SASEMOOD), 786
- CASOUT= option
 - LIBNAME statement (SASEMOOD), 786
- CCDM procedure, 115
 - COUNTMODEL statement, 128
 - EXTERNALCOUNTS statement, 131
 - OUTPUT statement, 132
 - OUTSUM statement, 133
 - SEVERITYMODEL statement, 136
 - SIMULATEDSYMBOL statement, 137
 - syntax, 115
- CCDM procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CCDM procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 130
 - EXCLUDE option, 18, 130
 - EXCLUDEALL option, 19, 130
 - TRACE option, 19, 130
- CCDM procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 131
 - NOREPLACE option, 19, 131
 - REPEATED option, 19, 131
- CCDM procedure, EFFECT statement

- BASIS option (spline), 27
- DATABOUNDARY option (spline), 28
- DEGREE option (polynomial), 25
- DEGREE option (spline), 28
- DETAILS option (multimember), 23
- DETAILS option (polynomial), 25
- DETAILS option (spline), 28
- KNOTMAX= option (spline), 28
- KNOTMETHOD option (spline), 28
- KNOTMIN= option (spline), 29
- MDEGREE option (polynomial), 25
- NATURALCUBIC option (spline), 29
- NOEFFECT option (multimember), 23
- NOSEPARATE option (polynomial), 25
- SEPARATE option (spline), 30
- SPLIT option (spline), 30
- STANDARDIZE option (polynomial), 25
- CCDM procedure, EXTERNALCOUNTS statement
 - COUNT= option, 131
 - ID= option, 131
- CCDM procedure, OUTPUT statement
 - ADJSAMPLEVAR= option, 132
 - OUT= option, 132
 - PERTURBOUT option, 133
 - SAMPLEVAR= option, 133
- CCDM procedure, OUTSUM statement
 - OUT= option, 133
 - PCTLNAME= option, 135
 - PCTLNDEC= option, 136
 - PCTLPTS= option, 135
- CCDM procedure, PROC CCDM statement, 118
 - ABSCONV option, 42
 - ABSFCONV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - ADJUSTEDSEVERITY= option, 118
 - COUNTSTORE= option, 118
 - DATA= option, 119
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - IGNORECOV option, 119
 - LEFTTRUNCATION= option, 119
 - MAXCOUNTDRAW= option, 119
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - NOPRINT option, 120
 - NPERTURBEDSAMPLES= option, 120
 - NREPLICATES= option, 120
 - ONEADJSEVCOMPAT option, 121
 - OUTDRAW= option, 121
 - PCTLDEF= option, 121
 - PERTURBMODE= option, 122
 - PLOTS= option, 122
 - PRINT= option, 125
 - RIGHTTRUNCATION= option, 125
 - SEED= option, 125
 - SEVERITYEST= option, 126
 - SEVERITYSTORE= option, 126
 - TECHNIQUE= option, 45
 - TRUNCATEZEROS option, 127
 - VARDEF= option, 128
 - XCONV option, 46
 - XSIZE option, 46
 - XTOL option, 46
- CCDM procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- CCDM procedure, SEVERITYMODEL statement
 - DEFINITIONSTABLE= option, 137
- CCF function
 - TSA package, 1086
- CCOPULA procedure, 212
 - BY statement, 214
 - DEFINE statement, 214
 - FIT statement, 218
 - PROC CCOPULA statement, 213
 - SIMULATE statement, 222
 - syntax, 212
 - VAR statement, 223
 - VIEWSTORE statement, 223
- CCOPULA procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15

- ORDER= option, 15
- PARAM= option, 15
- REF= option, 16
- SPLIT option, 17
- CCOPULA procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 216
 - EXCLUDE option, 18, 216
 - EXCLUDEALL option, 19, 216
 - TRACE option, 19, 217
- CCOPULA procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 217
 - NOREPLACE option, 19, 217
 - REPEATED option, 19, 217
- CCOPULA procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CCOPULA procedure, PROC CCOPULA statement
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 213
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - TECHNIQUE= option, 45
 - XCONV option, 46
 - XSIZE option, 46
 - XTOL option, 46
- CCOPULA procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- CENSORED option
 - ENDOGENOUS statement (CQLIM), 451, 454
- CESM procedure, 252
 - syntax, 252
- CESM procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CESM procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CESM procedure, PROC CESM statement
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42

- ABSTOL option, 42
- ABSXCONV option, 43
- ABSXTOL option, 43
- DATA= option, 253
- FCONV option, 43
- FCONV2 option, 43
- FSIZE option, 44
- FTOL option, 43
- FTOL2 option, 43
- GCONV option, 44
- GCONV2 option, 44
- GTOL option, 44
- GTOL2 option, 44
- MAXFUNC= option, 45
- MAXITER= option, 45
- MAXTIME= option, 45
- MINITER= option, 45
- TECHNIQUE= option, 45
- XCONV option, 46
- XSIZE option, 46
- XTOL option, 46
- CESM procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- CFC object
 - TSM Package, 1124
- CHAIN= option
 - OPTIMIZATION statement, 663
- CHOOSE= option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37, 309
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37, 861
- CLASS statement
 - CARIMA procedure, 14, 46
 - CCDM procedure, 14, 46
 - CCOPULA procedure, 14, 46
 - CESM procedure, 14, 46
 - CNTSELECT procedure, 14, 46, 298
 - CPANEL procedure, 14, 46
 - CQLIM procedure, 14, 46
 - CSPATIALREG procedure, 14, 46, 492
 - HMM procedure, 14, 46
 - SEVSELECT procedure, 14, 46, 842
- CLIMITS object, 1196
- Close method
 - TSM Package, 1134, 1142, 1150, 1160
- CNTSELECT procedure, 292
 - syntax, 292
 - VIEWSTORE statement, 311
- CNTSELECT procedure, CLASS statement, 298
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CNTSELECT procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 299
 - EXCLUDE option, 18, 299
 - EXCLUDEALL option, 19, 299
 - TRACE option, 19, 299
- CNTSELECT procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 300
 - NOREPLACE option, 19, 300
 - REPEATED option, 19, 300
- CNTSELECT procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CNTSELECT procedure, PROC CNTSELECT statement
 - ABSCONV option, 42
 - ABSFCONV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42

- ABSGTOL option, 42
- ABSTOL option, 42
- ABSXCONV option, 43
- ABSXTOL option, 43
- DATA= option, 295
- FCONV option, 43
- FCONV2 option, 43
- FSIZE option, 44
- FTOL option, 43
- FTOL2 option, 43
- GCONV option, 44
- GCONV2 option, 44
- GTOL option, 44
- GTOL2 option, 44
- MAXFUNC= option, 45
- MAXITER= option, 45
- MAXTIME= option, 45
- MINITER= option, 45
- TECHNIQUE= option, 45
- XCONV option, 46
- XSIZE option, 46
- XTOL option, 46
- CNTSELECT procedure, SELECTION statement, 308
 - CHOOSE= option, 37, 309
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36, 308
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37, 309
 - SELECTION= option, 41
 - STOP= option, 38, 309
 - STOPHORIZON= option, 41
- CNTSELECT procedure, TEST statement, 310
- CNTSELECT procedure, WEIGHT statement, 312
- Collect method, 1200, 1205
 - TSA package, 1045, 1053, 1060, 1061, 1068, 1069, 1076
- COMMONPACKAGE option
 - OUTSCORELIB statement (SEVSELECT), 857
- COMPARE statement
 - CPANEL procedure, 350
- COMPETITIVE option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37
- Compute method, 1198
- CONDITIONAL
 - OUTPUT statement (CQLIM), 456
- CONGRA option
 - METHOD= option, PROC CSPATIALREG statement, 491
- CONV= option
 - LIBNAME statement (SASEMOOD), 786
- CONVERGE= value
 - ESTIMATE statement (CARIMA), 77
- COPULASIM= option
 - PROC ECM statement, 559
- COPYVAR= option
 - OUTPUT statement (CNTSELECT), 305
 - OUTPUT statement (CPANEL), 366
 - OUTPUT statement (CQLIM), 456
 - OUTPUT statement (CSPATIALREG), 497
- COPYVARS= option
 - OUTPUT statement (SEVSELECT), 853
- CORR object
 - TSA package, 1041
- CORR package
 - GetResult method, 1042
 - Initialize method, 1043
 - OUTCORR object, 1045
 - Run method, 1044
 - SetNLags method, 1044
 - SetNParms method, 1044
 - SetY method, 1044
- CORRB option
 - CQLIM procedure, 445
 - MODEL statement, 304, 496
 - MODEL statement (CPANEL), 365
 - PROC CNTSELECT statement, 295
 - PROC CSPATIALREG statement, 489
- CORRELATED statement
 - CPANEL procedure, 352
- COST option
 - ENDOGENOUS statement (CQLIM), 451, 455
- COUNT= option
 - EXTERNALCOUNTS statement (CCDM), 131
- COUNTMODEL statement
 - CCDM procedure, 128
- COUNTSTORE= option
 - PROC CCDM statement, 118
- COVB option
 - CQLIM procedure, 445
 - MODEL statement, 304, 496
 - MODEL statement (CPANEL), 365
 - PROC CNTSELECT statement, 295
 - PROC CSPATIALREG statement, 489

- COVEST= option
 - CQLIM procedure, 446
 - PROC CNTSELECT statement, 296
 - PROC CSPATIALREG statement, 490
- COVOUT option
 - PROC SEVSELECT statement, 830
- CPANEL procedure, 346
 - details, 369
 - syntax, 346
- CPANEL procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CPANEL procedure, COMPARE statement, 350
 - MSTAT option, 350
 - OUTPARM= option, 351
 - OUTSTAT= option, 351
 - PSTAT option, 351
- CPANEL procedure, CORRELATED statement, 352
- CPANEL procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 353
 - EXCLUDE option, 18, 353
 - EXCLUDEALL option, 19, 353
 - TRACE option, 19, 353
- CPANEL procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 354
 - NOREPLACE option, 19, 354
 - REPEATED option, 19, 354
- CPANEL procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CPANEL procedure, ENDOGENOUS statement, 354
- CPANEL procedure, ID statement, 355
- CPANEL procedure, INSTRUMENTS statement, 356
- CPANEL procedure, MODEL statement, 357
 - AMACURDY option, 359
 - BTWNG option, 359
 - BTWNT option, 359
 - CORRB option, 365
 - COVB option, 365
 - DYNDIFF option, 359
 - DYNSYS option, 359
 - FDONE option, 359
 - FIXONE option, 360
 - FIXTWO option, 360
 - HTAYLOR option, 360
 - IVBTWNG option, 360
 - IVFIXONE option, 360
 - IVFIXTWO option, 360
 - IVG2SLS option, 360
 - IVPOOLED option, 360
 - IVRANONE option, 361
 - IVRANTWO option, 361
 - NOINT option, 364
 - NOLABEL option, 365
 - POOLED option, 361
 - PRINTFIXED option, 365
 - RANONE option, 361
 - RANTWO option, 361
 - VCOMP= option, 364
- CPANEL procedure, OUTPUT statement, 366
 - COPYVAR= option, 366
 - OUT= option, 366
 - P= option, 366
 - PREDICTED= option, 366
 - R= option, 366
 - RESIDUAL= option, 366
- CPANEL procedure, PREDETERMINED statement, 367
- CPANEL procedure, PROC CPANEL statement, 348
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 348
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - MAXFUNC= option, 45
 - MAXITER= option, 45

- MAXTIME= option, 45
- MINITER= option, 45
- NOPART option, 349
- NOPRINT option, 349
- PRINTALL option, 349
- TECHNIQUE= option, 45
- XCONV option, 46
- XSIZE option, 46
- XTOL option, 46
- CPANEL procedure, RESTRICT statement, 367
- CPANEL procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- CPANEL procedure, TEST statement, 368
 - ALL option, 368
 - LM option, 368
 - LR option, 368
 - WALD option, 368
- CQLIM procedure, 443
 - syntax, 443
- CQLIM procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CQLIM procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 449
 - EXCLUDE option, 18, 449
 - EXCLUDEALL option, 19, 449
 - TRACE option, 19, 449
- CQLIM procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 449
 - NOREPLACE option, 19, 450
 - REPEATED option, 19, 450
- CQLIM procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CQLIM procedure, FREQ statement, 452
- CQLIM procedure, OUTPUT statement
 - OUT= option, 456
- CQLIM procedure, PROC CQLIM statement
 - ABSCONV option, 42
 - ABSFCONV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 445
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - TECHNIQUE= option, 45
 - XCONV option, 46
 - XSIZE option, 46
 - XTOL option, 46
- CQLIM procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37

- SELECTION= option, 41
- STOP= option, 38
- STOPHORIZON= option, 41
- CQLIM procedure, TEST statement, 458
- CQLIM procedure, WEIGHT statement, 459
- Criterion method
 - TSM Package, 1121, 1127
- CRITERION= option
 - FORECAST statement (CESM), 255
 - PROC SEVSELECT statement, 830
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37
- CROSSCORR object
 - TSA package, 1048
- CROSSCORR package
 - GetResult method, 1051
 - Initialize method, 1051
 - OUTCROSSCORR object, 1052
 - Run method, 1051
 - SetNLags method, 1051
 - SetX method, 1052
 - SetY method, 1052
- CSPATIALREG procedure, 485
 - syntax, 485
- CSPATIALREG procedure, CLASS statement, 492
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- CSPATIALREG procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 493
 - EXCLUDE option, 18, 493
 - EXCLUDEALL option, 19, 493
 - TRACE option, 19, 494
- CSPATIALREG procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 494
 - NOREPLACE option, 19, 494
 - REPEATED option, 19, 494
- CSPATIALREG procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- CSPATIALREG procedure, OUTPUT statement
 - OUT= option, 496
- CSPATIALREG procedure, PROC CSPATIALREG statement
 - ABSCONV option, 42
 - ABSFCONV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 487
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - TECHNIQUE= option, 45
 - XCONV option, 46
 - XSIZE option, 46
 - XTOL option, 46
- CSPATIALREG procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37

SELECTION= option, 41
 STOP= option, 38
 STOPHORIZON= option, 41
 CSPATIALREG procedure, TEST statement, 499
 DATA= option
 MARGINAL statement (ECM), 566
 PROC CARIMA statement, 76
 PROC CCDM statement, 119
 PROC CCOPULA statement, 213
 PROC CESM statement, 253
 PROC CNTSELECT statement, 295
 PROC CPANEL statement, 348
 PROC CQLIM statement, 445
 PROC CSPATIALREG statement, 487
 PROC ECM statement, 559
 PROC HMM statement, 650
 PROC SEVSELECT statement, 831
 PROC TSINFO statement, 988
 PROC TSMODEL statement, 1004
 DATABOUNDARY option
 EFFECT statement, spline (CARIMA), 28
 EFFECT statement, spline (CCDM), 28
 EFFECT statement, spline (CCOPULA), 28
 EFFECT statement, spline (CESM), 28
 EFFECT statement, spline (CNTSELECT), 28
 EFFECT statement, spline (CPANEL), 28
 EFFECT statement, spline (CQLIM), 28
 EFFECT statement, spline (CSPATIALREG), 28
 EFFECT statement, spline (HMM), 28
 EFFECT statement, spline (SEVSELECT), 28
 DBLDOG option
 METHOD= option, PROC CSPATIALREG statement, 491
 DEBUG= option
 LIBNAME statement (SASEMOOD), 786
 DECODE statement, 652
 DECOMP object
 TSA package, 1055
 DECOMP package
 GetResult method, 1057
 Initialize method, 1057
 OUTDECOMP object, 1059
 OUTDECOMP2 object, 1061
 Run method, 1057
 SetLambda method, 1057
 SetMode method, 1058
 SetResults method, 1058
 SetSeasonality method, 1058
 SetY method, 1059
 DEFINE statement
 CCOPULA procedure, 214
 DEFINITIONSTABLE= option
 SEVERITYMODEL statement (CCDM), 137

DEGREE option
 EFFECT statement, polynomial (CARIMA), 25
 EFFECT statement, polynomial (CCDM), 25
 EFFECT statement, polynomial (CCOPULA), 25
 EFFECT statement, polynomial (CESM), 25
 EFFECT statement, polynomial (CNTSELECT), 25
 EFFECT statement, polynomial (CPANEL), 25
 EFFECT statement, polynomial (CQLIM), 25
 EFFECT statement, polynomial (CSPATIALREG), 25
 EFFECT statement, polynomial (HMM), 25
 EFFECT statement, polynomial (SEVSELECT), 25
 EFFECT statement, spline (CARIMA), 28
 EFFECT statement, spline (CCDM), 28
 EFFECT statement, spline (CCOPULA), 28
 EFFECT statement, spline (CESM), 28
 EFFECT statement, spline (CNTSELECT), 28
 EFFECT statement, spline (CPANEL), 28
 EFFECT statement, spline (CQLIM), 28
 EFFECT statement, spline (CSPATIALREG), 28
 EFFECT statement, spline (HMM), 28
 EFFECT statement, spline (SEVSELECT), 28
 DELTA= value
 ESTIMATE statement (CARIMA), 77
 DESCENDING option
 CLASS statement (CARIMA), 15
 CLASS statement (CCDM), 15
 CLASS statement (CCOPULA), 15
 CLASS statement (CESM), 15
 CLASS statement (CNTSELECT), 15
 CLASS statement (CPANEL), 15
 CLASS statement (CQLIM), 15
 CLASS statement (CSPATIALREG), 15
 CLASS statement (HMM), 15
 CLASS statement (SEVSELECT), 15
 DETAILS option
 EFFECT statement, multimember (CARIMA), 23
 EFFECT statement, multimember (CCDM), 23
 EFFECT statement, multimember (CCOPULA), 23
 EFFECT statement, multimember (CESM), 23
 EFFECT statement, multimember (CNTSELECT), 23
 EFFECT statement, multimember (CPANEL), 23
 EFFECT statement, multimember (CQLIM), 23
 EFFECT statement, multimember (CSPATIALREG), 23
 EFFECT statement, multimember (HMM), 23
 EFFECT statement, multimember (SEVSELECT), 23
 EFFECT statement, polynomial (CARIMA), 25
 EFFECT statement, polynomial (CCDM), 25

- EFFECT statement, polynomial (CCOPULA), 25
- EFFECT statement, polynomial (CESM), 25
- EFFECT statement, polynomial (CNTSELECT), 25
- EFFECT statement, polynomial (CPANEL), 25
- EFFECT statement, polynomial (CQLIM), 25
- EFFECT statement, polynomial (CSPATIALREG), 25
- EFFECT statement, polynomial (HMM), 25
- EFFECT statement, polynomial (SEVSELECT), 25
- EFFECT statement, spline (CARIMA), 28
- EFFECT statement, spline (CCDM), 28
- EFFECT statement, spline (CCOPULA), 28
- EFFECT statement, spline (CESM), 28
- EFFECT statement, spline (CNTSELECT), 28
- EFFECT statement, spline (CPANEL), 28
- EFFECT statement, spline (CQLIM), 28
- EFFECT statement, spline (CSPATIALREG), 28
- EFFECT statement, spline (HMM), 28
- EFFECT statement, spline (SEVSELECT), 28
- DETAILS= option
 - SELECTION statement (CARIMA), 38, 39
 - SELECTION statement (CCDM), 38, 39
 - SELECTION statement (CCOPULA), 38, 39
 - SELECTION statement (CESM), 38, 39
 - SELECTION statement (CNTSELECT), 38, 39
 - SELECTION statement (CPANEL), 38, 39
 - SELECTION statement (CQLIM), 38, 39
 - SELECTION statement (CSPATIALREG), 38, 39
 - SELECTION statement (HMM), 38, 39
 - SELECTION statement (SEVSELECT), 38, 39
- DFMIXTURE= option
 - SCALEMODEL statement (SEVSELECT), 858
- DIAG function, 692
- DIFF= value-list
 - ESTIMATE statement (CARIMA), 77
- DISCRETE option
 - ENDOGENOUS statement (CQLIM), 450, 454
- DISPERSION= option
 - OUTPUT statement (CNTSELECT), 306
- DISPLAY statement
 - CCDM procedure, 18, 129
 - CCOPULA procedure, 18, 216
 - CNTSELECT procedure, 18, 298
 - CPANEL procedure, 18, 352
 - CQLIM procedure, 18, 448
 - CSPATIALREG procedure, 18, 493
 - ECM procedure, 18, 564
 - HMM procedure, 18, 653
 - SEVSELECT procedure, 18, 843
- DISPLAYOUT statement
 - CCDM procedure, 19, 130
 - CCOPULA procedure, 19, 217
 - CNTSELECT procedure, 19, 299
 - CPANEL procedure, 19, 353
 - CQLIM procedure, 19, 449
 - CSPATIALREG procedure, 19, 494
 - ECM procedure, 19, 565
 - HMM procedure, 19, 654
 - SEVSELECT procedure, 19, 844
- DISPMODEL statement
 - CNTSELECT procedure, 300
- DIST statement
 - SEVSELECT procedure, 845
- DIST= option
 - CNTSELECT statement (CNTSELECT), 303
 - MODEL statement (CNTSELECT), 303
- DISTRIBUTE= option
 - ID statement (TSMODEL), 1013
- DISTRIBUTION= option
 - ENDOGENOUS statement (CQLIM), 450, 454
- DPF object
 - TSA package, 1064
- DPF package
 - Initialize method, 1066
 - OUTDPE object, 1067
 - OUTDPROB object, 1069
 - Run method, 1066
 - SetOption method, 1066
- DRAWID= option
 - MARGINAL statement (ECM), 566
 - PROC ECM statement, 560
- DYNDIFF option
 - MODEL statement (CPANEL), 359
- DYNSYS option
 - MODEL statement (CPANEL), 359
- ECM procedure, 557
 - MARGINAL statement, 566
 - OUTPUT statement, 567
 - OUTSUM statement, 568
 - syntax, 557
 - VAR statement, 570
- ECM procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 565
 - EXCLUDE option, 18, 565
 - EXCLUDEALL option, 19, 565
 - TRACE option, 19, 565
- ECM procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 565
 - NOREPLACE option, 19, 565
 - REPEATED option, 19, 566
- ECM procedure, MARGINAL statement
 - DATA= option, 566
 - DRAWID= option, 566
 - PCTLDEF= option, 567
 - SAMPLEVAR= option, 566

- ECM procedure, OUTPUT statement
 - MARGINALSOUT option, 567
 - OUT= option, 567
 - TOTALLOSSVAR= option, 568
- ECM procedure, OUTSUM statement
 - OUT= option, 568
 - PCTLNDEC= option, 568
 - PCTLPRE= option, 568
 - PCTLPTS= option, 569
 - TVARPRE= option, 569
 - TVARPTS= option, 569
- ECM procedure, PROC ECM statement, 559
 - COPULASIM= option, 559
 - DATA= option, 559
 - DRAWID= option, 560
 - EDFACCURACY= option, 560
 - EXACTFINALCOUNT option, 560
 - MAXITER= option, 560
 - NOPRINT option, 560
 - NOSHUFFLE option, 561
 - PCTLDEF= option, 561
 - PRINT= option, 561
 - SAMPLEFRACTION= option, 563
 - SEED= option, 563
 - TAILEDFAccURACY= option, 563
 - TAILSTARTEDF= option, 563
 - TOLERANCE= option, 563
 - VARDEF= option, 564
- EDFACCURACY= option
 - PROC ECM statement, 560
- EDFALPHA= option
 - PROC SEVSELECT statement, 831
- EFFECT statement
 - CARIMA procedure, 20
 - CCDM procedure, 20
 - CCOPULA procedure, 20
 - CESM procedure, 20
 - CNTSELECT procedure, 20, 301
 - collection effect, 22
 - CPANEL procedure, 20
 - CQLIM procedure, 20
 - CSPATIALREG procedure, 20
 - HMM procedure, 20
 - multimember effect, 22
 - polynomial effect, 24
 - SEVSELECT procedure, 20, 847
 - spline effect, 27
- EMPIRICALCDF= option
 - PROC SEVSELECT statement, 838
- END= option
 - ID statement (CARIMA), 81
 - ID statement (CESM), 258
 - ID statement (TSINFO), 990
 - ID statement (TSMODEL), 1014
- LIBNAME statement (SASEMOOD), 786
- ENDOGENOUS statement
 - CPANEL procedure, 354
- ENDSUBMIT statement
 - TSMODEL procedure, 1020
- ERRORCOMP= option
 - CNTSELECT statement (CNTSELECT), 304
 - MODEL statement (CNTSELECT), 304
- ERRSTD
 - OUTPUT statement (CQLIM), 456
- ESMSPEC object
 - TSM Package, 1132
- ESTIMATE statement
 - CARIMA procedure, 77
 - HMM procedure, 655
- ESTISPV option
 - MODEL statement (HMM), 659
- EVALUATE statement, 655
- EXACTFINALCOUNT option
 - PROC ECM statement, 560
- EXCLUDE option
 - DISPLAY statement (CCDM), 18, 130
 - DISPLAY statement (CCOPULA), 18, 216
 - DISPLAY statement (CNTSELECT), 18, 299
 - DISPLAY statement (CPANEL), 18, 353
 - DISPLAY statement (CQLIM), 18, 449
 - DISPLAY statement (CSPATIALREG), 18, 493
 - DISPLAY statement (ECM), 18, 565
 - DISPLAY statement (HMM), 18, 653
 - DISPLAY statement (SEVSELECT), 18, 843
- EXCLUDEALL option
 - DISPLAY statement (CCDM), 19, 130
 - DISPLAY statement (CCOPULA), 19, 216
 - DISPLAY statement (CNTSELECT), 19, 299
 - DISPLAY statement (CPANEL), 19, 353
 - DISPLAY statement (CQLIM), 19, 449
 - DISPLAY statement (CSPATIALREG), 19, 493
 - DISPLAY statement (ECM), 19, 565
 - DISPLAY statement (HMM), 19, 653
 - DISPLAY statement (SEVSELECT), 19, 843
- EXMSPEC object
 - TSM Package, 1145
- EXPECTED
 - OUTPUT statement (CQLIM), 456
- EXTERNALCOUNTS statement
 - CCDM procedure, 131
- FCONV option
 - PROC CARIMA statement, 43
 - PROC CCDM statement, 43
 - PROC CCOPULA statement, 43
 - PROC CESM statement, 43
 - PROC CNTSELECT statement, 43
 - PROC CPANEL statement, 43

- PROC CQLIM statement, 43
- PROC CSPATIALREG statement, 43
- PROC HMM statement, 43
- PROC SEVSELECT statement, 43
- FCONV2 option
 - PROC CARIMA statement, 43
 - PROC CCDM statement, 43
 - PROC CCOPULA statement, 43
 - PROC CESM statement, 43
 - PROC CNTSELECT statement, 43
 - PROC CPANEL statement, 43
 - PROC CQLIM statement, 43
 - PROC CSPATIALREG statement, 43
 - PROC HMM statement, 43
 - PROC SEVSELECT statement, 43
- FDONE option
 - MODEL statement (CPANEL), 359
- FEASTOL= option
 - OPTIMIZE statement (HMM), 661
- FILTER statement, 656
- FIT statement
 - CCOPULA procedure, 218
- FIXONE option
 - MODEL statement (CPANEL), 360
- FIXONETIME option
 - MODEL statement (CPANEL), 360
- FIXTWO option
 - MODEL statement (CPANEL), 360
- FORECAST statement, 656
 - CARIMA procedure, 79
 - CESM procedure, 255
- FORMAT= option
 - ID statement (CARIMA), 81
 - ID statement (CESM), 259
 - ID statement (TSMODEL), 1014
- FREQ object
 - TSA package, 1071
- FREQ package
 - GetResult method, 1074
 - Initialize method, 1074
 - Nvalues method, 1074
 - OUTFREQ object, 1075
 - Run method, 1074
 - SetOption method, 1074
 - SetY method, 1075
- FREQ statement
 - CNTSELECT procedure, 302
- FREQ= option
 - LIBNAME statement (SASEMOOD), 787
- FRONTIER option
 - ENDOGENOUS statement (CQLIM), 451, 455
- FSIZE option
 - PROC CARIMA statement, 44
 - PROC CCDM statement, 44
 - PROC CCOPULA statement, 44
 - PROC CESM statement, 44
 - PROC CNTSELECT statement, 44
 - PROC CPANEL statement, 44
 - PROC CQLIM statement, 44
 - PROC CSPATIALREG statement, 44
 - PROC HMM statement, 44
 - PROC SEVSELECT statement, 44
- FTOL option
 - PROC CARIMA statement, 43
 - PROC CCDM statement, 43
 - PROC CCOPULA statement, 43
 - PROC CESM statement, 43
 - PROC CNTSELECT statement, 43
 - PROC CPANEL statement, 43
 - PROC CQLIM statement, 43
 - PROC CSPATIALREG statement, 43
 - PROC HMM statement, 43
 - PROC SEVSELECT statement, 43
- FTOL2 option
 - PROC CARIMA statement, 43
 - PROC CCDM statement, 43
 - PROC CCOPULA statement, 43
 - PROC CESM statement, 43
 - PROC CNTSELECT statement, 43
 - PROC CPANEL statement, 43
 - PROC CQLIM statement, 43
 - PROC CSPATIALREG statement, 43
 - PROC HMM statement, 43
 - PROC SEVSELECT statement, 43
- FUNCTIONS= option
 - OUTPUT statement (SEVSELECT), 853
- GCONV option
 - PROC CARIMA statement, 44
 - PROC CCDM statement, 44
 - PROC CCOPULA statement, 44
 - PROC CESM statement, 44
 - PROC CNTSELECT statement, 44
 - PROC CPANEL statement, 44
 - PROC CQLIM statement, 44
 - PROC CSPATIALREG statement, 44
 - PROC HMM statement, 44
 - PROC SEVSELECT statement, 44
- GCONV2 option
 - PROC CARIMA statement, 44
 - PROC CCDM statement, 44
 - PROC CCOPULA statement, 44
 - PROC CESM statement, 44
 - PROC CNTSELECT statement, 44
 - PROC CPANEL statement, 44
 - PROC CQLIM statement, 44
 - PROC CSPATIALREG statement, 44
 - PROC HMM statement, 44

- PROC SEVSELECT statement, 44
- GDELTA= option
 - OUTPUT statement (CNTSELECT), 306
- GetForecast method, 1198
 - TSM Package, 1121, 1127
- GetResult method
 - CORR package, 1042
 - CROSSCORR package, 1051
 - DECOMP package, 1057
 - FREQ package, 1074
- GOCAS= option
 - LIBNAME statement (SASEMOOD), 789
- GROUPID= option
 - PROC CNTSELECT statement, 295
- GTOL option
 - PROC CARIMA statement, 44
 - PROC CCDM statement, 44
 - PROC CCOPULA statement, 44
 - PROC CESM statement, 44
 - PROC CNTSELECT statement, 44
 - PROC CPANEL statement, 44
 - PROC CQLIM statement, 44
 - PROC CSPATIALREG statement, 44
 - PROC HMM statement, 44
 - PROC SEVSELECT statement, 44
- GTOL2 option
 - PROC CARIMA statement, 44
 - PROC CCDM statement, 44
 - PROC CCOPULA statement, 44
 - PROC CESM statement, 44
 - PROC CNTSELECT statement, 44
 - PROC CPANEL statement, 44
 - PROC CQLIM statement, 44
 - PROC CSPATIALREG statement, 44
 - PROC HMM statement, 44
 - PROC SEVSELECT statement, 44
- HCCME= option
 - MODEL statement (CPANEL), 364
- HIERARCHY= option
 - SELECTION statement (CARIMA), 39
 - SELECTION statement (CCDM), 39
 - SELECTION statement (CCOPULA), 39
 - SELECTION statement (CESM), 39
 - SELECTION statement (CNTSELECT), 39
 - SELECTION statement (CPANEL), 39
 - SELECTION statement (CQLIM), 39
 - SELECTION statement (CSPATIALREG), 39
 - SELECTION statement (HMM), 39
 - SELECTION statement (SEVSELECT), 39
- HMM procedure, 647
 - syntax, 647
- HMM procedure, CLASS statement
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- HMM procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 653
 - EXCLUDE option, 18, 653
 - EXCLUDEALL option, 19, 653
 - TRACE option, 19, 654
- HMM procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 654
 - NOREPLACE option, 19, 654
 - REPEATED option, 19, 654
- HMM procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- HMM procedure, OPTIMIZATION statement
 - ADAPTIVEDECAY= option, 662
 - ANNEALINGRATE= option, 662
 - BUFFER= option, 663
 - CHAIN= option, 663
 - LEARNINGRATE= option, 663
 - LENGTH= option, 663
 - MINIBATCHSIZE= option, 663
 - MOMENTUM= option, 663
 - NONADUPDATE option, 662
 - USELOCKING option, 664
- HMM procedure, PROC HMM statement
 - ABSCONV option, 42
 - ABSFCNV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - DATA= option, 650
 - FCONV option, 43

- FCONV2 option, 43
- FSIZE option, 44
- FTOL option, 43
- FTOL2 option, 43
- GCONV option, 44
- GCONV2 option, 44
- GTOL option, 44
- GTOL2 option, 44
- MAXFUNC= option, 45
- MAXITER= option, 45
- MAXTIME= option, 45
- MINITER= option, 45
- TECHNIQUE= option, 45
- XCONV option, 46
- XSIZE option, 46
- XTOL option, 46
- HMM procedure, SELECTION statement
 - CHOOSE= option, 37
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37
 - SELECTION= option, 41
 - STOP= option, 38
 - STOPHORIZON= option, 41
- HTAYLOR option
 - MODEL statement (CPANEL), 360
- I function, 693
- IACF function
 - TSA package, 1088
- ID statement
 - CARIMA procedure, 79
 - CESM procedure, 257
 - CPANEL procedure, 355
 - HMM procedure, 657
 - TSINFO procedure, 989
 - TSMODEL procedure, 1009
- ID= option
 - EXTERNALCOUNTS statement (CCDM), 131
- IDENTIFY statement
 - CARIMA procedure, 83
- IDMSPEC object
 - TSM Package, 1149
- IGNORECOV option
 - PROC CCDM statement, 119
- IMPACT option
 - PROC CSPATIALREG statement, 489
- IN= option
 - ESTIMATE statement (HMM), 655
 - LEARN statement (HMM), 655
- INCLUDE option
 - SCALEMODEL statement (SEVSELECT), 860
- INCLUDEALL option
 - DISPLAYOUT statement (CCDM), 19, 131
 - DISPLAYOUT statement (CCOPULA), 19, 217
 - DISPLAYOUT statement (CNTSELECT), 19, 300
 - DISPLAYOUT statement (CPANEL), 19, 354
 - DISPLAYOUT statement (CQLIM), 19, 449
 - DISPLAYOUT statement (CSPATIALREG), 19, 494
 - DISPLAYOUT statement (ECM), 19, 565
 - DISPLAYOUT statement (HMM), 19, 654
 - DISPLAYOUT statement (SEVSELECT), 19, 844
- INEST= option
 - PROC SEVSELECT statement, 831
- INFORMATIVE option
 - SCALEMODEL statement (SEVSELECT), 860
- INFUNCDEF= option
 - DIST statement (SEVSELECT), 846
- INIT statement
 - CNTSELECT procedure, 303
 - CQLIM procedure, 453
 - CSPATIALREG procedure, 495
- INIT= option
 - DIST statement (SEVSELECT), 846
- INITIAL statement
 - HMM procedure, 657
- Initialize method
 - CORR package, 1043
 - CROSSCORR package, 1051
 - DECOMP package, 1057
 - DPF package, 1066
 - FREQ package, 1074
 - TSM Package, 1122, 1128
- INITSAMPLE option
 - PROC SEVSELECT statement, 840
- INMODEL= option
 - SCORE statement (HMM), 667
- INOBJ= option
 - PROC TSMODEL statement, 1005
- INSCALAR= option
 - PROC TSMODEL statement, 1005
- INSCALARS statement
 - TSMODEL procedure, 1017
- INSTRUMENTS statement
 - CPANEL procedure, 356
- INTERMITTENCYTEST function
 - TSA package, 1089
- INTERVAL= option

- ID statement (CARIMA), 79
- ID statement (CESM), 257
- ID statement (TSMODEL), 1009
- ITPRINT option
 - CQLIM procedure, 445
 - PROC CCOPULA statement, 221
- IVBTWNG option
 - MODEL statement (CPANEL), 360
- IVFIXONE option
 - MODEL statement (CPANEL), 360
- IVFIXONETIME option
 - MODEL statement (CPANEL), 360
- IVFIXTWO option
 - MODEL statement (CPANEL), 360
- IVG2SLS option
 - MODEL statement (CPANEL), 360
- IVPOOLED option
 - MODEL statement (CPANEL), 360
- IVRANONE option
 - MODEL statement (CPANEL), 361
- IVRANTWO option
 - MODEL statement (CPANEL), 361
- J function, 693
- KNOTMAX= option
 - EFFECT statement, spline (CARIMA), 28
 - EFFECT statement, spline (CCDM), 28
 - EFFECT statement, spline (CCOPULA), 28
 - EFFECT statement, spline (CESM), 28
 - EFFECT statement, spline (CNTSELECT), 28
 - EFFECT statement, spline (CPANEL), 28
 - EFFECT statement, spline (CQLIM), 28
 - EFFECT statement, spline (CSPATIALREG), 28
 - EFFECT statement, spline (HMM), 28
 - EFFECT statement, spline (SEVSELECT), 28
- KNOTMETHOD option
 - EFFECT statement, spline (CARIMA), 28
 - EFFECT statement, spline (CCDM), 28
 - EFFECT statement, spline (CCOPULA), 28
 - EFFECT statement, spline (CESM), 28
 - EFFECT statement, spline (CNTSELECT), 28
 - EFFECT statement, spline (CPANEL), 28
 - EFFECT statement, spline (CQLIM), 28
 - EFFECT statement, spline (CSPATIALREG), 28
 - EFFECT statement, spline (HMM), 28
 - EFFECT statement, spline (SEVSELECT), 28
- KNOTMIN= option
 - EFFECT statement, spline (CARIMA), 29
 - EFFECT statement, spline (CCDM), 29
 - EFFECT statement, spline (CCOPULA), 29
 - EFFECT statement, spline (CESM), 29
 - EFFECT statement, spline (CNTSELECT), 29
 - EFFECT statement, spline (CPANEL), 29
- EFFECT statement, spline (CQLIM), 29
- EFFECT statement, spline (CSPATIALREG), 29
- EFFECT statement, spline (HMM), 29
- EFFECT statement, spline (SEVSELECT), 29
- LABELSWITCH= option
 - PROC HMM statement, 650
- LAMBDA= option
 - OUTPUT statement (CNTSELECT), 306
- LEAD= number
 - FORECAST statement (CARIMA), 79
- LEAD= option
 - FORECAST statement (CESM), 255
 - FORECAST statement (HMM), 656
 - PROC TSMODEL statement, 1006
- LEARN statement
 - HMM procedure, 655
- LEARNINGRATE=option
 - OPTIMIZATION statement, 663
- LEFTCENSORED= option
 - LOSS statement (SEVSELECT), 849
- LEFTTRUNCATED= option
 - LOSS statement (SEVSELECT), 850
- LEFTTRUNCATION= option
 - PROC CCDM statement, 119
- LENGTH=option
 - OPTIMIZATION statement, 663
- LIBNAME libref SASMOOD statement, 785
- LISTONLY option
 - DIST statement (SEVSELECT), 846
- LM option
 - TEST statement (CNTSELECT), 310
 - TEST statement (CPANEL), 368
 - TEST statement (CQLIM), 458
 - TEST statement (CSPATIALREG), 499
- LOGCONTROL= option
 - PROC TSMODEL statement, 1006
- LOGSELECT procedure, OUTPUT statement
 - OUT= option, 305
- LOSS statement
 - SEVSELECT procedure, 849
- LOWERBOUND= option
 - ENDOGENOUS statement (CQLIM), 451, 454, 455
- LR option
 - TEST statement (CNTSELECT), 310
 - TEST statement (CPANEL), 368
 - TEST statement (CQLIM), 458
 - TEST statement (CSPATIALREG), 499
- MARGINAL
 - OUTPUT statement (CQLIM), 456
- MARGINAL statement
 - ECM procedure, 566

- MARGINALSOUT option
 - OUTPUT statement (ECM), 567
- MAXCOUNTDRAW= option
 - PROC CCDM statement, 119
- MAXEFFECTS= option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37
- MAXFUNC= option
 - PROC CARIMA statement, 45
 - PROC CCDM statement, 45
 - PROC CCOPULA statement, 45
 - PROC CESM statement, 45
 - PROC CNTSELECT statement, 45
 - PROC CPANEL statement, 45
 - PROC CQLIM statement, 45
 - PROC CSPATIALREG statement, 45
 - PROC HMM statement, 45
 - PROC SEVSELECT statement, 45
- MAXITER= option
 - OPTIMIZE statement (HMM), 664
 - PROC CARIMA statement, 45
 - PROC CCDM statement, 45
 - PROC CCOPULA statement, 45
 - PROC CESM statement, 45
 - PROC CNTSELECT statement, 45
 - PROC CPANEL statement, 45
 - PROC CQLIM statement, 45
 - PROC CSPATIALREG statement, 45
 - PROC ECM statement, 560
 - PROC HMM statement, 45
 - PROC SEVSELECT statement, 45
- MAXITER= value
 - ESTIMATE statement (CARIMA), 77
- MAXSTEPS= option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37
- MAXTIME= option
 - OPTIMIZE statement (HMM), 664
- PROC CARIMA statement, 45
- PROC CCDM statement, 45
- PROC CCOPULA statement, 45
- PROC CESM statement, 45
- PROC CNTSELECT statement, 45
- PROC CPANEL statement, 45
- PROC CQLIM statement, 45
- PROC CSPATIALREG statement, 45
- PROC HMM statement, 45
- PROC SEVSELECT statement, 45
- MDEGREE option
 - EFFECT statement, polynomial (CARIMA), 25
 - EFFECT statement, polynomial (CCDM), 25
 - EFFECT statement, polynomial (CCOPULA), 25
 - EFFECT statement, polynomial (CESM), 25
 - EFFECT statement, polynomial (CNTSELECT), 25
 - EFFECT statement, polynomial (CPANEL), 25
 - EFFECT statement, polynomial (CQLIM), 25
 - EFFECT statement, polynomial (CSPATIALREG), 25
 - EFFECT statement, polynomial (HMM), 25
 - EFFECT statement, polynomial (SEVSELECT), 25
- MEDIAN option
 - FORECAST statement (CESM), 256
- METHOD= option
 - ESTIMATE statement (CARIMA), 77
 - FORECAST statement (CESM), 256
 - MODEL statement (HMM), 659
 - PROC CNTSELECT statement, 296
 - PROC CQLIM statement, 446
 - PROC CSPATIALREG statement, 491
 - SELECTION statement (CARIMA), 36
 - SELECTION statement (CCDM), 36
 - SELECTION statement (CCOPULA), 36
 - SELECTION statement (CESM), 36
 - SELECTION statement (CNTSELECT), 36, 308
 - SELECTION statement (CPANEL), 36
 - SELECTION statement (CQLIM), 36
 - SELECTION statement (CSPATIALREG), 36
 - SELECTION statement (HMM), 36
 - SELECTION statement (SEVSELECT), 36, 861
- MILLS
 - OUTPUT statement (CQLIM), 456
- MINEFFECTS= option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37

- SELECTION statement (HMM), 37
- SELECTION statement (SEVSELECT), 37
- MINIBATCHSIZE=option
 - OPTIMIZATION statement, 663
- MINITER= option
 - PROC CARIMA statement, 45
 - PROC CCDM statement, 45
 - PROC CCOPULA statement, 45
 - PROC CESM statement, 45
 - PROC CNTSELECT statement, 45
 - PROC CPANEL statement, 45
 - PROC CQLIM statement, 45
 - PROC CSPATIALREG statement, 45
 - PROC HMM statement, 45
 - PROC SEVSELECT statement, 45
- MISSING option
 - CLASS statement (CARIMA), 15
 - CLASS statement (CCDM), 15
 - CLASS statement (CCOPULA), 15
 - CLASS statement (CESM), 15
 - CLASS statement (CNTSELECT), 15
 - CLASS statement (CPANEL), 15
 - CLASS statement (CQLIM), 15
 - CLASS statement (CSPATIALREG), 15
 - CLASS statement (HMM), 15
 - CLASS statement (SEVSELECT), 15
- MNEMONIC= option
 - LIBNAME statement (SASEMOOD), 789
- MODE= option
 - OUTPUT statement (CNTSELECT), 306
- MODEL statement
 - CNTSELECT procedure, 303
 - CPANEL procedure, 357
 - CQLIM procedure, 453
 - CSPATIALREG procedure, 495
 - HMM procedure, 658
- MOMENTUM=option
 - OPTIMIZATION statement, 663
- MOVINGSUMMARY function
 - TSA package, 1091
- MSRANGE= option
 - OPTIMIZE statement (HMM), 661
- MSTAT option
 - COMPARE statement (CPANEL), 350
- MU= option
 - OUTPUT statement (CNTSELECT), 306
- MU= value
 - ESTIMATE statement (CARIMA), 78
- MULTISTART= option
 - OPTIMIZE statement (HMM), 661
- NATURALCUBIC option
 - EFFECT statement, spline (CARIMA), 29
 - EFFECT statement, spline (CCDM), 29
 - EFFECT statement, spline (CCOPULA), 29
 - EFFECT statement, spline (CESM), 29
 - EFFECT statement, spline (CNTSELECT), 29
 - EFFECT statement, spline (CPANEL), 29
 - EFFECT statement, spline (CQLIM), 29
 - EFFECT statement, spline (CSPATIALREG), 29
 - EFFECT statement, spline (HMM), 29
 - EFFECT statement, spline (SEVSELECT), 29
- NCOMPONENT= option
 - MODEL statement (HMM), 659
- NDRAW= option
 - CQLIM procedure, 446
- NEWRAP option
 - METHOD= option, PROC CSPATIALREG statement, 491
- nfor method
 - TSM Package, 1122, 1128
- NLFORMAT= option
 - ID statement (TSMODEL), 1014
- NLOPTIONS statement
 - SEVSELECT procedure, 852
- NMSIMP option
 - METHOD= option, PROC CSPATIALREG statement, 491
- NOBALANCETHREADS option
 - PROC SEVSELECT statement, 841
- NOCLPRINT option
 - PROC SEVSELECT statement, 831
- NOCONSTFITSTATS option
 - PROC SEVSELECT statement, 841
- NOCORR option
 - PROC CCOPULA statement, 221
- NOCURRENTX option
 - MODEL statement (HMM), 659
- NOEFFECT option
 - EFFECT statement, multimember (CARIMA), 23
 - EFFECT statement, multimember (CCDM), 23
 - EFFECT statement, multimember (CCOPULA), 23
 - EFFECT statement, multimember (CESM), 23
 - EFFECT statement, multimember (CNTSELECT), 23
 - EFFECT statement, multimember (CPANEL), 23
 - EFFECT statement, multimember (CQLIM), 23
 - EFFECT statement, multimember (CSPATIALREG), 23
 - EFFECT statement, multimember (HMM), 23
 - EFFECT statement, multimember (SEVSELECT), 23
- NOINT option
 - MODEL statement (CNTSELECT), 304
 - MODEL statement (CPANEL), 364
 - MODEL statement (CQLIM), 453
 - MODEL statement (CSPATIALREG), 495

- MODEL statement (HMM), 659
- NOLABEL option
 - MODEL statement (CPANEL), 365
- NONADUPDATE option
 - OPTIMIZATION statement, 662
- NONE option
 - METHOD= option, PROC CSPATIALREG statement, 491
- NONNORMALIZE option
 - PROC CSPATIALREG statement, 488
 - WEIGHT statement (CNTSELECT), 312
 - WEIGHT statement (CQLIM), 459
- NOPART option
 - PROC CPANEL statement, 349
- NOPRINT option
 - PROC CCDM statement, 120
 - PROC CCOPULA statement, 221
 - PROC CNTSELECT statement, 295, 304
 - PROC CPANEL statement, 349
 - PROC CSPATIALREG statement, 489, 496
 - PROC ECM statement, 560
 - PROC SEVSELECT statement, 831
- NOREPLACE option
 - DISPLAYOUT statement (CCDM), 19, 131
 - DISPLAYOUT statement (CCOPULA), 19, 217
 - DISPLAYOUT statement (CNTSELECT), 19, 300
 - DISPLAYOUT statement (CPANEL), 19, 354
 - DISPLAYOUT statement (CQLIM), 19, 450
 - DISPLAYOUT statement (CSPATIALREG), 19, 494
 - DISPLAYOUT statement (ECM), 19, 565
 - DISPLAYOUT statement (HMM), 19, 654
 - DISPLAYOUT statement (SEVSELECT), 19, 844
- NOSEPARATE option
 - EFFECT statement, polynomial (CARIMA), 25
 - EFFECT statement, polynomial (CCDM), 25
 - EFFECT statement, polynomial (CCOPULA), 25
 - EFFECT statement, polynomial (CESM), 25
 - EFFECT statement, polynomial (CNTSELECT), 25
 - EFFECT statement, polynomial (CPANEL), 25
 - EFFECT statement, polynomial (CQLIM), 25
 - EFFECT statement, polynomial (CSPATIALREG), 25
 - EFFECT statement, polynomial (HMM), 25
 - EFFECT statement, polynomial (SEVSELECT), 25
- NOSHUFFLE option
 - PROC ECM statement, 561
- NPERTURBEDSAMPLES= option
 - PROC CCDM statement, 120
- NREPLICATES= option
 - PROC CCDM statement, 120
- nrows method, 1201, 1205
 - TSA package, 1046, 1053, 1060, 1062, 1068, 1070, 1076
- NRRIDG option
 - METHOD= option, PROC CSPATIALREG statement, 491
- NSEASON= option
 - MODEL statement (HMM), 659
- NSIM= option
 - FORECAST statement (HMM), 656
- NSTATE= option
 - MODEL statement (HMM), 659
- NTHREADS= option
 - PROC TSINFO statement, 989
- NU= option
 - OUTPUT statement (CNTSELECT), 306
- Nvalues method
 - FREQ package, 1074
- NVAR= option
 - PROC CCOPULA statement, 221, 222
- OBJECTIVE= option
 - PROC SEVSELECT statement, 841
- OBJLIMIT= option
 - OPTIMIZE statement (HMM), 662
- OFFSET= option
 - MODEL statement (CNTSELECT), 304
 - SCALEMODEL statement (SEVSELECT), 860
- ONEADJSEVCOMPAT option
 - PROC CCDM statement, 121
- ONLINE option
 - FORECAST statement (HMM), 656
- Open method
 - TSM Package, 1134, 1142, 1151, 1160
- OPTIMIZE statement
 - HMM procedure, 661
- OPTTOL= option
 - OPTIMIZE statement (HMM), 662
- ORDER= option
 - CLASS statement (CARIMA), 15
 - CLASS statement (CCDM), 15
 - CLASS statement (CCOPULA), 15
 - CLASS statement (CESM), 15
 - CLASS statement (CNTSELECT), 15
 - CLASS statement (CPANEL), 15
 - CLASS statement (CQLIM), 15
 - CLASS statement (CSPATIALREG), 15
 - CLASS statement (HMM), 15
 - CLASS statement (SEVSELECT), 15
 - ENDOGENOUS statement (CQLIM), 450, 454
- ORDERSELECT option
 - SELECTION statement (CARIMA), 39
 - SELECTION statement (CCDM), 39

- SELECTION statement (CCOPULA), 39
- SELECTION statement (CESM), 39
- SELECTION statement (CNTSELECT), 39
- SELECTION statement (CPANEL), 39
- SELECTION statement (CQLIM), 39
- SELECTION statement (CSPATIALREG), 39
- SELECTION statement (HMM), 39
- SELECTION statement (SEVSELECT), 39
- OUT= option
 - DECODE statement (HMM), 652
 - ESTIMATE statement (HMM), 655
 - EVALUATE statement (HMM), 655
 - FILTER statement (HMM), 656
 - FORECAST statement (HMM), 657
 - LABELSWITCH= option, 651
 - LEARN statement (HMM), 655
 - LIBNAME statement (SASEMOOD), 789
 - OUTPUT statement (CCDM), 132
 - OUTPUT statement (CPANEL), 366
 - OUTPUT statement (CQLIM), 456
 - OUTPUT statement (CSPATIALREG), 496
 - OUTPUT statement (ECM), 567
 - OUTPUT statement (LOGSELECT), 305
 - OUTPUT statement (SEVSELECT), 852
 - OUTSUM statement (CCDM), 133
 - OUTSUM statement (ECM), 568
 - PROC CARIMA statement, 76
 - PROC CESM statement, 253
 - PROC TSMODEL statement, 1006
 - SMOOTH statement (HMM), 667
- OUTALL= option
 - ESTIMATE statement (HMM), 655
 - FORECAST statement (HMM), 657
 - LEARN statement (HMM), 655
- OUTARRAY statement
 - TSMODEL procedure, 1017
- OUTARRAY= option
 - PROC TSMODEL statement, 1007
- OUTARRAYS statement
 - TSMODEL procedure, 1017
- OUTBYID= option
 - OUTSCORELIB statement (SEVSELECT), 857
- OUTCORR object
 - CORR package, 1045
- OUTCROSSCORR object
 - CROSSCORR package, 1052
- OUTDECOMP object
 - DECOMP package, 1059
- OUTDECOMP2 object
 - DECOMP package, 1061
- OUTDPE object
 - DPF package, 1067
- OUTDPROB object
 - DPF package, 1069
- OUTDRAW= option
 - PROC CCDM statement (CCDM), 121
- OUTEST= option
 - PROC CARIMA statement, 76
 - PROC CESM statement, 254
 - PROC SEVSELECT statement, 831
- OUTFOR= option
 - PROC CARIMA statement, 76
 - PROC CESM statement, 254
- OUTFREQ object
 - FREQ package, 1075
- OUTFUNCDEF= option
 - DIST statement (SEVSELECT), 846
- OUTINTERVALDETAILS= option
 - PROC TSINFO statement, 989
- OUTLIB= option
 - OUTSCORELIB statement (SEVSELECT), 856
- OUTLOG= option
 - PROC TSMODEL statement, 1007
- OUTMODEL= option
 - SCORE statement (HMM), 667
- OUTMODELINFO= option
 - PROC SEVSELECT statement, 832
- OUTNVP object, 1199
- OUTOBJ= option
 - PROC TSMODEL statement, 1007
- OUTPARM= option
 - COMPARE statement (CPANEL), 351
- OUTPUT statement
 - CCDM procedure, 132
 - CNTSELECT procedure, 305
 - CPANEL procedure, 366
 - CQLIM procedure, 455
 - CSPATIALREG procedure, 496
 - ECM procedure, 567
 - SEVSELECT procedure, 852
- OUTSCALAR statement
 - TSMODEL procedure, 1017
- OUTSCALAR= option
 - PROC TSMODEL statement, 1008
- OUTSCALARS statement
 - TSMODEL procedure, 1017
- OUTSCORELIB statement
 - SEVSELECT procedure, 856
- OUTSTAT= option
 - COMPARE statement (CPANEL), 351
 - PROC CESM statement, 254
 - PROC HMM statement, 652
 - PROC SEVSELECT statement, 832
- OUTSTORE= option
 - PROC SEVSELECT statement, 832
- OUTSUM statement
 - CCDM procedure, 133
 - ECM procedure, 568

- OUTSUM= option
 - PROC CARIMA statement, 76
 - PROC CESM statement, 254
 - PROC TSMODEL statement, 1008
- P= option
 - OUTPUT statement (CPANEL), 366
- P= value
 - ESTIMATE statement (CARIMA), 78
- PACF function
 - TSA package, 1094
- PARAM= option
 - CLASS statement (CARIMA), 15
 - CLASS statement (CCDM), 15
 - CLASS statement (CCOPULA), 15
 - CLASS statement (CESM), 15
 - CLASS statement (CNTSELECT), 15
 - CLASS statement (CPANEL), 15
 - CLASS statement (CQLIM), 15
 - CLASS statement (CSPATIALREG), 15
 - CLASS statement (HMM), 15
 - CLASS statement (SEVSELECT), 15
- PARAMETER= option
 - CNTSELECT statement (CNTSELECT), 304
 - MODEL statement (CNTSELECT), 304
- PASS= option
 - LIBNAME statement (SASEMOOD), 789
- PCTLDEF= option
 - MARGINAL statement (ECM), 567
 - PROC CCDM statement, 121
 - PROC ECM statement, 561
- PCTLNAME= option
 - OUTSUM statement (CCDM), 135
- PCTLNDEC= option
 - OUTSUM statement (CCDM), 136
 - OUTSUM statement (ECM), 568
- PCTLPRE= option
 - OUTSUM statement (ECM), 568
- PCTLPTS= option
 - OUTSUM statement (CCDM), 135
 - OUTSUM statement (ECM), 569
- PERTURBMODE= option
 - PROC CCDM statement, 122
- PERTURBOUT option
 - OUTPUT statement (CCDM), 133
- PLOTS= option
 - PROC CCDM statement, 122
 - PROC SEVSELECT statement, 832
- POOLED option
 - MODEL statement (CPANEL), 361
- PRED= option
 - OUTPUT statement (CNTSELECT), 305
 - OUTPUT statement (CSPATIALREG), 497
- PREDETERMINED statement
 - CPANEL procedure, 367
- PREDICTED
 - OUTPUT statement (CQLIM), 456
- PREDICTED= option
 - OUTPUT statement (CPANEL), 366
- PRINT statement
 - TSMODEL procedure, 1018
- PRINT= option
 - PROC CCDM statement, 125
 - PROC ECM statement, 561
 - PROC SEVSELECT statement, 835
- PRINTALL option
 - MODEL statement, 304, 496
 - PROC CCOPULA statement, 222
 - PROC CNTSELECT statement, 295
 - PROC CPANEL statement, 349
 - PROC CSPATIALREG statement, 489
- PRINTFIXED option
 - MODEL statement (CPANEL), 365
- PRINTINTERNALNAMES option
 - PROC CNTSELECT statement, 296
 - PROC CSPATIALREG statement, 489
- PRINTITERFREQ= option
 - OPTIMIZE statement (HMM), 664
- PRINTLEVEL= option
 - OPTIMIZE statement (HMM), 664
- PRINTTIMING option
 - PROC CCOPULA statement, 214
 - PROC CNTSELECT statement, 296
 - PROC CSPATIALREG statement, 490
- PRIOR statement
 - HMM procedure, 664
- PROB
 - OUTPUT statement (CQLIM), 456
- PROB= option
 - OUTPUT statement (CNTSELECT), 305
- PROBALL
 - OUTPUT statement (CQLIM), 456
- PROBCOUNT option
 - OUTPUT statement (CNTSELECT), 305
- PROBOBSERVED= option
 - LOSS statement (SEVSELECT), 850
- PROBZERO= option
 - OUTPUT statement (CNTSELECT), 306
- PROC CARIMA statement, 76
- PROC CCDM statement, 118, *see* CCDM procedure
- PROC CCOPULA statement
 - CCOPULA procedure, 213
- PROC CESM statement, 253
- PROC CPANEL statement, *see* CPANEL procedure
- PROC ECM statement, 559, *see* ECM procedure
- PROC HMM statement, 650
- PROC SEVSELECT statement, 829
- PROC TSINFO statement, 988

- PROC TSMODEL statement, 1004
- PRODUCTION option
 - ENDOGENOUS statement (CQLIM), 451, 455
- programming statements
 - TSMODEL procedure, 1021
- PSTAT option
 - COMPARE statement (CPANEL), 351
- PUTTOLOG= option
 - PROC TSMODEL statement, 1008
- Q= value
 - ESTIMATE statement (CARIMA), 78
- QUANNEW option
 - METHOD= option, PROC CSPATIALREG statement, 491
- QUANTILES= option
 - OUTPUT statement (SEVSELECT), 854
- R= option
 - OUTPUT statement (CPANEL), 366
- RANONE option
 - MODEL statement (CPANEL), 361
- RANTWO option
 - MODEL statement (CPANEL), 361
- REF= option
 - CLASS statement (CARIMA), 16
 - CLASS statement (CCDM), 16
 - CLASS statement (CCOPULA), 16
 - CLASS statement (CESM), 16
 - CLASS statement (CNTSELECT), 16
 - CLASS statement (CPANEL), 16
 - CLASS statement (CQLIM), 16
 - CLASS statement (CSPATIALREG), 16
 - CLASS statement (HMM), 16
 - CLASS statement (SEVSELECT), 16
- REORDER= option
 - LABELSWITCH= option, 651
- REORDERCOMPONENT= option
 - LABELSWITCH= option, 651
- REPEATED option
 - DISPLAYOUT statement (CCDM), 19, 131
 - DISPLAYOUT statement (CCOPULA), 19, 217
 - DISPLAYOUT statement (CNTSELECT), 19, 300
 - DISPLAYOUT statement (CPANEL), 19, 354
 - DISPLAYOUT statement (CQLIM), 19, 450
 - DISPLAYOUT statement (CSPATIALREG), 19, 494
 - DISPLAYOUT statement (ECM), 19, 566
 - DISPLAYOUT statement (HMM), 19, 654
 - DISPLAYOUT statement (SEVSELECT), 19, 844
- Replay method
 - TSM Package, 1122
- REQUIRE statement
 - TSMODEL procedure, 1019
- RESID= option
 - OUTPUT statement (CSPATIALREG), 497
- RESIDUAL
 - OUTPUT statement (CQLIM), 456
- RESIDUAL= option
 - OUTPUT statement (CPANEL), 366
- RESOLUTION= option
 - PROC CCOPULA statement, 221, 222
- RESTRICT statement
 - CNTSELECT procedure, 306
 - CPANEL procedure, 367
 - CQLIM procedure, 457
 - CSPATIALREG procedure, 497
- RIGHTCENSORED= option
 - LOSS statement (SEVSELECT), 851
- RIGHTTRUNCATED= option
 - LOSS statement (SEVSELECT), 851
- RIGHTTRUNCATION= option
 - PROC CCDM statement, 125
- Run method
 - CORR package, 1044
 - CROSSCORR package, 1051
 - DECOMP package, 1057
 - DPF package, 1066
 - FREQ package, 1074
 - TSM Package, 1122, 1128
- SAMPLEFRACTION= option
 - PROC ECM statement, 563
- SAMPLEVAR= option
 - MARGINAL statement (ECM), 566
 - OUTPUT statement (CCDM), 133
- SCALE function
 - TSA package, 1095
- SCALEMODEL statement
 - SEVSELECT procedure, 858
- SCATTER
 - PROC CCOPULA statement, 221, 222
- SCENTER option
 - MODEL statement (HMM), 659
- SCORE statement
 - HMM procedure, 666
- SEASONALDECOMP function
 - TSA package, 1097
- SEASONALINDICES function
 - TSA package, 1098
- SEASONALITY= option
 - PROC CESM statement, 254
 - PROC TSMODEL statement, 1008
- SEASONTEST function
 - TSA package, 1100
- SECTION= option

- ID statement (HMM), 657
- SEED= option
 - CQLIM procedure, 446
 - PROC CCDM statement, 125
 - PROC ECM statement, 563
 - PROC HMM statement, 652
- SELECT= option
 - SELECTION statement (CARIMA), 37
 - SELECTION statement (CCDM), 37
 - SELECTION statement (CCOPULA), 37
 - SELECTION statement (CESM), 37
 - SELECTION statement (CNTSELECT), 37, 309
 - SELECTION statement (CPANEL), 37
 - SELECTION statement (CQLIM), 37
 - SELECTION statement (CSPATIALREG), 37
 - SELECTION statement (HMM), 37
 - SELECTION statement (SEVSELECT), 37, 861
- SELECTION statement
 - CARIMA procedure, 36
 - CCDM procedure, 36
 - CCOPULA procedure, 36
 - CESM procedure, 36
 - CNTSELECT procedure, 36, 308
 - CPANEL procedure, 36
 - CQLIM procedure, 36
 - CSPATIALREG procedure, 36
 - HMM procedure, 36
 - SEVSELECT procedure, 36, 860
- SELECTION= option
 - SELECTION statement (CARIMA), 41
 - SELECTION statement (CCDM), 41
 - SELECTION statement (CCOPULA), 41
 - SELECTION statement (CESM), 41
 - SELECTION statement (CNTSELECT), 41
 - SELECTION statement (CPANEL), 41
 - SELECTION statement (CQLIM), 41
 - SELECTION statement (CSPATIALREG), 41
 - SELECTION statement (HMM), 41
 - SELECTION statement (SEVSELECT), 41
- SELECTNLOTECH= option
 - PROC SEVSELECT statement, 841
- SELECTOUT option
 - PROC SEVSELECT statement, 837
- SEPARATE option
 - EFFECT statement, spline (CARIMA), 30
 - EFFECT statement, spline (CCDM), 30
 - EFFECT statement, spline (CCOPULA), 30
 - EFFECT statement, spline (CESM), 30
 - EFFECT statement, spline (CNTSELECT), 30
 - EFFECT statement, spline (CPANEL), 30
 - EFFECT statement, spline (CQLIM), 30
 - EFFECT statement, spline (CSPATIALREG), 30
 - EFFECT statement, spline (HMM), 30
 - EFFECT statement, spline (SEVSELECT), 30
- SetDiff method
 - TSM Package, 1142
- SetLambda method
 - DECOMP package, 1057
- SetMethod method
 - TSM Package, 1151
- SETMISSING= option
 - FORECAST statement (CESM), 256
 - ID statement (CARIMA), 81
 - ID statement (CESM), 259
 - ID statement (TSMODEL), 1015
 - VAR statement (TSMODEL), 1019
- SetMode method
 - DECOMP package, 1058
- SetNLags method
 - CORR package, 1044
 - CROSSCORR package, 1051
- SetNParms method
 - CORR package, 1044
- SetOption method
 - DPF package, 1066
 - FREQ package, 1074
 - TSM Package, 1123, 1129, 1134, 1142, 1152
- SetParm method
 - TSM Package, 1135, 1152
- SetResults method
 - DECOMP package, 1058
- SetSeasonality method
 - DECOMP package, 1058
- SetTFTTransform method
 - TSM Package, 1143
- SetTransform method
 - TSM Package, 1136, 1144, 1153, 1160
- SetX method
 - CROSSCORR package, 1052
- SetY method
 - CORR package, 1044
 - CROSSCORR package, 1052
 - DECOMP package, 1059
 - FREQ package, 1075
 - TSM Package, 1123, 1132
- SEVERITYEST= option
 - PROC CCDM statement, 126
- SEVERITYMODEL statement
 - CCDM procedure, 136
- SEVERITYSTORE= option
 - PROC CCDM statement, 126
- SEVSELECT procedure, 825
 - DIST statement, 845
 - LOSS statement, 849
 - NLOPTIONS statement, 852
 - OUTPUT statement, 852
 - OUTSCORELIB statement, 856
 - SCALEMODEL statement, 858

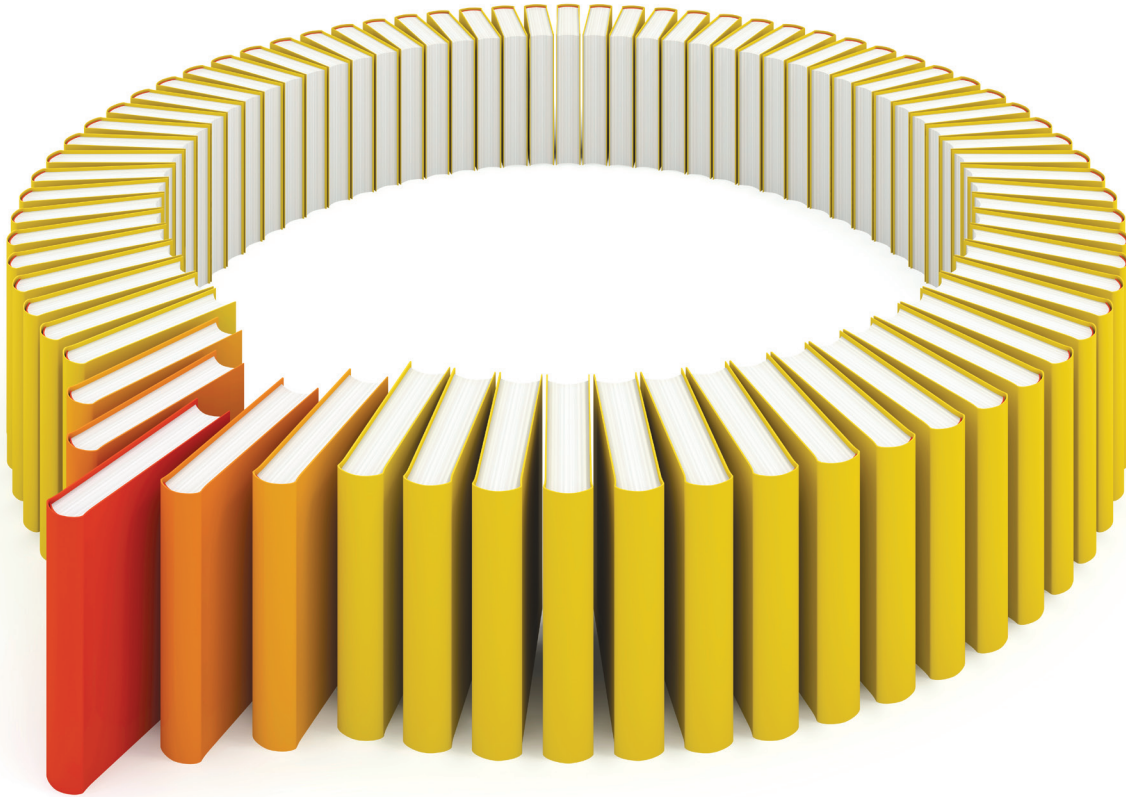
- syntax, 825
 - WEIGHT statement, 862
- SEVSELECT procedure, CLASS statement, 842
 - DESCENDING option, 15
 - MISSING option, 15
 - ORDER= option, 15
 - PARAM= option, 15
 - REF= option, 16
 - SPLIT option, 17
- SEVSELECT procedure, DISPLAY statement
 - CASESENSITIVE option, 18, 843
 - EXCLUDE option, 18, 843
 - EXCLUDEALL option, 19, 843
 - TRACE option, 19, 844
- SEVSELECT procedure, DISPLAYOUT statement
 - INCLUDEALL option, 19, 844
 - NOREPLACE option, 19, 844
 - REPEATED option, 19, 844
- SEVSELECT procedure, DIST statement
 - INFUNCDEF= option, 846
 - INIT= option, 846
 - LISTONLY option, 846
 - OUTFUNCDEF= option, 846
 - VALIDATEONLY option, 847
- SEVSELECT procedure, EFFECT statement
 - BASIS option (spline), 27
 - DATABOUNDARY option (spline), 28
 - DEGREE option (polynomial), 25
 - DEGREE option (spline), 28
 - DETAILS option (multimember), 23
 - DETAILS option (polynomial), 25
 - DETAILS option (spline), 28
 - KNOTMAX= option (spline), 28
 - KNOTMETHOD option (spline), 28
 - KNOTMIN= option (spline), 29
 - MDEGREE option (polynomial), 25
 - NATURALCUBIC option (spline), 29
 - NOEFFECT option (multimember), 23
 - NOSEPARATE option (polynomial), 25
 - SEPARATE option (spline), 30
 - SPLIT option (spline), 30
 - STANDARDIZE option (polynomial), 25
- SEVSELECT procedure, LOSS statement
 - LEFTCENSORED= option, 849
 - LEFTTRUNCATED= option, 850
 - PROBOBSERVED= option, 850
 - RIGHTCENSORED= option, 851
 - RIGHTTRUNCATED= option, 851
- SEVSELECT procedure, OUTPUT statement
 - COPYVARS= option, 853
 - FUNCTIONS= option, 853
 - OUT= option, 852
 - QUANTILES= option, 854
- SEVSELECT procedure, OUTSCORELIB statement
 - COMMONPACKAGE option, 857
 - OUTBYID= option, 857
 - OUTLIB= option, 856
- SEVSELECT procedure, PROC SEVSELECT
 - statement, 829
 - ABSCONV option, 42
 - ABSFCONV option, 42
 - ABSFTOL option, 42
 - ABSGCONV option, 42
 - ABSGTOL option, 42
 - ABSTOL option, 42
 - ABSXCONV option, 43
 - ABSXTOL option, 43
 - COVOUT option, 830
 - CRITERION= option, 830
 - DATA= option, 831
 - EDF=AUTO option, 838
 - EDF=KAPLANMEIER option, 838
 - EDF=MODIFIEDKM option, 838
 - EDF=NOTURNBULL option, 839
 - EDF=STANDARD option, 839
 - EDF=TURNBULL option, 839
 - EMPIRICALCDF= option, 838
 - FCONV option, 43
 - FCONV2 option, 43
 - FSIZE option, 44
 - FTOL option, 43
 - FTOL2 option, 43
 - GCONV option, 44
 - GCONV2 option, 44
 - GTOL option, 44
 - GTOL2 option, 44
 - INEST= option, 831
 - INITSAMPLE option, 840
 - MAXFUNC= option, 45
 - MAXITER= option, 45
 - MAXTIME= option, 45
 - MINITER= option, 45
 - NOBALANCETHREADS option, 841
 - NOCLPRINT option, 831
 - NOCONSTFITSTATS option, 841
 - NOPRINT option, 831
 - OBJECTIVE= option, 841
 - OUTEST= option, 831
 - OUTMODELINFO= option, 832
 - OUTSTAT= option, 832
 - OUTSTORE= option, 832
 - PLOTS= option, 832
 - PRINT= option, 835
 - SELECTNLOTECH= option, 841
 - SELECTOUT option, 837
 - TECHNIQUE= option, 45
 - VARDEF= option, 838
 - XCONV option, 46

- XSIZE option, 46
- XTOL option, 46
- ZEROEST option, 838
- SEVSELECT procedure, SCALEMODEL statement
 - DFMIXTURE= option, 858
 - INCLUDE option, 860
 - INFORMATIVE option, 860
 - OFFSET= option, 860
 - START option, 860
- SEVSELECT procedure, SELECTION statement, 860
 - CHOOSE= option, 37, 861
 - COMPETITIVE option, 37
 - CRITERION= option, 37
 - DETAILS= option, 38, 39
 - HIERARCHY= option, 39
 - MAXEFFECTS= option, 37
 - MAXSTEPS= option, 37
 - METHOD= option, 36, 861
 - MINEFFECTS= option, 37
 - ORDERSELECT option, 39
 - SELECT= option, 37, 861
 - SELECTION= option, 41
 - STOP= option, 38, 861
 - STOPHORIZON= option, 41
- SHAPE function, 693
- SIMILARITY function
 - TSA package, 1101
- SIMULATE statement
 - CCOPULA procedure, 222
- SIMULATEDSYMBOL statement
 - CCDM procedure, 137
- SINGULAR= value
 - ESTIMATE statement (CARIMA), 78
- SMOOTH statement
 - HMM procedure, 667
- SOLTYPE= option
 - OPTIMIZE statement (HMM), 662
- SORT= option
 - LABELSWITCH= option, 651
- SPATIALEFFECTS statement
 - CSPATIALREG procedure, 498
- SPATIALID statement
 - CSPATIALREG procedure, 498
- SPLIT option
 - CLASS statement (CARIMA), 17
 - CLASS statement (CCDM), 17
 - CLASS statement (CCOPULA), 17
 - CLASS statement (CESM), 17
 - CLASS statement (CNTSELECT), 17
 - CLASS statement (CPANEL), 17
 - CLASS statement (CQLIM), 17
 - CLASS statement (CSPATIALREG), 17
 - CLASS statement (HMM), 17
 - CLASS statement (SEVSELECT), 17
 - EFFECT statement, spline (CARIMA), 30
 - EFFECT statement, spline (CCDM), 30
 - EFFECT statement, spline (CCOPULA), 30
 - EFFECT statement, spline (CESM), 30
 - EFFECT statement, spline (CNTSELECT), 30
 - EFFECT statement, spline (CPANEL), 30
 - EFFECT statement, spline (CQLIM), 30
 - EFFECT statement, spline (CSPATIALREG), 30
 - EFFECT statement, spline (HMM), 30
 - EFFECT statement, spline (SEVSELECT), 30
- STANDARDIZE option
 - EFFECT statement, polynomial (CARIMA), 25
 - EFFECT statement, polynomial (CCDM), 25
 - EFFECT statement, polynomial (CCOPULA), 25
 - EFFECT statement, polynomial (CESM), 25
 - EFFECT statement, polynomial (CNTSELECT), 25
 - EFFECT statement, polynomial (CPANEL), 25
 - EFFECT statement, polynomial (CQLIM), 25
 - EFFECT statement, polynomial (CSPATIALREG), 25
 - EFFECT statement, polynomial (HMM), 25
 - EFFECT statement, polynomial (SEVSELECT), 25
- START option
 - SCALEMODEL statement (SEVSELECT), 860
- START= option
 - ID statement (CARIMA), 82
 - ID statement (CESM), 259
 - ID statement (TSINFO), 990
 - ID statement (TSMODEL), 1015
- STATEINDEPENDENT= name-list
 - MODEL statement, 660
- STATIONARITYTEST function
 - TSA package, 1104
- STDATE= option
 - LIBNAME statement (SASEMOOD), 790
- STOP= option
 - SELECTION statement (CARIMA), 38
 - SELECTION statement (CCDM), 38
 - SELECTION statement (CCOPULA), 38
 - SELECTION statement (CESM), 38
 - SELECTION statement (CNTSELECT), 38, 309
 - SELECTION statement (CPANEL), 38
 - SELECTION statement (CQLIM), 38
 - SELECTION statement (CSPATIALREG), 38
 - SELECTION statement (HMM), 38
 - SELECTION statement (SEVSELECT), 38, 861
- STOPHORIZON= option
 - SELECTION statement (CARIMA), 41
 - SELECTION statement (CCDM), 41
 - SELECTION statement (CCOPULA), 41
 - SELECTION statement (CESM), 41
 - SELECTION statement (CNTSELECT), 41

- SELECTION statement (CPANEL), 41
- SELECTION statement (CQLIM), 41
- SELECTION statement (CSPATIALREG), 41
- SELECTION statement (HMM), 41
- SELECTION statement (SEVSELECT), 41
- STORE statement
 - HMM procedure, 667
- STORE= option
 - PROC CNTSELECT statement, 295
- SUBMIT statement
 - TSMODEL procedure, 1019
- TAILEDFAccuracy= option
 - PROC ECM statement, 563
- TAILSTARTEDF= option
 - PROC ECM statement, 563
- TE1
 - OUTPUT statement (CQLIM), 456
- TE2
 - OUTPUT statement (CQLIM), 457
- TECHNIQUE= option
 - PROC CARIMA statement, 45
 - PROC CCDM statement, 45
 - PROC CCOPULA statement, 45
 - PROC CESM statement, 45
 - PROC CNTSELECT statement, 45
 - PROC CPANEL statement, 45
 - PROC CQLIM statement, 45
 - PROC CSPATIALREG statement, 45
 - PROC HMM statement, 45
 - PROC SEVSELECT statement, 45
- TEST statement
 - CPANEL procedure, 368
- TIME= option
 - ID statement (HMM), 657
- TOLERANCE= option
 - PROC ECM statement, 563
- TOTALLOSSVAR= option
 - OUTPUT statement (ECM), 568
- TRACE option
 - DISPLAY statement (CCDM), 19, 130
 - DISPLAY statement (CCOPULA), 19, 217
 - DISPLAY statement (CNTSELECT), 19, 299
 - DISPLAY statement (CPANEL), 19, 353
 - DISPLAY statement (CQLIM), 19, 449
 - DISPLAY statement (CSPATIALREG), 19, 494
 - DISPLAY statement (ECM), 19, 565
 - DISPLAY statement (HMM), 19, 654
 - DISPLAY statement (SEVSELECT), 19, 844
- TRANS= option
 - LIBNAME statement (SASEMOOD), 790
- TRANSFORM function
 - TSA package, 1105
- TRANSFORM= option
 - FORECAST statement (CESM), 256
- TRANSFORM= transform-option
 - ESTIMATE statement (CARIMA), 78
- TREND= option
 - MODEL statement (HMM), 660
- TRIMID= option
 - ID statement (CARIMA), 82
 - ID statement (CESM), 260
 - ID statement (TSMODEL), 1016
- TRUNCATED option
 - ENDOGENOUS statement (CQLIM), 451, 455
- TRUNCATEZEROS option
 - PROC CCDM statement, 127
- TRUREG option
 - METHOD= option, PROC CSPATIALREG statement, 491
- TSA package
 - ACCUMULATE function, 1079
 - ACCUMULATE2 Method, 1081
 - ACF function, 1083
 - ARMAORDERS function, 1085
 - CCF function, 1086
 - Collect method, 1045, 1053, 1060, 1061, 1068, 1069, 1076
 - CORR object, 1041
 - CROSSCORR object, 1048
 - DECOMP object, 1055
 - DPF object, 1064
 - FREQ object, 1071
 - IACF function, 1088
 - INTERMITTENCYTEST function, 1089
 - MOVINGSUMMARY function, 1091
 - nrows method, 1046, 1053, 1060, 1062, 1068, 1070, 1076
 - PACF function, 1094
 - SCALE function, 1095
 - SEASONALDECOMP function, 1097
 - SEASONALINDICES function, 1098
 - SEASONTEST function, 1100
 - SIMILARITY function, 1101
 - STATIONARITYTEST function, 1104
 - TRANSFORM function, 1105
 - UNBIASEDNESS function, 1107
 - WHITENOISE function, 1108
- TSINFO procedure
 - syntax, 988
- TSINFO procedure, PROC TSINFO statement
 - DATA= option, 988
 - NTHREADS= option, 989
 - OUTINTERVALDETAILS= option, 989
- TSM object
 - TSM Package, 1117
- TSM Package
 - AddARPoly method, 1139

- AddBlockSeason method, 1156
- AddComponent method, 1157
- AddCycle method, 1158
- AddDeplag method, 1158
- AddExternal method, 1119
- AddInput method, 1159
- AddMAPoly method, 1140
- AddModel method, 1126
- AddPredict method, 1126
- AddSeason method, 1159
- AddTF method, 1140
- AddTFDenPoly method, 1141
- AddTFNumPoly method, 1141
- AddX method, 1120
- ARIMASPEC object, 1137
- Autoreg method, 1156
- CFC object, 1124
- Close method, 1134, 1142, 1150, 1160
- Criterion method, 1121, 1127
- ESMSPEC object, 1132
- EXMSPEC object, 1145
- GetForecast method, 1121, 1127
- IDMSPEC object, 1149
- Initialize method, 1122, 1128
- nfor method, 1122, 1128
- Open method, 1134, 1142, 1151, 1160
- Replay method, 1122
- Run method, 1122, 1128
- SetDiff method, 1142
- SetMethod method, 1151
- SetOption method, 1123, 1129, 1134, 1142, 1152
- SetParm method, 1135, 1152
- SetTFTransform method, 1143
- SetTransform method, 1136, 1144, 1153, 1160
- SetY method, 1123, 1132
- TSM object, 1117
- TSMFOR object, 1165
- TSMINEST object, 1175
- TSMINSPEC object, 1176
- TSMPEST object, 1161
- TSMSPEC object, 1164
- TSMSTAT object, 1169
- TSMFOR object
 - TSM Package, 1165
- TSMINEST object
 - TSM Package, 1175
- TSMINSPEC object
 - TSM Package, 1176
- TSMODEL procedure, 1002, 1004
 - syntax, 1002, 1004
- TSMODEL procedure, PROC TSMODEL statement
 - DATA= option, 1004
- TSMPEST object
 - TSM Package, 1161
- TSMSPEC object
 - TSM Package, 1164
- TSMSTAT object
 - TSM Package, 1169
- TVARPRE= option
 - OUTSUM statement (ECM), 569
- TVARPTS= option
 - OUTSUM statement (ECM), 569
- TYPE= option
 - CSPATIALREG procedure, 490
 - MODEL statement (HMM), 660
 - MODEL statement, CSPATIALREG procedure, 495
 - PROC CSPATIALREG statement, 495
- UNBIASEDNESS function
 - TSA package, 1107
- UNPACK= option
 - PROC CCOPULA statement, 221, 222
- UPPERBOUND= option
 - ENDOGENOUS statement (CQLIM), 451, 454, 455
- USELOCKING option
 - OPTIMIZATION statement, 664
- USER= option
 - LIBNAME statement (SASEMOOD), 790
- UTLSTAT object, 1201
- VALIDATEONLY option
 - DIST statement (SEVSELECT), 847
- VAR statement
 - CCOPULA procedure, 223
 - ECM procedure, 570
 - TSMODEL procedure, 1018
- VARDEF= option
 - PROC CCDM statement, 128
 - PROC ECM statement, 564
 - PROC SEVSELECT statement, 838
- VARIANCE= option
 - OUTPUT statement (CNTSELECT), 306
- VCOMP= option
 - MODEL statement (CPANEL), 364
- VIEWSTORE statement
 - CCOPULA procedure, 223
 - CNTSELECT procedure, 311
- WALD option
 - TEST statement (CNTSELECT), 310
 - TEST statement (CPANEL), 368
 - TEST statement (CQLIM), 458
 - TEST statement (CSPATIALREG), 499
- WEIGHT statement
 - SEVSELECT procedure, 862
- WHITENOISE function
 - TSA package, 1108

- WMAT=option
 - PROC CSPATIALREG statement, 487
- XBETA
 - OUTPUT statement (CQLIM), 457
- XBETA= option
 - OUTPUT statement (CNTSELECT), 306
 - OUTPUT statement (CSPATIALREG), 497
- XCONV option
 - PROC CARIMA statement, 46
 - PROC CCDM statement, 46
 - PROC CCOPULA statement, 46
 - PROC CESM statement, 46
 - PROC CNTSELECT statement, 46
 - PROC CPANEL statement, 46
 - PROC CQLIM statement, 46
 - PROC CSPATIALREG statement, 46
 - PROC HMM statement, 46
 - PROC SEVSELECT statement, 46
- XLAG= option
 - MODEL statement (HMM), 660
- XSIZE option
 - PROC CARIMA statement, 46
 - PROC CCDM statement, 46
 - PROC CCOPULA statement, 46
 - PROC CESM statement, 46
 - PROC CNTSELECT statement, 46
 - PROC CPANEL statement, 46
 - PROC CQLIM statement, 46
 - PROC CSPATIALREG statement, 46
 - PROC HMM statement, 46
 - PROC SEVSELECT statement, 46
- XTOL option
 - PROC CARIMA statement, 46
 - PROC CCDM statement, 46
 - PROC CCOPULA statement, 46
 - PROC CESM statement, 46
 - PROC CNTSELECT statement, 46
 - PROC CPANEL statement, 46
 - PROC CQLIM statement, 46
 - PROC CSPATIALREG statement, 46
 - PROC HMM statement, 46
 - PROC SEVSELECT statement, 46
- YLAG= option
 - MODEL statement (HMM), 661
- ZEROEST option
 - PROC SEVSELECT statement, 838
- ZEROMODEL statement
 - CNTSELECT procedure, 312
- ZGAMMA= option
 - OUTPUT statement (CNTSELECT), 306



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

 support.sas.com/bookstore
for additional books and resources.


THE POWER TO KNOW[®]

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613