## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What’s New in SAS Econometrics 8.3 Procedures and Packages</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Shared Concepts</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>The CARIMA Procedure</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>The CCDM Procedure</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td>The CCOPULA Procedure</td>
<td>165</td>
</tr>
<tr>
<td>7</td>
<td>The CESM Procedure</td>
<td>199</td>
</tr>
<tr>
<td>8</td>
<td>The CNTSELECT Procedure</td>
<td>235</td>
</tr>
<tr>
<td>9</td>
<td>The CPANEL Procedure</td>
<td>285</td>
</tr>
<tr>
<td>10</td>
<td>The CQLIM Procedure</td>
<td>365</td>
</tr>
<tr>
<td>11</td>
<td>The CSPATIALREG Procedure</td>
<td>399</td>
</tr>
<tr>
<td>12</td>
<td>The ECM Procedure</td>
<td>465</td>
</tr>
<tr>
<td>13</td>
<td>The HMM Procedure</td>
<td>511</td>
</tr>
<tr>
<td>14</td>
<td>The SEVSELECT Procedure</td>
<td>627</td>
</tr>
<tr>
<td>15</td>
<td>The TSMODEL Procedure</td>
<td>797</td>
</tr>
<tr>
<td>16</td>
<td>Time Series Analysis Package</td>
<td>839</td>
</tr>
<tr>
<td>17</td>
<td>Time Series Model Package</td>
<td>879</td>
</tr>
<tr>
<td>18</td>
<td>Utility Package</td>
<td>947</td>
</tr>
</tbody>
</table>

### Subject Index

967

### Syntax Index

977
Chapter 1
What’s New in SAS Econometrics 8.3
Procedures and Packages

Contents

Overview ................................................................. 1
Highlights of Changes and Enhancements ......................... 1
New Procedures and Packages ........................................ 2
CARIMA Procedure .................................................... 2
CESM Procedure ....................................................... 2
ECM Procedure ......................................................... 3
Time Series Utility Package ........................................... 3
Enhancements to Procedures .......................................... 4
CCOPULA Procedure .................................................. 4
CSPATIALREG Procedure ............................................ 4
HMM Procedure ....................................................... 4
SEVSELECT Procedure ............................................... 5
Highlights of SAS Econometrics 8.2 ................................. 5

Overview

This chapter summarizes the new features available in SAS Econometrics 8.3 and the highlights of SAS Econometrics 8.2.

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, turn to the appropriate chapter to read about that feature in detail.

Highlights of Changes and Enhancements

The following new procedures and packages have been added to SAS Econometrics software:

- CARIMA procedure
- CESM procedure
- ECM procedure
- Time series utility package
New features have been added to the following SAS Econometrics procedures:

- CCOPULA procedure
- CSPATIALREG procedure
- HMM procedure
- SEVSELECT procedure

New Procedures and Packages

**CARIMA Procedure**

The new CARIMA procedure enables you to fit autoregressive integrated moving average (ARIMA) models to univariate time series data.

Specifically, the CARIMA procedure supports seasonal and factor ARIMA models and enables you to use both maximum likelihood and least squares (conditional and unconditional) estimation approaches. You can specify several data transformations such as log, logistic, square-root, and Box-Cox. Moreover, several useful estimation options, such as maximization over stable parameter regions and convergence termination criteria, are available.

For all fitted models and estimation methods, you can request multistep-ahead forecasting. You can also handle the delicate issue of unequally spaced or missing observations (or both), which are often present in transactional data.

**CESM Procedure**

The new CESM procedure generates forecasts by using exponential smoothing models that have 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.
**ECM Procedure**

The new ECM procedure helps you develop an economic capital model by analyzing losses from multiple dependent lines of business or risk categories. It enables you to estimate the distribution of the enterprise-wide losses by combining the simulation data from a copula dependency model with the empirical samples of the marginal distributions of losses in individual lines of business or risk categories.

Some key features of the ECM procedure are as follows:

- It uses novel and efficient algorithms to estimate the empirical distribution function (EDF) of each marginal variable when its sample is large and distributed across multiple nodes of a cluster. PROC ECM uses these EDF estimates to estimate the percentiles for each marginal’s loss probability that it reads from the copula simulation table. It sums up the percentile estimates of individual marginals to estimate the total loss for each observation of the copula simulation table.

- It enables you to control the accuracy of the EDF estimates, and thereby the accuracy of the percentiles, by specifying various tuning options. In particular, PROC ECM enables you to divide each random variable’s domain into body and tail regions by specifying a guidance for the EDF value at which the tail region starts. You can then specify different accuracy thresholds for the two regions. You can also specify sampling and shuffling options to control the amount of data that the EDF estimation process needs to move between the worker nodes. In general, the tuning options help you control the trade-off between accuracy and resource utilization.

- It computes estimates of the two popular risk measures for distribution of total loss: value-at-risk (VaR) and tail value-at-risk (TVaR). It can also write the sample of the total loss to an output sample table, which is distributed across worker nodes in the same manner as the copula simulation input table. You can then analyze the output sample table to compute any other risk measures of your choice.

- It accepts as input the data tables that the CCOPULA and CCDM procedures prepare. Taken together with the CNTSELECT and SEVSELECT procedures (which help you model the frequency and severity of losses in individual lines of business, respectively), PROC ECM completes the suite of procedures that SAS Econometrics offers you to implement the entire ECM development process.

**Time Series Utility Package**

The time series utility (UTL) package for the TSMODEL procedure contains a set of functions that can be used as part of programming statements in PROC TSMODEL, providing a flexible way to validate time series forecasting results and to compute and output desired statistics.

The UTL package includes the following features:

- distributed and scriptable time series computations on SAS Viya
- three time series computation functions that are invoked directly as programming statements
- forecasting results validation
- confidence limits computation
- fit and forecast statistics (accuracy measurements) computation
Enhancements to Procedures

**CCOPULA Procedure**

The following features have been added to the CCOPULA procedure:

- The BY statement specifies groups for which separate copula fitting analyses or simulations are performed.
- The FIT statement estimates the parameters for a particular copula type.
- The RESTORE= option in the SIMULATE statement enables you to simulate a copula defined by the model fitting results that are preserved in an item store. You create the item store by using the STORE= option in the FIT statement.
- The VIEWSTORE statement enables you to view the contents of an item store that was created using the STORE= option in the FIT statement.

**CSPATIALREG Procedure**

The following models have been added to the CSPATIALREG procedure:

- 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

**HMM Procedure**

The following features have been added to the HMM procedure:

- 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) are supported.
Several models that have different numbers of states or different orders of autoregressive process (or both) can be estimated together in order to help you with the important and difficult task of model selection.

The parameter estimates in a data table that is output from previous estimation can be read back in order to use those parameter estimates as the initial values for a new estimation.

---

**SEVSELECT Procedure**

The following features have been added to the SEVSELECT procedure:

- The new PLOTS= option in the PROC SEVSELECT statement helps you generate different types of plots to visually diagnose and compare how models that are based on different distributions fit the data.
- The new PLOTS= option in the SELECTION statement helps you graphically view the progress of the regression effect selection process.

---

**Highlights of SAS Econometrics 8.2**

Some users might be unfamiliar with updates made in the previous releases of SAS Econometrics. Following are highlights of enhancements in SAS Econometrics 8.2:

- The new CCDM procedure estimates a compound distribution model (CDM).
- The new CSPATIALREG procedure estimates linear spatial econometric models for cross-sectional data whose observations are spatially referenced.
- The new HMM procedure estimates hidden Markov models (HMMs) where the observables are independently distributed, conditional on the hidden states that follow a Markov chain.
- The new TSMODEL procedure provides a new environment for time series modeling and cloud computing.
- The CNTSELECT procedure adds the EFFECT and SELECTION statements to create constructed effects and perform variable selection, respectively. The VIEWSTORE statement enables you to view the contents of an item store that was previously created using the STORE= option.
- The CPANEL procedure provides additional options for the dynamic panel estimator, calculated by the general method of moments (GMM).
- The CQLIM procedure adds the CLASS statement, which enables you to model the effects of classification variables.
The SEVSELECT procedure adds the STORE= option, which enables you to save model estimates to an item-store table. The OBJECTIVE= option enables you to create a custom objective function, and the OUTSCORELIB statement creates scoring functions. The OUTMODELINFO= and OUTSTAT= options create data tables that contain information about each candidate distribution and contain convergence information and fit statistics for all candidate distributions.

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 .................................................................................. 7
Chapter Organization ......................................................................... 8
Typographical Conventions ................................................................. 8
Options Used in Examples .................................................................. 9
Where to Turn for More Information .................................................... 9
Online Documentation ........................................................................ 9
SAS Technical Support Services ......................................................... 9

Overview of SAS Econometrics Procedures

SAS Econometrics procedures provide 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 SAS Visual Data Management and Utility 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:

- **roman** 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.
- **oblique** 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.
- **italic** is used for terms that are defined in text, for emphasis, and for references to publications.
- **monospace** 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 in an ODS destination statement as follows:

```ods html style=HTMLBlue;
...;
ods html close;
```

```ods pdf style=PearlJ;
...;
ods pdf close;
```

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 .................................................................... 34
SELECTION Statement ................................................................. 35
Optimization Options ................................................................. 40
Details for SAS Econometrics Procedures ..................................... 45
Levelization of Classification Variables ........................................ 45
Specification and Parameterization of Model Effects ...................... 47
  Effect Operators .................................................................... 48
  GLM Parameterization of Classification Variables and Effects .... 50
  Nonsingular Parameterization ............................................... 55
  Class Variable Parameterization with Unbalanced Designs ........ 57
Model Selection Methods ............................................................. 58
  Forward Selection .................................................................. 58
  Backward Elimination ......................................................... 60
  Stepwise Selection ............................................................... 61
  Forward-Swap Selection ....................................................... 61
Informative Missingness .............................................................. 62
Multithreading ........................................................................... 63
Choosing an Optimization Algorithm ........................................... 64
  First- or Second-Order Algorithms ....................................... 64
  Algorithm Descriptions ....................................................... 65
References .................................................................................. 68
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 45 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: Language Reference. 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:

```sas
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:

```sas
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:

```sas
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: Language Reference*.

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 45.

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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
</tbody>
</table>
### Table 3.1  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Allows design columns for a variable to enter or leave the model independently</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Value of ORDER=</th>
<th>Levels Sorted By</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORMATTED</td>
<td>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.</td>
</tr>
<tr>
<td>FREQ</td>
<td>Descending frequency count (levels that have more observations come earlier in the order)</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Unformatted value. The sort order is machine-dependent.</td>
</tr>
</tbody>
</table>

For more information about sort order, see the chapter about the SORT procedure in *SAS Visual Data Management and Utility 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=

<table>
<thead>
<tr>
<th>Value of PARAM=</th>
<th>Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFFECT</td>
<td>Effect coding. The REF= option in the CLASS statement determines the reference level.</td>
</tr>
<tr>
<td>GLM</td>
<td>Less-than-full-rank reference cell coding. This keyword can be used only as a global-option 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.</td>
</tr>
<tr>
<td>ORDINAL</td>
<td>Cumulative parameterization for an ordinal CLASS variable</td>
</tr>
<tr>
<td>THERMOMETER</td>
<td></td>
</tr>
<tr>
<td>POLYNOMIAL</td>
<td>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.</td>
</tr>
<tr>
<td>POLY</td>
<td></td>
</tr>
<tr>
<td>REFERENCE</td>
<td>Reference cell coding. The REF= option in the CLASS statement determines the reference level.</td>
</tr>
<tr>
<td>REF</td>
<td></td>
</tr>
<tr>
<td>ORTHEFFECT</td>
<td>Orthogonalizes PARAM=EFFECT coding. The REF= option in the CLASS statement determines the reference level.</td>
</tr>
<tr>
<td>ORTHORDINAL</td>
<td>Orthogonalizes PARAM=ORDINAL coding</td>
</tr>
<tr>
<td>ORTHOTHERM</td>
<td></td>
</tr>
<tr>
<td>ORTHPOLY</td>
<td>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.</td>
</tr>
<tr>
<td>ORTHREF</td>
<td>Orthogonalizes PARAM=REFERENCE coding. The REF= option in the CLASS statement determines the reference level.</td>
</tr>
</tbody>
</table>

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 47.

**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.

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:

```plaintext
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:

```plaintext
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:

```plaintext
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

```sas
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 subsetted 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 50. 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collection Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the constituents of the collection effect</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Multimember Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the levels of the multimember effect</td>
</tr>
<tr>
<td>NOEFFECT</td>
<td>Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns</td>
</tr>
<tr>
<td>STDIZE</td>
<td>Standardizes the design matrix entries so that each observation has a sum of 1</td>
</tr>
<tr>
<td>WEIGHT=</td>
<td>Specifies the weight variable for the contributions of each classification effect</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Polynomial Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DEGREE=</td>
<td>Specifies the degree of the polynomial</td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays details of the specified polynomial</td>
</tr>
<tr>
<td>MDEGREE=</td>
<td>Specifies the maximum degree of any variable in a term of the polynomial</td>
</tr>
<tr>
<td>NOSEPARATE</td>
<td>Treats the polynomial as a single effect with multiple degrees of freedom</td>
</tr>
<tr>
<td>STANDARDIZE=</td>
<td>Specifies centering and scaling suboptions for the variables that define the polynomial</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Spline Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>BASIS=</td>
<td>Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect</td>
</tr>
</tbody>
</table>
Table 3.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATABOUNDARY</td>
<td>Uses the extremes of the data as boundary knots for a B-spline basis</td>
</tr>
<tr>
<td>DEGREE=</td>
<td>Specifies the degree of the spline effect</td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the knots and locations for each spline basis function</td>
</tr>
<tr>
<td>KNOTMAX=</td>
<td>Requests equally spaced right-side boundary knots starting at the variables’ maximum and ending at the KNOTMAX= value</td>
</tr>
<tr>
<td>KNOTMETHOD=</td>
<td>Specifies how to construct the knots for the spline effect</td>
</tr>
<tr>
<td>KNOTMIN=</td>
<td>Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables’ minimum value</td>
</tr>
<tr>
<td>NATURALCUBIC</td>
<td>Specifies a natural cubic spline basis for the spline effect</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>Treats the spline basis for each variable as a separate effect when multiple variables are specified</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Treats each design matrix column as a separate effect for selection methods</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Student</th>
<th>Score</th>
<th>Teacher1</th>
<th>Teacher2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>87</td>
<td>Tobias</td>
<td>Cohen</td>
</tr>
<tr>
<td>Tom</td>
<td>89</td>
<td>Rodriguez</td>
<td>Tobias</td>
</tr>
<tr>
<td>Fred</td>
<td>82</td>
<td>Cohen</td>
<td>Cohen</td>
</tr>
<tr>
<td>Jane</td>
<td>88</td>
<td>Tobias</td>
<td>.</td>
</tr>
<tr>
<td>Jack</td>
<td>99</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Student</th>
<th>Cohen</th>
<th>Rodriguez</th>
<th>Tobias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Tom</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fred</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Jane</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jack</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

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 \( x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1 x_2^2, x_1^2 x_2^2 \) and \( x_1 x_2^2 \):

```
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

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Center</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANGE</td>
<td>( \frac{x(n) + x(1)}{2} )</td>
<td>( \frac{x(n) - x(1)}{2} )</td>
</tr>
<tr>
<td>MOMENTS</td>
<td>( \bar{x} = \frac{\sum_{i=1}^{n} f_i x_i}{F} )</td>
<td>( \sqrt{\frac{\sum_{i=1}^{n} f_i (x_i - \bar{x})^2}{F - 1}} )</td>
</tr>
<tr>
<td>WMOMENTS</td>
<td>( \bar{x}<em>w = \frac{\sum</em>{i=1}^{n} w_i f_i x_i}{WF} )</td>
<td>( \sqrt{\frac{\sum_{i=1}^{n} w_i f_i (x_i - \bar{x}_w)^2}{F - 1}} )</td>
</tr>
</tbody>
</table>

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

**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 \(x, x^2, \ldots, x^d\), 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.
  - **NPOWERS**  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:

```text
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:

\[
\text{EFFECT spl = spline(x / knotmethod=percentiles(9));}
\]

RANGEFRACTIONS(fraction-list) places internal knots at each fraction of the ranges of the variables in the EFFECT statement. For example, if variable \(x_1\) ranges between 1 and 3 and variable \(x_2\) ranges between 0 and 20, then the following EFFECT statement uses internal knots 1.2, 2, and 2.5 for variable \(x_1\) and internal knots 2, 10, and 15 for variable \(x_2\):

\[
\text{EFFECT spl = spline(x1 x2 / knotmethod=rangefractions(.1 .5 .75));}
\]

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, \(i = 1, 2, \ldots, n - 2\), 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:

\[
\text{EFFECT spl = spline(x1 x2 / separate);} 
\]

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 \( k_1 < k_2 < k_3 < \cdots < k_n \). Then a spline of degree \( d \geq 0 \) is a function \( S(x) \) with \( d-1 \) continuous derivatives such that

\[
S(x) = \begin{cases} 
  P_0(x) & x < k_1 \\
  P_1(x) & k_i \leq x < k_{i+1}; i = 1, 2, \ldots, n-1 \\
  P_n(x) & x \geq k_n 
\end{cases}
\]

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 \( n + d + 1 \). 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 $k_1 \leq k_2 \leq k_3 < \cdots \leq k_n$.

**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 $x = 1$](image)

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 $1, x, x^2, \ldots, x^d$. These $d+1$ functions together with the $n$ truncated power functions $t_i(x), i = 1, 2, \ldots, n$ 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>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSplineDetails</td>
<td>B-spline basis details</td>
<td>BASIS=BSPLINE</td>
</tr>
<tr>
<td>CollectionLevelInfo</td>
<td>Levels of collection effects</td>
<td></td>
</tr>
<tr>
<td>MMLLevelInfo</td>
<td>Levels of multimember effects</td>
<td></td>
</tr>
<tr>
<td>PolyDetails</td>
<td>Number of variables and columns, polynomial degree, and standardization method</td>
<td>STANDARDIZE</td>
</tr>
<tr>
<td>PolyScaling</td>
<td>Centering and scaling details</td>
<td></td>
</tr>
<tr>
<td>SplineKnots</td>
<td>knot and boundary knot values</td>
<td></td>
</tr>
<tr>
<td>TPFSplineDetails</td>
<td>Truncated power function spline basis details</td>
<td>BASIS=TPF or NATURALCUBIC</td>
</tr>
</tbody>
</table>
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 58. 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>
<thead>
<tr>
<th>method-option</th>
<th>FORWARD</th>
<th>BACKWARD</th>
<th>STEPWISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOOSE</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>COMPETITIVE</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>CRITERION</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>MAXEFFECTS</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>MAXSTEPS</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>MINEFFECTS</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>SELECT</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>STOP</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
The syntax of the *method-options* that you can specify in parentheses after the SELECTION= option 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.

- **criterion** stops the selection process if the selection process produces a local extremum of this criterion 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** 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(ALL)), 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.

- **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**

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, METHOD=LAR, or METHOD=LASSO, then n is ignored and STOPHORIZON=1 is used. By default, STOPHORIZON=3.

---

**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. $\mathbf{\beta}$ denotes the $p \times 1$ vector of parameters for the optimization and $\beta_i$ 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 $f(\mathbf{\beta})$, $g(\mathbf{\beta})$, and $H(\mathbf{\beta})$, respectively. The gradient with respect to the $i$th parameter is denoted as $g_i(\mathbf{\beta})$. Superscripts in parentheses denote the iteration count; for example, $f(\mathbf{\beta})^{(k)}$ is the value of the objective function at iteration $k$. 
ABSCONV=r
ABSTOL=r
specifies an absolute function convergence criterion. For minimization, termination requires \( f(\beta^{(k)}) \leq r \), where \( \beta \) 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
ABSFXTOL=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:

\[ |f(\beta^{(k-1)}) - f(\beta^{(k)})| \leq r \]

Here, \( \beta \) 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 \( \beta^{(k)} \) is defined as the vertex that has the lowest function value and \( \beta^{(k-1)} \) 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:

\[ \max_j |g_j(\beta^{(k)})| \leq r \]

Here, \( \beta \) 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
ABSXSTOL=r
specifies an absolute parameter convergence criterion: For all techniques except NMSIMP, termination requires a small Euclidean distance between successive parameter vectors,

\[ \| \beta^{(k)} - \beta^{(k-1)} \|_2 \leq r \]

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 \( \xi^{(k)} \) that has the smallest function value to the other \( p \) simplex points \( \beta_i^{(k)} \neq \xi^{(k)} \):

\[ \delta^{(k)} = \sum_{\beta_i \neq \xi} \| \beta_i^{(k)} - \xi^{(k)} \|_1 \]

The default is \( r = 1E–8 \) for the NMSIMP technique and \( r = 0 \) otherwise.
Chapter 3: Shared Concepts

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,

\[
\frac{|f(\mathbf{\beta}_k) - f(\mathbf{\beta}_{k-1})|}{\max(|f(\mathbf{\beta}_{k-1})|, \text{FSIZE})} \leq r
\]

where FSIZE is defined by the FSIZE= option. Here, \( \mathbf{\beta} \) 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 \( \mathbf{\beta}_k \) is defined as the vertex that has the lowest function value and \( \mathbf{\beta}_{k-1} \) is defined as the vertex that has the highest function value in the simplex.

The default value is \( r = 2 \times \epsilon \) where \( \epsilon \) 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:

\[
df^{(k)} \approx f(\mathbf{\beta}^{(k)}) - f(\mathbf{\beta}^{(k)} + s^{(k)})
\]

The predicted reduction

\[
df^{(k)} = -g^{(k)}s^{(k)} - \frac{1}{2}s^{(k)^T}H^{(k)}s^{(k)}
\]

\[
= -\frac{1}{2}s^{(k)^T}g^{(k)} \leq r
\]

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

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

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

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

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

The default value is \( r = 1E-6 \) 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.
optimization options

GCONV=r

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

\[
g(\beta^{(k)})/\left[H(\beta^{(k)})\right]^{-1} g(\beta^{(k)}) \leq r
\]

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

\[
\frac{\| g(\beta^{(k)}) \|^{2/2} \| s(\beta^{(k)}) \|_{2}}{\| g(\beta^{(k)}) - g(\beta^{(k-1)}) \|_{2} \max(|f(\beta^{(k)})|, \text{FSIZE})} \leq r
\]

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

GCONV2=r

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

\[
\max_{j} \frac{|g_{j}(\beta^{(k)})|}{\sqrt{f(\beta^{(k)})H_{j,j}^{(k)}}} \leq r
\]

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.
- NRRIDG 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 64.

**XCONV**=$r$

**XTOL**=$r$

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

$$\max_j \frac{|\beta_j^{(k)} - \beta_j^{(k-1)}|}{\max(|\beta_j^{(k)}|, |\beta_j^{(k-1)}|, XSIZE)} \leq r$$
where XSIZE is defined by the XSIZE= option. \( \beta_j^{(i)} \) 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 \( r = 1E-8 \) 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>
<thead>
<tr>
<th>Obs</th>
<th>A</th>
<th>x</th>
<th>FORMAT x 3.0</th>
<th>FORMAT x 3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1.09</td>
<td>1</td>
<td>1.1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1.13</td>
<td>1</td>
<td>1.1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1.27</td>
<td>1</td>
<td>1.3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2.26</td>
<td>2</td>
<td>2.3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2.48</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3.34</td>
<td>3</td>
<td>3.3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3.34</td>
<td>3</td>
<td>3.3</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3.14</td>
<td>3</td>
<td>3.1</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Table 3.8</th>
<th>Values and Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
<td>A Value</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Table 3.9</th>
<th>Interpretation of Values of ORDER= Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of ORDER=</td>
<td>Levels Sorted By</td>
</tr>
<tr>
<td>FORMATTED</td>
<td>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.</td>
</tr>
<tr>
<td>FREQ</td>
<td>Descending frequency count (levels that have the most observations come first in the order)</td>
</tr>
<tr>
<td>INTERNAL</td>
<td>Unformatted value. The sort order is machine-dependent.</td>
</tr>
</tbody>
</table>

For more information about sort order, see the chapter about the SORT procedure in the SAS Visual Data Management and Utility 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>
<thead>
<tr>
<th>Obs</th>
<th>Value</th>
<th>Level</th>
<th>X Value</th>
<th>Level</th>
<th>FORMAT x 3.0 Value</th>
<th>Level</th>
<th>FORMAT x 3.1 Value</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1.09</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1.13</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1.27</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1.3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2.26</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2.3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2.48</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>2.5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3.34</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>3.3</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>3.34</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>3.3</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>3.14</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3.1</td>
<td>6</td>
</tr>
</tbody>
</table>

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

\[ X\beta \]

or

\[ X\beta + Z\gamma \]

The model matrices \( X \) and \( Z \) 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 45. 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 48.

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 50 and “Nonsingular Parameterization” on page 55.

**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>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>A*B</td>
<td>Crosses the levels of the effects</td>
</tr>
<tr>
<td>Nesting</td>
<td>A(B)</td>
<td>Nests A levels within B levels</td>
</tr>
<tr>
<td>Bar operator</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>At sign operator</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Dash operator</td>
<td>A1-A10</td>
<td>Specifies sequentially numbered variables</td>
</tr>
<tr>
<td>Colon operator</td>
<td>A:</td>
<td>Specifies variables with common prefix</td>
</tr>
<tr>
<td>Double dash operator</td>
<td>A-C</td>
<td>Specifies sequential variables in data set order</td>
</tr>
</tbody>
</table>

**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:
Specification and Parameterization of Model Effects

\[
\text{model } Y = A \ B \ C \quad A*B \ A*C \ B*C \quad A*B*C;
\]

\[
\text{model } Y = A|B|C;
\]

When the bar (l) 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:

\[
A \ B \ C \rightarrow \{ A \ B \} \ C \\
\rightarrow \{ A \ B \ A*B \} \ C \\
\rightarrow A \ B \ A*B \ C \ A*C \ B*C \ A*B*C
\]

- 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:

\[
\text{model } Y = A|B|C@2;
\]

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

\[
\text{model } Y = A \ B \ C \quad A*B \ A*C \ B*C;
\]

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:

```plaintext
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:

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

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

```plaintext
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 SAS Visual Data Management and Utility 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:

```plaintext
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:

```plaintext
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>
<thead>
<tr>
<th>Effect</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>Default</td>
<td>Intercept (unless the NOINT option is specified)</td>
</tr>
<tr>
<td>Regression</td>
<td>X Z</td>
<td>Continuous variables</td>
</tr>
<tr>
<td>Polynomial</td>
<td>X*Z</td>
<td>Interaction of continuous variables</td>
</tr>
<tr>
<td>Main</td>
<td>A B</td>
<td>CLASS variables</td>
</tr>
<tr>
<td>Interaction</td>
<td>A*B</td>
<td>Crossing of CLASS variables</td>
</tr>
<tr>
<td>Nested</td>
<td>A(B)</td>
<td>Main effect A nested within CLASS effect B</td>
</tr>
<tr>
<td>Continuous-by-class</td>
<td>X*A</td>
<td>Crossing of continuous and CLASS variables</td>
</tr>
<tr>
<td>Continuous-nesting-class</td>
<td>X(A)</td>
<td>Continuous variable X1 nested within CLASS variable A</td>
</tr>
<tr>
<td>General</td>
<td>X*Z*A(B)</td>
<td>Combinations of different types of effects</td>
</tr>
</tbody>
</table>
Table 3.13 shows some examples of MODEL statements that use various types of effects.

### Table 3.13  MODEL Statement Effect Examples

<table>
<thead>
<tr>
<th>Specification</th>
<th>Type of Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>model Y=X;</td>
<td>Simple regression</td>
</tr>
<tr>
<td>model Y=X Z;</td>
<td>Multiple regression</td>
</tr>
<tr>
<td>model Y=X X*X;</td>
<td>Polynomial regression</td>
</tr>
<tr>
<td>model Y=A;</td>
<td>One-way analysis of variance (ANOVA)</td>
</tr>
<tr>
<td>model Y=A B C;</td>
<td>Main-effects ANOVA</td>
</tr>
<tr>
<td>model Y=A B A*B;</td>
<td>Factorial ANOVA with interaction</td>
</tr>
<tr>
<td>model y=A B(A) C(B A);</td>
<td>Nested ANOVA</td>
</tr>
<tr>
<td>model Y=A X;</td>
<td>Analysis of covariance (ANCOVA)</td>
</tr>
<tr>
<td>model Y=A X(A);</td>
<td>Separate-slopes regression</td>
</tr>
<tr>
<td>model Y=A X<em>X</em>A;</td>
<td>Homogeneity-of-slopes regression</td>
</tr>
</tbody>
</table>

### 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 \( X X X X X X X X \), 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 \( \beta_0 \) denotes the intercept and A and B are classification variables that have two and three levels, respectively.

### Table 3.14  Example of Main Effects

<table>
<thead>
<tr>
<th>Data</th>
<th>I</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>( \beta_0 )</td>
<td>A1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Chapter 3: Shared Concepts

Table 3.14  continued

<table>
<thead>
<tr>
<th>Data</th>
<th>I</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data</th>
<th>I</th>
<th>A</th>
<th>B</th>
<th>A*B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>0</td>
<td>A1</td>
<td>A2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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):

```plaintext
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 \texttt{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).

\begin{table}[h]
\centering
\begin{tabular}{llllllllll}
\hline
Data & I & \(A\) & \(A\) & \(B(A)\) \\
\hline
\(A\) & \(B\) & \(\beta_0\) & \(A_1\) & \(A_2\) & \(B_1A_1\) & \(B_2A_1\) & \(B_3A_1\) & \(B_1A_2\) & \(B_2A_2\) & \(B_3A_2\) \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 3 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
2 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
2 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
2 & 3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{tabular}
\caption{Example of Nested Effects}
\end{table}

\textbf{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).

\begin{table}[h]
\centering
\begin{tabular}{llllll}
\hline
Data & I & \(A\) & \(X(A)\) \\
\hline
\(X\) & \(A\) & \(\beta_0\) & \(A_1\) & \(A_2\) & \(X(A_1)\) & \(X(A_2)\) \\
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 \\
19 & 2 & 1 & 0 & 1 & 0 & 19 \\
23 & 2 & 1 & 0 & 1 & 0 & 23 \\
\hline
\end{tabular}
\caption{Example of Continuous-Nesting-Class Effects}
\end{table}

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

\textbf{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.

\begin{table}[h]
\centering
\begin{tabular}{llllllll}
\hline
Data & I & \(X\) & \(A\) & \(X*A\) \\
\hline
\(X\) & \(A\) & \(\beta_0\) & \(X\) & \(A_1\) & \(A_2\) & \(X*A_1\) & \(X*A_2\) \\
21 & 1 & 1 & 21 & 1 & 0 & 21 & 0 \\
24 & 1 & 1 & 24 & 1 & 0 & 24 & 0 \\
22 & 1 & 1 & 22 & 1 & 0 & 22 & 0 \\
\hline
\end{tabular}
\caption{Example of Continuous-by-Class Effects}
\end{table}
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 $X_1^*X_2^*A^*B^*C(D E)$. 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:

```plaintext
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:

$$
A_1B_1C_1D_1 \rightarrow A_1B_2C_1D_1 \rightarrow A_2B_1C_1D_1 \rightarrow A_2B_2C_1D_1 \\
A_1B_1C_2D_1 \rightarrow A_1B_2C_2D_1 \rightarrow A_2B_1C_2D_1 \rightarrow A_2B_2C_2D_1 \\
A_1B_1C_2D_2 \rightarrow A_1B_2C_2D_2 \rightarrow A_2B_1C_2D_2 \rightarrow A_2B_2C_2D_2
$$

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.

<table>
<thead>
<tr>
<th>Reference Coding</th>
<th>Design Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>2 0</td>
<td>1</td>
</tr>
<tr>
<td>5 0</td>
<td>0</td>
</tr>
<tr>
<td>7 0</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Effect Coding</th>
<th>Design Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>2 0</td>
<td>1</td>
</tr>
<tr>
<td>5 0</td>
<td>0</td>
</tr>
<tr>
<td>7 −1</td>
<td>−1</td>
</tr>
</tbody>
</table>

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.
Chapter 3: Shared Concepts

Ordinal Coding

<table>
<thead>
<tr>
<th>Design Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Polynomial Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Matrix</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

ORTHODEFFECT

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.

<table>
<thead>
<tr>
<th>Orthogonal Effect Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Matrix</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>AOORD1</th>
<th>AOORD2</th>
<th>AOORD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1.732</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.577</td>
<td>-1.632</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.577</td>
<td>0.816</td>
<td>-1.414</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0.577</td>
<td>0.816</td>
<td>1.414</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>AOPOLY1</th>
<th>AOPOLY2</th>
<th>AOPOLY5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1.153</td>
<td>0.907</td>
<td>-0.920</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-0.734</td>
<td>-0.540</td>
<td>1.473</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0.524</td>
<td>-1.370</td>
<td>-0.920</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>1.363</td>
<td>1.004</td>
<td>0.368</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>AOREF1</th>
<th>AOREF2</th>
<th>AOREF3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.732</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-0.577</td>
<td>1.632</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-0.577</td>
<td>-0.816</td>
<td>1.414</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>-0.577</td>
<td>-0.816</td>
<td>-1.414</td>
</tr>
</tbody>
</table>

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:
Chapter 3: Shared Concepts

Design Matrix

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>A(B=1)</th>
<th>A(B=2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:
Model Selection Methods

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:

\[ \text{selection method=forward(select=SBC choose=AIC);} \]

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:

\[ \text{selection method=forward(select=SBC stop=AIC) stophorizon=1;} \]

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:

\[ \text{selection method=forward(select=SL choose=CRITERION);} \]
\[ \text{selection method=forward(select=SL stop=CRITERION);} \]

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:

\[ \text{selection method=forward(select=SL choose=CRITERION stop=20);} \]
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=none maxeffects=20 choose=AICC);
```

You can also 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

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 58.

### 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 58.

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=FORWARDSWAP 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 levelized 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 \( \beta_2 \) measures the amount by which the predicted value differs from a predicted value at \( \bar{x} \).
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.
Choosing an Optimization Algorithm

Table 3.19 Derivatives Required

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>First-Order</th>
<th>Second-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUREG</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NEWRAP</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NRRIDG</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>QUANEW</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>DBLDOG</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>CONGRA</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>NMSIMP</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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 \(\text{ABSGCONV} < 1\times10^{-5}\), \(\text{FCONV} < 2 \times \epsilon\), or \(\text{GCONV} < 1\times10^{-8}\).

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 \(\text{TECHNIQUE}={\text{QUANEW}}\) 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 (TRUREG)**

The trust region method uses the gradient $g(\beta^{(k)})$ and the Hessian matrix $H(\beta^{(k)})$; thus, it requires that the objective function $f(\beta)$ 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 $\Delta$ 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 $g(\beta^{(k)})$ and the Hessian matrix $H(\beta^{(k)})$; 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 $g(\beta^{(k)})$ and the Hessian matrix $H(\beta^{(k)})$; 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 $g(\beta^{(k)})$, 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 TRUREG, 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.
Choosing an Optimization Algorithm

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 $\alpha$ 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 $\alpha$ 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


Chapter 4
The CARIMA Procedure

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 forecasting of univariate time series. It also offers great flexibility in the types of ARIMA models that can be analyzed; it supports seasonal, subset, and factored ARIMA models.

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:

```sas
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:

```sas
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:

```sas
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:

```plaintext
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 86.
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 variable-list < options > ;
   FORECAST variable-list < 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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify data tables and options</td>
<td>PROC CARIMA</td>
<td></td>
</tr>
<tr>
<td>Request BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specify the variables to model</td>
<td>IDENTIFY</td>
<td></td>
</tr>
<tr>
<td>Specify the variables to estimate</td>
<td>ESTIMATE</td>
<td></td>
</tr>
<tr>
<td>Specify the variables to forecast</td>
<td>FORECAST</td>
<td></td>
</tr>
<tr>
<td>Data Table Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify the input data table</td>
<td>PROC CARIMA</td>
<td>DATA =</td>
</tr>
<tr>
<td>Specify the output data table</td>
<td>PROC CARIMA</td>
<td>OUT =</td>
</tr>
<tr>
<td>Specify parameter output data table</td>
<td>PROC CARIMA</td>
<td>OUTTEST =</td>
</tr>
<tr>
<td>Specify forecast output data table</td>
<td>PROC CARIMA</td>
<td>OUTFOR =</td>
</tr>
<tr>
<td>Specify summary output data table</td>
<td>PROC CARIMA</td>
<td>OUTSUM =</td>
</tr>
<tr>
<td>Accumulation Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify the accumulation frequency</td>
<td>ID</td>
<td>INTERVAL =</td>
</tr>
<tr>
<td>Specify the interval alignment</td>
<td>ID</td>
<td>ALIGN =</td>
</tr>
<tr>
<td>Specify the starting time ID value</td>
<td>ID</td>
<td>START =</td>
</tr>
<tr>
<td>Specify the ending time ID value</td>
<td>ID</td>
<td>END =</td>
</tr>
<tr>
<td>Specify the accumulation statistic</td>
<td>ID, FORECAST</td>
<td>ACCUMULATE =</td>
</tr>
<tr>
<td>Specify how to interpret missing values</td>
<td>ID, FORECAST</td>
<td>SETMISSING =</td>
</tr>
</tbody>
</table>
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 70.

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 70.

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 variable-list < 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:
Chapter 4: The CARIMA Procedure

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:

- AUTO automatically chooses between NONE and LOG on the basis of model selection criteria.
- 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

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.
Chapter 4: The CARIMA Procedure

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 81.

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=\textit{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:

- $n$ interprets a missing value as having the value $n$. You can specify any number for $n$, 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=\textit{date} | \textit{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 \textit{START}= value, the series is prefixed with missing values. If the first time ID variable value is less than the \textit{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=\textit{method} specifies the method for trimming the data in the BY groups. The output time ID variable span that is calculated by the \textit{method} is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the \textit{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 \textit{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 \textit{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**

```plaintext
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** CARIMA Processing Steps and Control Options

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
<th>Option</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accumulation</td>
<td>ACCUMULATE=ID</td>
<td>ID</td>
</tr>
<tr>
<td>2</td>
<td>Missing value interpretation</td>
<td>SETMISSING=ID</td>
<td>ID</td>
</tr>
<tr>
<td>3</td>
<td>Parameter estimation</td>
<td>P=, Q=</td>
<td>ESTIMATE</td>
</tr>
<tr>
<td>4</td>
<td>Inverse transformation</td>
<td>TRANSFORM, DIFF</td>
<td>ESTIMATE</td>
</tr>
<tr>
<td>5</td>
<td>Forecasting</td>
<td>LEAD=</td>
<td>FORECAST</td>
</tr>
</tbody>
</table>
Accumulation

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:

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>19MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>19MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>11MAY1999</td>
<td>50</td>
</tr>
<tr>
<td>12MAY1999</td>
<td>20</td>
</tr>
<tr>
<td>23MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>40</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>90</td>
</tr>
</tbody>
</table>

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>20</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>30</td>
</tr>
</tbody>
</table>

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>20</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>50</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>14.14</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>17.32</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>Variable name</td>
</tr>
<tr>
<td><em>MODEL</em></td>
<td>Forecasting model</td>
</tr>
<tr>
<td><em>MODELVAR</em></td>
<td>Model variable</td>
</tr>
<tr>
<td><em>DSVAR</em></td>
<td>Data table variable</td>
</tr>
<tr>
<td><em>VARTYPE</em></td>
<td>Variable type</td>
</tr>
<tr>
<td><em>TRANSFORM</em></td>
<td>Transformation</td>
</tr>
<tr>
<td><em>COMPONENT</em></td>
<td>Component</td>
</tr>
<tr>
<td><em>COMPMODEL</em></td>
<td>Component model</td>
</tr>
<tr>
<td><em>PARM</em></td>
<td>Parameter name</td>
</tr>
<tr>
<td><em>FACTOR</em></td>
<td>Factor</td>
</tr>
<tr>
<td><em>LAG</em></td>
<td>Lag</td>
</tr>
<tr>
<td><em>SHIFT</em></td>
<td>Shift</td>
</tr>
<tr>
<td><em>EST</em></td>
<td>Value of parameter estimate</td>
</tr>
<tr>
<td><em>STDERR</em></td>
<td>Standard error of parameter estimate</td>
</tr>
<tr>
<td><em>TVALUE</em></td>
<td>t values of parameter estimate</td>
</tr>
<tr>
<td><em>STDERR</em></td>
<td>p-values of parameter estimate</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>Variable name</td>
</tr>
<tr>
<td>ID</td>
<td>Values of the variable specified in the ID statement</td>
</tr>
<tr>
<td>ACTUAL</td>
<td>Actual values</td>
</tr>
<tr>
<td>PREDICT</td>
<td>Predicted values</td>
</tr>
<tr>
<td>ERROR</td>
<td>Prediction errors</td>
</tr>
<tr>
<td>STD</td>
<td>Prediction standard errors</td>
</tr>
<tr>
<td>UPPER</td>
<td>Upper confidence limits</td>
</tr>
<tr>
<td>LOWER</td>
<td>Lower confidence limits</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>Variable name</td>
</tr>
<tr>
<td>NOBS</td>
<td>Number of observations</td>
</tr>
<tr>
<td>N</td>
<td>Number of nonmissing observations</td>
</tr>
<tr>
<td>NMISS</td>
<td>Number of missing observations</td>
</tr>
<tr>
<td>MIN</td>
<td>Minimum value</td>
</tr>
<tr>
<td>MAX</td>
<td>Maximum value</td>
</tr>
<tr>
<td>MEAN</td>
<td>Mean value</td>
</tr>
<tr>
<td>STDDEV</td>
<td>Standard deviation</td>
</tr>
<tr>
<td><em>STATUS</em></td>
<td>Forecasting status (nonzero values imply that no forecast was generated for the series)</td>
</tr>
<tr>
<td>START</td>
<td>Starting date of each series</td>
</tr>
<tr>
<td>END</td>
<td>Ending date of each series</td>
</tr>
<tr>
<td>STARTOBS</td>
<td>Beginning observation number of each series</td>
</tr>
<tr>
<td>ENDOBS</td>
<td>Ending observation number of each series</td>
</tr>
</tbody>
</table>

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. 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 CARIMA 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.
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:

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

Output 4.1.1 indicates that the series is nonstationary and seasonal.

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)\((0,1,1)_{12}\) 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of analysis variables</td>
<td>1</td>
</tr>
<tr>
<td>Number of rows read</td>
<td>288</td>
</tr>
<tr>
<td>Number of groups read</td>
<td>1</td>
</tr>
<tr>
<td>Memory for group packages (KB)</td>
<td>15</td>
</tr>
<tr>
<td>Time to load groups (seconds)</td>
<td>0.0071570873</td>
</tr>
<tr>
<td>Minimum time ID</td>
<td>JAN1949</td>
</tr>
<tr>
<td>Maximum time ID</td>
<td>DEC1960</td>
</tr>
<tr>
<td>Minimum time periods</td>
<td>144</td>
</tr>
<tr>
<td>Maximum time periods</td>
<td>144</td>
</tr>
<tr>
<td>Number of nodes run</td>
<td>4</td>
</tr>
<tr>
<td>Number of nodes with data</td>
<td>3</td>
</tr>
<tr>
<td>Number of nodes with groups</td>
<td>1</td>
</tr>
<tr>
<td>Number of threads budgeted</td>
<td>1</td>
</tr>
<tr>
<td>Minimum thread group count</td>
<td>0</td>
</tr>
<tr>
<td>Maximum thread group count</td>
<td>1</td>
</tr>
<tr>
<td>Minimum threads active</td>
<td>1</td>
</tr>
<tr>
<td>Maximum threads active</td>
<td>1</td>
</tr>
<tr>
<td>Number of groups processed by submitted code</td>
<td>1</td>
</tr>
<tr>
<td>Number of groups failing</td>
<td>0</td>
</tr>
<tr>
<td>Elapsed time to process groups (seconds)</td>
<td>0.0317268372</td>
</tr>
</tbody>
</table>
```

The following statements plot the forecasts that are produced by the ARIMA(0,1,1)\((0,1,1)_{12}\) model. The plot is shown in Output 4.1.3.

```
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;
```
Chapter 4: The CARIMA Procedure

Output 4.1.3  Airline Passengers Forecasts

References

Chapter 5
The CCDM Procedure

Contents

Overview: CCDM Procedure ................................................................. 90
Using CAS Sessions and CAS Engine Librefs ........................................ 92
Getting Started: CCDM Procedure ....................................................... 93
Estimating a Simple Compound Distribution Model ................................ 93
Analyzing the Effect of Parameter Estimate Uncertainty on the Compound Distribution ................................................................. 96
Scenario Analysis .................................................................................... 98
Syntax: CCDM Procedure ........................................................................ 109
Functional Summary ................................................................................ 109
PROC CCDM Statement ........................................................................... 111
BY Statement ......................................................................................... 115
DISPLAY Statement ................................................................................. 116
DISPLAYOUT Statement ........................................................................... 117
EXTERNALCOUNTS Statement ................................................................. 117
OUTPUT Statement .................................................................................. 118
OUTSUM Statement .................................................................................. 119
SEVERITYMODEL Statement .................................................................. 123
Programming Statements ......................................................................... 123
Details: CCDM Procedure ....................................................................... 124
Specifying Scenario Data in the DATA= Data Table .................................. 124
Simulation Process .................................................................................... 125
Simulation of Adjusted Compound Distribution Sample .......................... 131
Parameter Perturbation Analysis .............................................................. 140
Descriptive Statistics ................................................................................ 141
Input Specification ..................................................................................... 142
Output Data Tables .................................................................................. 144
Displayed Output ..................................................................................... 146
Examples: CCDM Procedure ................................................................. 147
Example 5.1: Estimating the Probability Distribution of Insurance Payments .. 147
Example 5.2: Using Externally Simulated Count Data .............................. 152
References .............................................................................................. 163
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 $X^d_j$ represents the $d$th adjusted severity value, then PROC CCDM computes $S^d$, the $d$th aggregate adjusted loss, as

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

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:

```sas
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:

```sas
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:

```sas
  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:

```sas
/* 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 y='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;
runcas.claimcount;
run;
data mycas.claimsev;
  set claimsev;
runcas.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 92, 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 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

<table>
<thead>
<tr>
<th>The CCDM Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity Model:</strong> Gamma</td>
</tr>
<tr>
<td><strong>Count Model:</strong> Poisson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compound Distribution Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity Model</strong></td>
</tr>
<tr>
<td><strong>Count Model</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The CCDM Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity Model:</strong> Gamma</td>
</tr>
<tr>
<td><strong>Count Model:</strong> Poisson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Summary Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
</tr>
<tr>
<td><strong>Median</strong></td>
</tr>
<tr>
<td><strong>Standard Deviation</strong></td>
</tr>
<tr>
<td><strong>Interquartile Range</strong></td>
</tr>
<tr>
<td><strong>Variance</strong></td>
</tr>
<tr>
<td><strong>Minimum</strong></td>
</tr>
<tr>
<td><strong>Skewness</strong></td>
</tr>
<tr>
<td><strong>Maximum</strong></td>
</tr>
<tr>
<td><strong>Kurtosis</strong></td>
</tr>
<tr>
<td><strong>Sample Size</strong></td>
</tr>
</tbody>
</table>
Chapter 5: The CCDM Procedure

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.

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:

```plaintext
/* 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 5.2 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 5.2 Summary Statistics and Percentiles of the Perturbed Samples

<table>
<thead>
<tr>
<th>SEVERITYMODEL</th>
<th>COUNTMODEL</th>
<th><em>DRAWID</em></th>
<th><em>SAMPLEVAR</em></th>
<th>N</th>
<th>MEAN</th>
<th>STDDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>0</td>
<td>aggloss</td>
<td>10000</td>
<td>4053.66</td>
<td>3462.35</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>1</td>
<td>aggloss</td>
<td>10000</td>
<td>4006.30</td>
<td>3432.04</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>2</td>
<td>aggloss</td>
<td>10000</td>
<td>4193.14</td>
<td>3531.70</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>3</td>
<td>aggloss</td>
<td>10000</td>
<td>4244.63</td>
<td>3582.15</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>4</td>
<td>aggloss</td>
<td>10000</td>
<td>3860.79</td>
<td>3243.82</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>5</td>
<td>aggloss</td>
<td>10000</td>
<td>4010.53</td>
<td>3460.88</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>6</td>
<td>aggloss</td>
<td>10000</td>
<td>3966.60</td>
<td>3550.98</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>7</td>
<td>aggloss</td>
<td>10000</td>
<td>4077.09</td>
<td>3446.39</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>8</td>
<td>aggloss</td>
<td>10000</td>
<td>3789.16</td>
<td>3325.96</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>9</td>
<td>aggloss</td>
<td>10000</td>
<td>3941.03</td>
<td>3406.70</td>
</tr>
<tr>
<td>Gamma</td>
<td>Poisson</td>
<td>10</td>
<td>aggloss</td>
<td>10000</td>
<td>3869.82</td>
<td>3192.93</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SKEWNESS</th>
<th>KURTOSIS</th>
<th>P01</th>
<th>P05</th>
<th>P90</th>
<th>P95</th>
<th>P97_5</th>
<th>var</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.14689</td>
<td>1.67854</td>
<td>0</td>
<td>0</td>
<td>8856.80</td>
<td>10739.64</td>
<td>12539.98</td>
<td>16324.10</td>
</tr>
<tr>
<td>1.11211</td>
<td>1.39824</td>
<td>0</td>
<td>0</td>
<td>8679.20</td>
<td>10581.28</td>
<td>12311.02</td>
<td>15804.85</td>
</tr>
<tr>
<td>1.11701</td>
<td>1.61203</td>
<td>0</td>
<td>0</td>
<td>8918.42</td>
<td>10909.75</td>
<td>12848.96</td>
<td>16626.61</td>
</tr>
<tr>
<td>1.08813</td>
<td>1.38386</td>
<td>0</td>
<td>0</td>
<td>9268.95</td>
<td>11215.17</td>
<td>12923.95</td>
<td>16507.94</td>
</tr>
<tr>
<td>1.11131</td>
<td>1.66723</td>
<td>0</td>
<td>0</td>
<td>8260.49</td>
<td>9965.40</td>
<td>11639.19</td>
<td>15728.73</td>
</tr>
<tr>
<td>1.21030</td>
<td>1.88056</td>
<td>0</td>
<td>0</td>
<td>8691.93</td>
<td>10665.61</td>
<td>12723.50</td>
<td>16728.11</td>
</tr>
<tr>
<td>1.23930</td>
<td>1.93125</td>
<td>0</td>
<td>0</td>
<td>8830.83</td>
<td>10902.80</td>
<td>12824.81</td>
<td>16851.13</td>
</tr>
<tr>
<td>1.13512</td>
<td>1.64431</td>
<td>0</td>
<td>0</td>
<td>8754.72</td>
<td>10688.15</td>
<td>12297.54</td>
<td>16419.73</td>
</tr>
<tr>
<td>1.21147</td>
<td>1.97307</td>
<td>0</td>
<td>0</td>
<td>8318.50</td>
<td>10196.68</td>
<td>11966.26</td>
<td>15849.36</td>
</tr>
<tr>
<td>1.17714</td>
<td>1.69551</td>
<td>0</td>
<td>0</td>
<td>8610.97</td>
<td>10536.83</td>
<td>12439.40</td>
<td>16078.76</td>
</tr>
<tr>
<td>1.11729</td>
<td>1.66063</td>
<td>0</td>
<td>0</td>
<td>8190.14</td>
<td>9890.74</td>
<td>11439.61</td>
<td>15679.92</td>
</tr>
</tbody>
</table>

The PRINT=PERTURBSUMMARY option in the preceding PROC CCDM step produces the “Sample Perturbation Analysis” and “Sample Percentile Perturbation Analysis” tables shown in Figure 5.3. The tables show that you can expect a mean aggregate loss of about 4,011 and the standard error of the mean is 210.6. 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 16,219.7, but also the standard error of 741.6 to account for the effect of uncertainty in your frequency and severity parameter estimates.
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:

```sas
/* 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);
```

/* Simulate number of losses incurred by this policyholder */
cxbeta = cbeta(1);
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;
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:

```sas
/* 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 5.4.

![Figure 5.4 Parameter Estimates of the Count Regression Model](image)

**The CNTSELECT Procedure**

**Model fit results restored from item store COUNTREGMODEL:**

| 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:

```sas
/* 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 5.5. 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).
Now, you are ready to analyze the distribution of the aggregate loss that can be expected from a specific policyholder—for example, a 59-year-old man who has an advanced degree, earns 159,870, and drives a sedan that has a very high safety rating about 11,474 miles annually. First, you need to encode and scale this information into the appropriate regressor variables of a data table. Let that data table be named mycas.SinglePolicy, which the following DATA step simulates. The data table contains an observation as shown in Figure 5.6.

```sql
/* Generate the scenario data table for single policyholder */
data singlePolicy(keep=age gender carType annualMiles education carSafety income);
call streaminit(67897);
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;
run;
/* Load data into the CAS server */
data mycas.singlePolicy;
  set singlePolicy;
run;
```
The following PROC CCDM step to analyzes the compound distribution of the aggregate loss that the policyholder in the 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:

```plaintext
/* 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.sevregest;
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

<table>
<thead>
<tr>
<th>Severity Model</th>
<th>Lognormal Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Model Effects</td>
<td>carSafety carType income</td>
</tr>
<tr>
<td>Count Model</td>
<td>NegBin(p=2) Model in Item Store COUNTRGEMODEL</td>
</tr>
</tbody>
</table>

#### The CCDM Procedure

**Severity Model:** Logn  
**Count Model:** NegBin(p=2)

#### Sample Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>206.68803</td>
<td>Median</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>412.86705</td>
<td>Interquartile Range 245.15287</td>
</tr>
<tr>
<td>Variance</td>
<td>170459.2</td>
<td>Minimum</td>
</tr>
<tr>
<td>Skewness</td>
<td>4.14952</td>
<td>Maximum</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>27.91648</td>
<td>Sample Size 10000</td>
</tr>
</tbody>
</table>
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 5.7 continued

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>75</td>
<td>245.15287</td>
</tr>
<tr>
<td>95</td>
<td>979.49217</td>
</tr>
<tr>
<td>97.5</td>
<td>1355.4</td>
</tr>
<tr>
<td>98.5</td>
<td>1719.2</td>
</tr>
<tr>
<td>99</td>
<td>1993.0</td>
</tr>
<tr>
<td>99.5</td>
<td>2438.6</td>
</tr>
</tbody>
</table>

**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)**

<table>
<thead>
<tr>
<th>Compound Distribution Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity Model</td>
</tr>
<tr>
<td>Scale Model Effects</td>
</tr>
<tr>
<td>Count Model</td>
</tr>
</tbody>
</table>
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:

```sas
/* 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; */
  end;
run;
```

The following table provides sample summary statistics for the simulated data:

<table>
<thead>
<tr>
<th>Sample Summary Statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>205.32261</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>390.38871</td>
</tr>
<tr>
<td>Variance</td>
<td>152403.3</td>
</tr>
<tr>
<td>Skewness</td>
<td>3.18983</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>14.22054</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Percentiles</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentile 1</td>
<td>0</td>
</tr>
<tr>
<td>Percentile 5</td>
<td>0</td>
</tr>
<tr>
<td>Percentile 25</td>
<td>0</td>
</tr>
<tr>
<td>Percentile 50</td>
<td>0</td>
</tr>
<tr>
<td>Percentile 75</td>
<td>254.28305</td>
</tr>
<tr>
<td>Percentile 95</td>
<td>999.32602</td>
</tr>
<tr>
<td>Percentile 97.5</td>
<td>1359.7</td>
</tr>
<tr>
<td>Percentile 98.5</td>
<td>1658.4</td>
</tr>
<tr>
<td>Percentile 99</td>
<td>1855.1</td>
</tr>
<tr>
<td>Percentile 99.5</td>
<td>2215.6</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>policyholderId</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>annualMiles</th>
<th>education</th>
<th>carSafety</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.18</td>
<td>2</td>
<td>1</td>
<td>2.2948</td>
<td>3</td>
<td>0.99532</td>
<td>1.59870</td>
</tr>
<tr>
<td>2</td>
<td>0.66</td>
<td>2</td>
<td>1</td>
<td>2.6718</td>
<td>2</td>
<td>0.86412</td>
<td>0.84459</td>
</tr>
<tr>
<td>3</td>
<td>0.64</td>
<td>2</td>
<td>2</td>
<td>1.9528</td>
<td>1</td>
<td>0.86478</td>
<td>0.50177</td>
</tr>
<tr>
<td>4</td>
<td>0.46</td>
<td>1</td>
<td>2</td>
<td>2.6402</td>
<td>2</td>
<td>0.27062</td>
<td>1.18870</td>
</tr>
<tr>
<td>5</td>
<td>0.62</td>
<td>1</td>
<td>1</td>
<td>1.7294</td>
<td>1</td>
<td>0.32830</td>
<td>0.37694</td>
</tr>
</tbody>
</table>

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
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 5,277.1 ± 356.9 and a worst-case loss of 15,713.8 ± 1,101.4 when you define the worst-case loss as the 97.5th percentile.
If the severity of each loss follows the fitted Pareto distribution, then you can expect an average loss of $5,096.7 \pm 683.7$ and a worst-case loss of $14,640.1 \pm 2,085.3$.

If you decide to use the 99.5th percentile to define the worst-case loss, then the worst-case loss is $22,571.5 \pm 1,730.2$ for the lognormal severity model and $19,920.1 \pm 2,883.2$ for the Pareto severity model. The numbers for lognormal and Pareto are well within one standard error 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.
### Compound Distribution Information

<table>
<thead>
<tr>
<th>Severity Model</th>
<th>Pareto Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Model Effects</td>
<td>carSafety carType income</td>
</tr>
<tr>
<td>Count Model</td>
<td>NegBin(p=2) Model in Item Store COUNTREMODEL</td>
</tr>
</tbody>
</table>

#### Sample Perturbation Analysis

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Estimate</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5096.7</td>
<td>683.67393</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>3806.5</td>
<td>535.92221</td>
</tr>
<tr>
<td>Variance</td>
<td>14777021</td>
<td>4224854.6</td>
</tr>
<tr>
<td>Skewness</td>
<td>1.51122</td>
<td>0.09369</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>3.93106</td>
<td>0.98149</td>
</tr>
</tbody>
</table>

Number of Perturbed Samples = 50
Size of Each Sample = 10000

#### Sample Percentile Perturbation Analysis

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Estimate</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>162.24664</td>
<td>36.04547</td>
</tr>
<tr>
<td>5</td>
<td>689.48641</td>
<td>95.59410</td>
</tr>
<tr>
<td>25</td>
<td>2320.8</td>
<td>304.62802</td>
</tr>
<tr>
<td>50</td>
<td>4237.9</td>
<td>556.32669</td>
</tr>
<tr>
<td>75</td>
<td>6932.7</td>
<td>924.71508</td>
</tr>
<tr>
<td>95</td>
<td>12409.3</td>
<td>1713.3</td>
</tr>
<tr>
<td>97.5</td>
<td>14640.1</td>
<td>2085.3</td>
</tr>
<tr>
<td>98.5</td>
<td>16310.2</td>
<td>2305.0</td>
</tr>
<tr>
<td>99</td>
<td>17630.9</td>
<td>2530.0</td>
</tr>
<tr>
<td>99.5</td>
<td>19920.1</td>
<td>2883.2</td>
</tr>
</tbody>
</table>

Number of Perturbed Samples = 50
Size of Each Sample = 10000
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 > ;
   OUTPUT OUT= SAS-data-set < variable-name-options > < / out-option > ;
   OUTSUM OUT= SAS-data-set 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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the names of severity distribution models</td>
<td>SEVERITYMODEL</td>
<td></td>
</tr>
<tr>
<td>Specifies externally simulated count data</td>
<td>EXTERNALCOUNTS</td>
<td></td>
</tr>
<tr>
<td>Specifies where and how the full simulated samples are written</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>Specifies where and how the summary statistics of simulated samples are written</td>
<td>OUTSUM</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display (an alternative to the PRINT= option)</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td>Specifies programming statements that define an objective function</td>
<td>Programming statements</td>
<td></td>
</tr>
<tr>
<td><strong>Data Table Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data table</td>
<td>PROC CCDM</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the output data table for the individual severity draws</td>
<td>PROC CCDM</td>
<td>OUTDRAW=</td>
</tr>
<tr>
<td>Specifies the output data table for the full simulated samples</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies the output data table for the summary statistics</td>
<td>OUTSUM</td>
<td>OUT=</td>
</tr>
<tr>
<td><strong>Model Input Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the variable that contains externally simulated counts</td>
<td>EXTERNALCOUNTS</td>
<td>COUNT=</td>
</tr>
<tr>
<td>Description</td>
<td>Statement</td>
<td>Option</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Specifies the item store that contains the frequency (count) model</td>
<td>PROC CCDM</td>
<td>COUNTSTORE=</td>
</tr>
<tr>
<td>Specifies the replicate identifier variable for external counts</td>
<td>EXTERNALCOUNTS</td>
<td>ID=</td>
</tr>
<tr>
<td>Specifies the input data table for parameter estimates of the severity models</td>
<td>PROC CCDM</td>
<td>SEVERITYEST=</td>
</tr>
<tr>
<td>Specifies the item store for parameter estimates of the severity models</td>
<td>PROC CCDM</td>
<td>SEVERITYSTORE=</td>
</tr>
</tbody>
</table>

**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 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 a seed for the internal pseudorandom number generator
  
  PROC CCDM SEED=

**Output Preparation Options**

- Specifies the variables for the aggregate adjusted loss samples
  
  OUTPUT ADJSAMPLEVAR=
- 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=

**Displayed Output Options**

- Suppresses all displayed output
  
  PROC CCDM NOPRINT
- 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 123.

**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 92. 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 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. 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 92.

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 140.

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 the number is equal to $N_{\text{max}}$, then any count greater than $N_{\text{max}}$ is assumed to be equal to $N_{\text{max}}$, and only $N_{\text{max}}$ 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, then the limit is applied to each count that PROC CCDM randomly draws from the count distribution in the COUNTSTORE= item store. 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
requests that parameter perturbation analysis be conducted. The number that you specify 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 COUNTSTORE= store. If covariance estimates are not available or valid, or if you specify the IGNORECOV option, then for each parameter, a random draw is made from the univariate normal distribution whose mean and standard deviation are equal to that parameter’s
point estimate and standard error, respectively. If neither covariance nor standard error estimates are available, 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 140.

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.

OUTDRAW=CAS-libref.data-table
OUTSAMPLE=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 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 92. 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 144.

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.

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).

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 92. 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=\texttt{CAS-libref.data-table}

\texttt{SEVSTORE=\texttt{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.

\texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the caslib and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 92. The \texttt{CAS-libref} must be identical to the \texttt{CAS-libref} that you specify in the \texttt{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.

\textbf{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.

\textbf{VARDEF=}\texttt{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 \texttt{divisor}:

\textbf{DF}

sets the divisor for variance to \(N - 1\). This also changes the definitions of skewness and kurtosis.

\textbf{N}

sets the divisor to \(N\).

By default, VARDEF=DF. For more information, see the section “Descriptive Statistics” on page 141.

\section*{BY Statement}

\texttt{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.
Chapter 5: The CCDM Procedure

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 146. 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 subsetted 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.
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 146. 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** generates 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.

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. 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 128.

**OUTPUT Statement**

    OUTPUT OUT=SAS-data-set < 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 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 92. 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 do not specify the ADJSAMPLEVAR= option and there is only one adjusted severity symbol in the ADJUSTEDSEVERITY= option, then “_AGGADJSEV_” is used by default.

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.

- **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 144.
You must specify the output data table by using the following option:

\texttt{OUT=\textit{CAS-libref.data-table}}
\texttt{OUTSUM=\textit{CAS-libref.data-table}}

specifies the output data table that contains the summary statistics of each of the simulated compound distribution samples. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 92. The \textit{CAS-libref} must be identical to the \textit{CAS-libref} that you specify in the \texttt{DATA=} option.

You can control the summary statistics that appear in this data table by specifying different \texttt{statistic-keywords} and \texttt{outsum-options}.

You can request that one or more predefined statistics of the compound distribution sample be written to the \texttt{OUTSUM=} data table. For each specification of the form \texttt{statistic-keyword=<variable-name>}, the statistic that is specified by the \texttt{statistic-keyword} is written to a variable named \texttt{variable-name}. If you do not specify the \texttt{variable-name}, then the statistic is written to a variable named \texttt{statistic-keyword}. You can specify the following \texttt{statistic-keywords}:

\begin{itemize}
  \item \texttt{KURTOSIS} \texttt{KURT} specifies the kurtosis of the compound distribution sample.
  \item \texttt{MEAN} specifies the mean of the compound distribution sample.
  \item \texttt{P01} specifies the 1st percentile of the compound distribution sample.
  \item \texttt{P05} specifies the 5th percentile of the compound distribution sample.
  \item \texttt{P10} specifies the 10th percentile of the compound distribution sample.
  \item \texttt{P25} \texttt{Q1} specifies the lower or 1st quartile (the 25th percentile) of the compound distribution sample.
  \item \texttt{P50} \texttt{MEDIAN} \texttt{Q2} specifies the median (the 50th percentile) of the compound distribution sample.
  \item \texttt{P75} \texttt{Q3} specifies the upper or 3rd quartile (the 75th percentile) of the compound distribution sample.
\end{itemize}
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 CCDM 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:

\begin{verbatim}
pctlpts=10, 20, 99.6, 99.7, 99.8, 99.9
pctlpts=10, 20, 99.6 to 99.9 by 0.1
\end{verbatim}

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.
Chapter 5: The CCDM Procedure

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*<sub>n</sub> names in the PCTLNAME= option and *k*<sub>v</sub> percentile values in the PCTLPTS= option, and if *k*<sub>n</sub> < *k*<sub>v</sub>, then the first *k*<sub>n</sub> percentiles are written to the variables that you specify. The remaining *k*<sub>v</sub> – *k*<sub>n</sub> percentiles are written to the variables that have the name of the form `Pn`, where *i* 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–nineneine5
```

Then PROC CCDM writes the 10th and 20th percentiles to the variables `pten` and `pttwenty`, respectively; the 99.3rd through 99.5th percentiles to the variables `ninenine3`, `nineneine4`, and `nineneine5`, 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 CCDM 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 145.
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.Svtdist and the libraries that you specify in the CM-PLIB= SAS system option and creates a temporary CAS table named cdm_svrtdist 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 external frequency model encoded by the COUNT= variable that you specify in the EXTERNALCOUNTS statement.

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 Process” on page 125.

For more information, see the section “Simulation of Adjusted Compound Distribution Sample” on page 131.
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 126 and illustrated using an example in the section “Illustration of Aggregate Loss Simulation Process” on page 126.

- 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 128 and illustrated using an example in the section “Illustration of the Simulation Process with External Counts” on page 129.
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 131.

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 Process

PROC CCDM selects a simulation process based on whether you specify external counts or you request that PROC CCDM simulate the counts, and whether the severity or frequency model contains regression effects. The following sections describe the processes for the two 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 following simulation process.

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 that you specify in the NREPLICATES= option or the default value of that option:

1. Use the frequency model that you specify in the COUNTSTORE= option to draw a value $N$ from the count distribution. $N$ is the number of loss events that are expected to occur in the time period that is being simulated. $N$ is adjusted to conform to the upper limit by setting it equal to $\min(N, N_{\text{max}})$, where $N_{\text{max}}$ is either 1,000 or the value that you specify in 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, $X_j$ ($j = 1, \ldots, N$).

3. Add the $N$ severity values that are drawn in step 2 to compute one point, $S$, from the compound distribution as

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

Note that 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 following simulation process.

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 that you specify in the NREPLICATES= option or the default value of that option:

1. For each observation \( k \) (\( k = 1, \ldots, K \)), a count \( N_k \) is drawn from the frequency model that you specify in the COUNTSTORE= option. The parameters of this model are determined by the frequency regressors in observation \( k \). \( N_k \) represents the number of loss events that are expected to be generated by entity \( k \) in the time period that is being simulated. PROC CCDM adjusts \( N_k \) to conform to the upper limit by setting it equal to \( \min(N_k, N_{\text{max}}) \), where \( N_{\text{max}} \) is either 1,000 or the value that you specify in the MAXCOUNTDRAW= option.

2. Counts from all observations are added to compute \( N = \sum_{k=1}^{K} N_k \). \( N \) is the total number of loss events that are expected to occur in the time period that is being simulated.

3. \( N \) random draws are made from the severity distribution, and they are added to generate one point of the compound distribution sample. Each of the \( N \) draws uses one of the \( K \) observations. 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 observation that is chosen for that draw.

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: \texttt{Age} (1: female, 2: male), and \texttt{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:
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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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:
Chapter 5: The CCDM Procedure

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>count</th>
<th><em>sev</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>350 2100</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>4500</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>700 4300</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>600 1500 950</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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.

PROC CCDM uses the following simulation process 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 or the default value of that option:

1. Compute the total number of losses, $N$. If there are $K$ ($K \geq 1$) observations for the current value of the replication identifier, then $N = \sum_{k=1}^{K} N_k$, where $N_k$ is the value of the COUNT= variable for observation $k$, after it is adjusted to conform to the upper limit of either 1,000 or the value that you specify in the MAXCOUNTDRAW= option.

2. $N$ random draws are made from the severity distribution, and they are added to generate one point of the compound distribution sample.

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>extCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

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`:

```plaintext
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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>extCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

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 126. 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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>replicateId</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>extCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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:

```plaintext
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 (ReplicatId=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.

<table>
<thead>
<tr>
<th>Obs</th>
<th>replicateId</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>extCount</th>
<th><em>sev</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>700 500</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>25</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>900 1400 300</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>350 2000 150 800 600</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>250</td>
</tr>
</tbody>
</table>

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 (ReplicatId=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 ReplicatId, then 30,000 observations are written to the data table mycas.AggLoss 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.AggLoss.

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 referred to as 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 $f^d$ $(d = 1, \ldots, D)$. The function $f^d$ uses the unadjusted severity value, $X_j$, to compute and return an adjusted severity value, $X_j^d$; $f^d$ can accept as input the sum of unadjusted severity values and the sum of adjusted severity values for the severity draws made prior to evaluating $f^d$.

In particular, 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 $(j = 1, \ldots, N)$, the adjusted severity value for the $d$th symbol is

$$X_j^d = f^d(X_j, S_{j-1}, S_{j-1}^d)$$

where $S_{j-1} = \sum_{l=1}^{j-1} X_l$ is the aggregate unadjusted loss and $S_{j-1}^d = \sum_{l=1}^{j-1} X_l^d$ is the aggregate value of the $d$th adjusted severity symbol before the $j$th severity draw that generates $X_j$. The initial values of both types of aggregate losses are set to 0. In other words, $S_0 = 0$ and $S_0^d = 0$, $\forall d$.

The aggregate adjusted loss for the replication is $S^d_N$, which is denoted by $S^d$ for simplicity and defined as

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

In your severity adjustment program, you can use the following keywords as placeholders for the input arguments of the function $f^d$:

- `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.

- `ADJSEVSUMd_`
- `CUMADJSEVd_`
  - 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 $m = 1$, you can use the symbol `ADJSEVSUM_` (alias `CUMADJSEV_`).

- `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 125 (step 2) or the section “Simulation with Regressors and No External Counts” on page 126 (step 3). PROC CCDM supplies this value to your program.

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 SAS Visual Data Management and Utility 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 126 or the section “Simulation with External Counts” on page 128, except that after making each of the \( N \) severity draws, PROC CCDM executes your severity adjustment program to compute the adjusted severity symbols \( (X^d_j) \). 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 135.

**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. Let \( A \) denote the set of values of the adjustment variables. Then, the form of the adjustment function \( f^d \) that computes the adjusted severity value becomes

\[
X^d_j = f^d (X_j, S^d_{j-1}, A)
\]

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 135.

**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 \((k = 1, \ldots, K)\) in the current iteration of the simulation process. Each value of \( N_k \) is adjusted to conform to the upper limit of either 1,000 or the value that you specify in the MAXCOUNTDRAW= option. The total number of severity draws that need to be
Your severity adjustment program must implement all

\[ S^d = \sum_{k=1}^{K} \sum_{j=1}^{N_k} X^d_{k,j} \]

where \( X^d_{k,j} \) is an adjusted severity value of the \( d \)th symbol for the \( k \)th entity in the \( j \)th draw (\( j = 1, \ldots, N_k \)).

The form of the adjustment function \( f^d \) that computes \( X^d_{k,j} \) is

\[ X^d_{k,j} = f^d(X_{k,j}, S_{k,j-1}, S^d_{k,j-1}, S_{n-1}, S^d_{n-1}, A) \]

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 \( S^d_{k,j-1} = \sum_{l=1}^{j-1} X^d_{k,l} \) are the aggregate unadjusted loss and the aggregate adjusted loss, respectively, for the \( k \)th entity before \( X_{k,j} \) is generated. The index \( n \) (\( n = 1, \ldots, N \)) keeps track of the total number of severity draws, across all entities, that are made before \( X_{k,j} \) is generated. So \( S_{n-1} = \sum_{l=1}^{n-1} X_l \) and \( S^d_{n-1} = \sum_{l=1}^{n-1} X^d_l \) 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 \( S^d_{n-1} \) 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 \( d \), \( S^d_0 = 0 \); for all values of \( k \), \( S^d_{k,0} = 0 \); and for all combinations of \( k \) and \( d \), \( S^d_{k,0} = 0 \).

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 CCDM 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 \( X^d_{k,j} \).

In your severity adjustment program, you can use the following two additional placeholder keywords:

**_SEVSUMFOROBS_**
**_CUMSEVSFOROBS_**

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_**
**_CUMADJSEVSFOROBSd_**

indicates the placeholder for \( S^d_{k,j-1} \), 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 _CUMADJSEVSFOROBS_).

The previously described placeholder symbols _SEVSUM_ and _ADJSEVSUM_ represent \( S_{n-1} \) and \( S^d_{n-1} \), 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.

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 137.
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 126 to illustrate the simulation of aggregate adjusted loss.

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>deductible</th>
<th>limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>250</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>500</td>
<td>3000</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>5000</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>2000</td>
</tr>
</tbody>
</table>

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:

```sas
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:

```sas
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:
Chapter 5: The CCDM Procedure

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>deductible</th>
<th>limit</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>500</td>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>2000</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>500</td>
<td>5000</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>2000</td>
<td>0</td>
</tr>
</tbody>
</table>

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 \( \text{Age}=30, \text{Gender}=2, \text{CarType}=1 \). 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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>deductible</th>
<th>limit</th>
<th>count</th>
<th><em>sev</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>2</td>
<td>350</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>500</td>
<td>3000</td>
<td>1</td>
<td>4500</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>2000</td>
<td>2</td>
<td>700</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>5000</td>
<td>3</td>
<td>950</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>2000</td>
<td>0</td>
<td>2100</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>deductible</th>
<th>limit</th>
<th><em>sev</em></th>
<th>adjsevsumforobs_</th>
<th>amountPaid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>350</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>2100</td>
<td>100</td>
<td>1850</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>3000</td>
<td>4500</td>
<td>0</td>
<td>3000</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>2000</td>
<td>700</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>2000</td>
<td>4300</td>
<td>600</td>
<td>1400</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>600</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>1500</td>
<td>400</td>
<td>1300</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>950</td>
<td>1700</td>
<td>750</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>deductible</th>
<th>limit</th>
<th><em>sev</em></th>
<th><em>adjsevsumforobs</em></th>
<th>amountPaid</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>600</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>2000</td>
<td>4300</td>
<td>0</td>
<td>2000</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>350</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>2000</td>
<td>700</td>
<td>2000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>950</td>
<td>400</td>
<td>750</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>5000</td>
<td>2100</td>
<td>100</td>
<td>1850</td>
</tr>
<tr>
<td>2</td>
<td>500</td>
<td>3000</td>
<td>4500</td>
<td>0</td>
<td>3000</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>5000</td>
<td>1500</td>
<td>1150</td>
<td>1300</td>
</tr>
</tbody>
</table>

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 137 and “Illustration of the Need to Randomize the Order of Severity Draws” on page 138.

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. If entity \( k \) generates \( N_k \) loss events, where \( N_k \) is adjusted to conform to the upper limit of either 1,000 or the value that you specify in 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. However, 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.

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= set if you do not specify the BY statement.

The randomization process over \( K \) observations is implemented as follows. First, one of the \( K \) observations is chosen at random, and one severity value is drawn from the severity distribution implied by that observation; then another observation is chosen at random, and one severity value is drawn from its implied severity distribution; and so on. In each step, the total number of events that are simulated for the selected observation \( k \) is incremented by 1. When all \( N_k \) events for an observation \( k \) are simulated, observation \( k \) is retired and the process continues with the remaining observations until a total of \( N \) severity draws are made. Let \( k(j) \)
denote a function that implements this randomization by returning an observation \( k \) \((k = 1, \ldots, K)\) for the \( j \)th draw \((j = 1, \ldots, N)\). The aggregate loss computation can then be formally written as

\[
S = \sum_{j=1}^{N} X_{k(j)}
\]

where \( X_{k(j)} \) denotes the severity value that is drawn by using observation \( k(j) \).

If you do not specify a scale regression model for severity, then all severity values are drawn from the same severity distribution. However, if you specify a scale regression model for severity, then the severity draw is made from the severity distribution that is determined by the values of regressors in observation \( k \). In particular, the scale parameter of the distribution depends on the values of regressors in observation \( k \). If \( R(l) \) denotes the scale regression model for observation \( l \) and \( X_{R(l)} \) denotes the severity value drawn from scale regression model \( R(l) \), then the aggregate loss computation can be formally written as

\[
S = \sum_{j=1}^{N} X_{R(k(j))}
\]

This 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, 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 135, 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 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 135. 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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>deductible</th>
<th>count</th>
<th><em>sev</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>2</td>
<td>1</td>
<td>250</td>
<td>2</td>
<td>350</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2100</td>
<td></td>
<td>2100</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1</td>
<td>2</td>
<td>500</td>
<td>1</td>
<td>4500</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>2</td>
<td>700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4300</td>
<td></td>
<td>4300</td>
</tr>
<tr>
<td>4</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td>3</td>
<td>600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1500</td>
<td></td>
<td>950</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>200</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>deductible</th>
<th><em>sev</em></th>
<th><em>adjsevsum</em></th>
<th>penaltyFactor</th>
<th>amountPaid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>250</td>
<td>350</td>
<td>1</td>
<td>1.0067</td>
<td>1848.32</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>2100</td>
<td>1948.32</td>
<td>1.1493</td>
<td>3925.36</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>700</td>
<td>5873.68</td>
<td>1.6436</td>
<td>535.64</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>4300</td>
<td>6409.32</td>
<td>1.7461</td>
<td>4125.39</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>600</td>
<td>10534.72</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>1500</td>
<td>10534.72</td>
<td>3</td>
<td>900</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>950</td>
<td>11434.72</td>
<td>3</td>
<td>350</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Obs</th>
<th>deductible</th>
<th><em>sev</em></th>
<th><em>adjsevsum</em></th>
<th>penaltyFactor</th>
<th>amountPaid</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>500</td>
<td>4500</td>
<td>0</td>
<td>1.3636</td>
<td>4000</td>
</tr>
<tr>
<td>1</td>
<td>250</td>
<td>350</td>
<td>4000</td>
<td>1.3648</td>
<td>563.52</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>700</td>
<td>4009.09</td>
<td>1.5361</td>
<td>1192.78</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>950</td>
<td>4572.61</td>
<td>2.1708</td>
<td>165.83</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>1500</td>
<td>5234.91</td>
<td>2.2242</td>
<td>4077.58</td>
</tr>
</tbody>
</table>

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 137.
Chapter 5: The CCDM Procedure

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 CCDM statement to specify the number of perturbed samples to be generated. PROC CCDM 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 CCDM 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 CCDM assumes that the joint probability distribution of the parameter estimates is a multivariate normal distribution, \( N(\mu, \Sigma) \), where the mean vector \( \mu \) is the set of point parameter estimates and \( \Sigma \) 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 CCDM assumes that each parameter has a univariate normal distribution, \( N(\mu, \sigma^2) \), where \( \mu \) is the point estimate of the parameter and \( \sigma \) is the standard error of the parameter estimate.

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.

For the frequency model, you must use the CNTSELECT procedure to create the COUNTSTORE= item store, which always contains the point estimates, standard errors, and variance-covariance matrix of the parameters.

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.
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 CCDM 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 CCDM 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 \( \sqrt{\hat{\beta}_1} = \frac{\mu_3 \mu_2^{-\frac{3}{2}}}{n} \), where \( \mu_2 \) and \( \mu_3 \) 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 5.2, depending on the value that you specify in the VARDEF= option.

**Table 5.2  Formulas for Kurtosis**

<table>
<thead>
<tr>
<th>VARDEF= Value</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF (default)</td>
<td>( \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)} )</td>
</tr>
<tr>
<td>N</td>
<td>( \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \bar{y}}{s} \right)^4 - 3 )</td>
</tr>
</tbody>
</table>

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 \( \hat{\beta}_2 - 3 \), where \( \hat{\beta}_2 = \frac{\mu_4}{\mu_2^2} \) and \( \mu_4 \) 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 the EXTERNAL-COUNTS 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 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 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 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_”.
- `_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_.

\(<\text{adjusted sample variable 1}> \ldots <\text{adjusted sample variable } D>\) 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 and there is only one adjusted severity symbol, 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. The value of this variable identifies the perturbed sample. A value of 0 indicates an unperturbed sample.

**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 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 there is only one adjusted severity symbol and if you do not specify the `ADJSAMPLEVAR=` option, then the value is the default value, `_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 5.3 relates the `PRINT=` options to ODS tables.

<table>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompoundInfo</td>
<td>Compound distribution information</td>
<td>Default</td>
</tr>
<tr>
<td>DataSummary</td>
<td>Input data summary</td>
<td>Default</td>
</tr>
<tr>
<td>Percentiles</td>
<td>Percentiles of the aggregate loss sample</td>
<td><code>PRINT=PERCENTILES</code></td>
</tr>
<tr>
<td>PerturbedPctlSummary</td>
<td>Perturbation analysis of percentiles</td>
<td><code>PRINT=PERTURBSUMMARY</code> and <code>NPERTURBEDSAMPLES &gt; 0</code></td>
</tr>
<tr>
<td>PerturbedSummary</td>
<td>Perturbation analysis of summary statistics</td>
<td><code>PRINT=PERTURBSUMMARY</code> and <code>NPERTURBEDSAMPLES &gt; 0</code></td>
</tr>
<tr>
<td>SummaryStatistics</td>
<td>Summary statistics of the aggregate loss sample</td>
<td><code>PRINT=SUMMARYSTATISTICS</code></td>
</tr>
<tr>
<td>Timing</td>
<td>Timing information for various computational stages of the procedure</td>
<td><code>PRINT=ALL</code></td>
</tr>
</tbody>
</table>

**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
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.

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 98 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:

$$
Y = \begin{cases} 
0 & X \leq d \\
\frac{d' - d}{d'} X & d < X \leq d' \\
X & d' < X \leq u \\
u & X > u 
\end{cases}
$$

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 98 to include three additional variables for each policyholder: LowDeductible to record $d$, HighDeductible to record $d'$, and Limit to record $u$.

```sas
/* 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;
   lowDeductible = int(rand('NORMAL', 50, 400));
   highDeductible = int(rand('NORMAL', 400, 1500));
   limit = int(rand('NORMAL', 15000, 50000));
   annualLimit = int(rand('NORMAL', 25000, 100000));
%
```
Example 5.1: Estimating the Probability Distribution of Insurance Payments

```r
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

<table>
<thead>
<tr>
<th>policyholderID</th>
<th>age</th>
<th>gender</th>
<th>carType</th>
<th>annualMiles</th>
<th>education</th>
<th>carSafety</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.18</td>
<td>2</td>
<td>1</td>
<td>2.2948</td>
<td>3</td>
<td>0.99532</td>
<td>1.59870</td>
</tr>
<tr>
<td>2</td>
<td>0.66</td>
<td>2</td>
<td>2</td>
<td>2.8148</td>
<td>1</td>
<td>0.05625</td>
<td>0.67539</td>
</tr>
<tr>
<td>3</td>
<td>0.82</td>
<td>1</td>
<td>2</td>
<td>1.6130</td>
<td>2</td>
<td>0.84146</td>
<td>1.05940</td>
</tr>
<tr>
<td>4</td>
<td>0.44</td>
<td>1</td>
<td>1</td>
<td>1.2280</td>
<td>3</td>
<td>0.14324</td>
<td>0.24110</td>
</tr>
<tr>
<td>5</td>
<td>0.44</td>
<td>1</td>
<td>1</td>
<td>0.9670</td>
<td>2</td>
<td>0.08656</td>
<td>0.65979</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lowDeductible</th>
<th>highDeductible</th>
<th>limit</th>
<th>annualLimit</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>1400</td>
<td>7500</td>
<td>10000</td>
</tr>
<tr>
<td>300</td>
<td>1300</td>
<td>2500</td>
<td>20000</td>
</tr>
<tr>
<td>100</td>
<td>1100</td>
<td>5000</td>
<td>10000</td>
</tr>
<tr>
<td>300</td>
<td>800</td>
<td>5000</td>
<td>20000</td>
</tr>
<tr>
<td>100</td>
<td>1100</td>
<td>5000</td>
<td>20000</td>
</tr>
</tbody>
</table>

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:

```r
/* 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 *
```
Chapter 5: The CCDM Procedure

\[
\frac{\text{\_sev\_lowDeductible}}{(\text{highDeductible} - \text{lowDeductible})};
\]

else

\[
\text{amountPaid} = \text{MIN}(\text{\_sev\_}, \text{limit}); /* imposes per-loss payment limit */
\]

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

<table>
<thead>
<tr>
<th>Severity Model</th>
<th>Lognormal Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale Model Effects</td>
<td>carSafety, carType, income</td>
</tr>
<tr>
<td>Count Model</td>
<td>NegBin(p=2) Model in Item Store COUNTREGMODEL</td>
</tr>
</tbody>
</table>

---

### The CCDM Procedure

**Severity Model:** Logn  
**Count Model:** NegBin(p=2)

#### Sample Summary Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5903.9</td>
</tr>
<tr>
<td>Median</td>
<td>4746.5</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>4961.4</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>5133.0</td>
</tr>
<tr>
<td>Variance</td>
<td>24615924.9</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Skewness</td>
<td>4.07140</td>
</tr>
<tr>
<td>Maximum</td>
<td>138633.3</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>62.07742</td>
</tr>
<tr>
<td>Sample Size</td>
<td>10000</td>
</tr>
</tbody>
</table>

---

### The CCDM Procedure

**Severity Model:** Logn  
**Count Model:** NegBin(p=2)

#### Adjusted Sample Summary Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>4352.4</td>
</tr>
<tr>
<td>Median</td>
<td>3770.4</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>3152.0</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>4086.7</td>
</tr>
<tr>
<td>Variance</td>
<td>9935181.2</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Skewness</td>
<td>1.12775</td>
</tr>
<tr>
<td>Maximum</td>
<td>31285.3</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>1.92631</td>
</tr>
<tr>
<td>Sample Size</td>
<td>10000</td>
</tr>
</tbody>
</table>
Example 5.1: Estimating the Probability Distribution of Insurance Payments

The perturbation summary of the distribution of \( \text{AmountPaid} \) is shown in Output 5.1.3. It shows that you can expect to pay a median of 3,795 ± 285 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 15,635 ± 932 in the worst case.

Output 5.1.3 Perturbation Summary of the Total Amount Paid

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Estimate</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.41427</td>
<td>1.88292</td>
</tr>
<tr>
<td>5</td>
<td>386.82844</td>
<td>59.20171</td>
</tr>
<tr>
<td>25</td>
<td>1995.1</td>
<td>194.17396</td>
</tr>
<tr>
<td>50</td>
<td>3794.9</td>
<td>284.61679</td>
</tr>
<tr>
<td>75</td>
<td>6150.2</td>
<td>404.83050</td>
</tr>
<tr>
<td>95</td>
<td>10448.4</td>
<td>642.52922</td>
</tr>
<tr>
<td>99</td>
<td>14145.4</td>
<td>816.03986</td>
</tr>
<tr>
<td>99.5</td>
<td>15635.2</td>
<td>931.66304</td>
</tr>
</tbody>
</table>

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 \( \text{AnnualLimit} \) 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:

```plaintext
/* 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.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;

/* impose policyholder's annual limit */
amountPaid = MIN(amountPaid, MAX(0,annualLimit - _adjsevsumforobs_));

/* impose group's annual limit */
amountPaid = MIN(amountPaid, MAX(0,15000 - _adjsevsu_)));
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 $3,795 \pm 285$ to $3,730 \pm 280$, and reduced the worst-case payment (99.5th percentile) from $15,635 \pm 932$ to $14,767 \pm 300$.

**Output 5.1.4** Perturbation Summary of the Total Amount Paid after Modified Policy Provisions

<table>
<thead>
<tr>
<th>The CCDM Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Severity Model:</strong> Logn</td>
</tr>
<tr>
<td><strong>Count Model:</strong> NegBin(p=2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adjusted Sample Percentile Perturbation Analysis</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentile</td>
<td>Estimate</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>1</td>
<td>0.38457</td>
</tr>
<tr>
<td>5</td>
<td>375.82998</td>
</tr>
<tr>
<td>25</td>
<td>1952.0</td>
</tr>
<tr>
<td>50</td>
<td>3729.9</td>
</tr>
<tr>
<td>75</td>
<td>6048.0</td>
</tr>
<tr>
<td>95</td>
<td>10277.8</td>
</tr>
<tr>
<td>99</td>
<td>13715.7</td>
</tr>
<tr>
<td>99.5</td>
<td>14767.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Perturbed Samples</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of Each Sample</td>
<td>10000</td>
</tr>
</tbody>
</table>

**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 128.

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
Example 5.2: Using Externally Simulated Count Data

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 KRIIs that you use in the count regression model of the commercial banking line, and let CorpKRI1, RbKRI1, and RbKRI2 be the KRIIs that you use in the count regression model of the retail banking line. Examples of corporate-level KRIIs (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 KRIIs 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 KRIIs 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 KRIIs, CorpKRI1 and CbKRI2. Let the severity of each loss event in the retail banking business be dependent on three KRIIs, CorpKRI2, RbKRI1, and RbKRI2. Note that for each line of business, the set of KRIIs that are used for the severity model is different from the set of KRIIs 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 KRIIs 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:

```plaintext
/* 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 */
   end;
   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;
Example 5.2: Using Externally Simulated Count Data

```plaintext
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;
run;
Example 5.2: Using Externally Simulated Count Data

/* 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**

<table>
<thead>
<tr>
<th>Input Data Summary</th>
<th>line=CommercialBanking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>LOSSCOUNTS1</td>
</tr>
<tr>
<td>Observations</td>
<td>10000</td>
</tr>
<tr>
<td>Valid Observations</td>
<td>10000</td>
</tr>
<tr>
<td>Replications</td>
<td>10000</td>
</tr>
<tr>
<td>Total Count</td>
<td>19028</td>
</tr>
</tbody>
</table>
Output 5.2.1  continued

The CCDM Procedure

Severity Model: Logn  
Count Model: External

\[ \text{line=} \text{CommercialBanking} \]

<table>
<thead>
<tr>
<th>Sample Summary Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>599.56756</td>
</tr>
<tr>
<td>Median</td>
<td>344.00869</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>786.88364</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>785.04322</td>
</tr>
<tr>
<td>Variance</td>
<td>619185.9</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Skewness</td>
<td>2.80654</td>
</tr>
<tr>
<td>Maximum</td>
<td>11176.8</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>13.58752</td>
</tr>
<tr>
<td>Sample Size</td>
<td>10000</td>
</tr>
</tbody>
</table>

\[ \text{line=} \text{CommercialBanking} \]

<table>
<thead>
<tr>
<th>Sample Percentiles</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>52.25800</td>
</tr>
<tr>
<td>50</td>
<td>344.00869</td>
</tr>
<tr>
<td>75</td>
<td>837.30121</td>
</tr>
<tr>
<td>95</td>
<td>2107.5</td>
</tr>
<tr>
<td>99</td>
<td>3637.3</td>
</tr>
<tr>
<td>99.5</td>
<td>4428.4</td>
</tr>
</tbody>
</table>

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

\[ \text{line=} \text{RetailBanking} \]

<table>
<thead>
<tr>
<th>Sample Percentiles</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>70.46309</td>
</tr>
<tr>
<td>75</td>
<td>142.15990</td>
</tr>
<tr>
<td>95</td>
<td>271.76555</td>
</tr>
<tr>
<td>99</td>
<td>378.32158</td>
</tr>
<tr>
<td>99.5</td>
<td>437.12308</td>
</tr>
</tbody>
</table>

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:

``` Sas
/* 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 RepId:

``` Sas
/* 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;
    alpha = _Alpha;
  end;
```
\[
\theta = \frac{1}{\alpha};
\]

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;
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.

```plaintext
/* 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 (RepId=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 (RepId=2), all operating environments incur at least one loss event. For the first replication (RepId=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

<table>
<thead>
<tr>
<th>line</th>
<th>opEnvl</th>
<th>corpKRI1</th>
<th>corpKRI2</th>
<th>cbKRI2</th>
<th>rbKRI1</th>
<th>rbKRI3</th>
<th>repid</th>
<th>numloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommercialBanking</td>
<td>1</td>
<td>0.45224</td>
<td>0.40661</td>
<td>-0.33680</td>
<td>-1.08692</td>
<td>-2.20557</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>2</td>
<td>-0.03799</td>
<td>0.98670</td>
<td>-0.03752</td>
<td>1.94589</td>
<td>1.22456</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>3</td>
<td>-0.29120</td>
<td>-0.45239</td>
<td>0.98855</td>
<td>-0.37208</td>
<td>-1.51534</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>4</td>
<td>0.87499</td>
<td>-0.67812</td>
<td>-0.04839</td>
<td>-1.44881</td>
<td>0.78221</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>1</td>
<td>0.45224</td>
<td>0.40661</td>
<td>-0.33680</td>
<td>-1.08692</td>
<td>-2.20557</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>2</td>
<td>-0.29120</td>
<td>-0.45239</td>
<td>0.98855</td>
<td>-0.37208</td>
<td>-1.51534</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>3</td>
<td>0.87499</td>
<td>-0.67812</td>
<td>-0.04839</td>
<td>-1.44881</td>
<td>0.78221</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>1</td>
<td>0.45224</td>
<td>0.40661</td>
<td>-0.33680</td>
<td>-1.08692</td>
<td>-2.20557</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>2</td>
<td>-0.03799</td>
<td>0.98670</td>
<td>-0.03752</td>
<td>1.94589</td>
<td>1.22456</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>3</td>
<td>-0.29120</td>
<td>-0.45239</td>
<td>0.98855</td>
<td>-0.37208</td>
<td>-1.51534</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>4</td>
<td>0.87499</td>
<td>-0.67812</td>
<td>-0.04839</td>
<td>-1.44881</td>
<td>0.78221</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>1</td>
<td>0.45224</td>
<td>0.40661</td>
<td>-0.33680</td>
<td>-1.08692</td>
<td>-2.20557</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>2</td>
<td>-0.03799</td>
<td>0.98670</td>
<td>-0.03752</td>
<td>1.94589</td>
<td>1.22456</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>3</td>
<td>-0.29120</td>
<td>-0.45239</td>
<td>0.98855</td>
<td>-0.37208</td>
<td>-1.51534</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>4</td>
<td>0.87499</td>
<td>-0.67812</td>
<td>-0.04839</td>
<td>-1.44881</td>
<td>0.78221</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
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 RepId in the ID= option of the EXTERNALCOUNTS statement:

```latex
/* 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 RepId, 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 128.

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

<table>
<thead>
<tr>
<th>The CCDM Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>line=CommercialBanking</td>
</tr>
</tbody>
</table>

### Input Data Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>LOSSCOUNTS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
<td>40000</td>
</tr>
<tr>
<td>Valid Observations</td>
<td>40000</td>
</tr>
<tr>
<td>Replications</td>
<td>10000</td>
</tr>
<tr>
<td>Total Count</td>
<td>145721</td>
</tr>
</tbody>
</table>

### Sample Percentiles

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>771.75856</td>
</tr>
<tr>
<td>5</td>
<td>1507.8</td>
</tr>
<tr>
<td>25</td>
<td>3097.6</td>
</tr>
<tr>
<td>50</td>
<td>4816.6</td>
</tr>
<tr>
<td>75</td>
<td>7000.8</td>
</tr>
<tr>
<td>95</td>
<td>11346.5</td>
</tr>
<tr>
<td>99</td>
<td>15740.3</td>
</tr>
<tr>
<td>99.5</td>
<td>17520.5</td>
</tr>
</tbody>
</table>

| Percentile Method = 5 |
References


# Chapter 6
## The CCOPULA Procedure

<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: CCOPULA Procedure</td>
<td>166</td>
</tr>
<tr>
<td>PROC CCOPULA Features</td>
<td>166</td>
</tr>
<tr>
<td>PROC CCOPULA Compared with Other SAS Procedures</td>
<td>166</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>167</td>
</tr>
<tr>
<td>Getting Started: CCOPULA Procedure</td>
<td>168</td>
</tr>
<tr>
<td>Syntax: CCOPULA Procedure</td>
<td>170</td>
</tr>
<tr>
<td>Functional Summary</td>
<td>170</td>
</tr>
<tr>
<td>PROC CCOPULA Statement</td>
<td>172</td>
</tr>
<tr>
<td>BY Statement</td>
<td>172</td>
</tr>
<tr>
<td>DEFINE Statement</td>
<td>172</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>173</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>174</td>
</tr>
<tr>
<td>FIT Statement</td>
<td>175</td>
</tr>
<tr>
<td>SIMULATE Statement</td>
<td>177</td>
</tr>
<tr>
<td>VAR Statement</td>
<td>178</td>
</tr>
<tr>
<td>VIEWSTORE Statement</td>
<td>178</td>
</tr>
<tr>
<td>Details: CCOPULA Procedure</td>
<td>179</td>
</tr>
<tr>
<td>Sklar’s Theorem</td>
<td>179</td>
</tr>
<tr>
<td>Dependence Measures</td>
<td>180</td>
</tr>
<tr>
<td>Data Processing and Fitting</td>
<td>181</td>
</tr>
<tr>
<td>Normal Copula</td>
<td>181</td>
</tr>
<tr>
<td>Student’s $t$ copula</td>
<td>182</td>
</tr>
<tr>
<td>Archimedean Copulas</td>
<td>184</td>
</tr>
<tr>
<td>Canonical Maximum Likelihood Estimation (CMLE)</td>
<td>190</td>
</tr>
<tr>
<td>Exact Maximum Likelihood Estimation (MLE)</td>
<td>191</td>
</tr>
<tr>
<td>Calibration Estimation</td>
<td>191</td>
</tr>
<tr>
<td>Nonlinear Optimization Options</td>
<td>191</td>
</tr>
<tr>
<td>Displayed Output</td>
<td>191</td>
</tr>
<tr>
<td>OUTPSUEDO= and OUTUNIFORM= Data Sets</td>
<td>194</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>194</td>
</tr>
<tr>
<td>Examples: CCOPULA Procedure</td>
<td>195</td>
</tr>
<tr>
<td>Example 6.1: Simulating Default Times</td>
<td>195</td>
</tr>
<tr>
<td>References</td>
<td>198</td>
</tr>
</tbody>
</table>
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 given sample data set. 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
- 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.”
Chapter 6: The CCOPULA Procedure

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](image1)

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.

```sas
/* 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 set, the type of model, and the correlations matrix.

![Figure 6.2 Copula Estimation: Fit Summary and Correlations Matrices](image2)

The CCOPULA Procedure

<table>
<thead>
<tr>
<th>Model Fit Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Copula Type</td>
</tr>
<tr>
<td>Fit Method</td>
</tr>
<tr>
<td>Marginals</td>
</tr>
</tbody>
</table>

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 SAS data set 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.

```sas
/* 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 SAS data set, 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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Pearson Correlations Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret_ibm</td>
<td>1.0000 0.6232 0.5294 0.4725 0.4902</td>
</tr>
<tr>
<td>ret_msft</td>
<td>0.6232 1.0000 0.5229 0.5015 0.4567</td>
</tr>
<tr>
<td>ret_bp</td>
<td>0.5294 0.5229 1.0000 0.3980 0.4378</td>
</tr>
<tr>
<td>ret_ko</td>
<td>0.4725 0.5015 0.3980 1.0000 0.5283</td>
</tr>
<tr>
<td>ret_duk</td>
<td>0.4902 0.4567 0.4378 0.5283 1.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Kendall Correlations Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret_ibm</td>
<td>1.0000 0.4283 0.3551 0.3133 0.3261</td>
</tr>
<tr>
<td>ret_msft</td>
<td>0.4283 1.0000 0.3503 0.3344 0.3020</td>
</tr>
<tr>
<td>ret_bp</td>
<td>0.3551 0.3503 1.0000 0.2606 0.2885</td>
</tr>
<tr>
<td>ret_ko</td>
<td>0.3133 0.3344 0.2606 1.0000 0.3544</td>
</tr>
<tr>
<td>ret_duk</td>
<td>0.3261 0.3020 0.2885 0.3544 1.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Spearman Correlations Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>ret_ibm</td>
<td>1.0000 0.6052 0.5116 0.4555 0.4729</td>
</tr>
<tr>
<td>ret_msft</td>
<td>0.6052 1.0000 0.5052 0.4841 0.4400</td>
</tr>
<tr>
<td>ret_bp</td>
<td>0.5116 0.5052 1.0000 0.3826 0.4215</td>
</tr>
<tr>
<td>ret_ko</td>
<td>0.4555 0.4841 0.3826 1.0000 0.5106</td>
</tr>
<tr>
<td>ret_duk</td>
<td>0.4729 0.4400 0.4215 0.5106 1.0000</td>
</tr>
</tbody>
</table>
Chapter 6: The CCOPULA Procedure

/* 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](image)

The simulated data are contained in the new SAS data set, 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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Options</td>
<td>PROC CCOPULA</td>
<td>DATA=</td>
</tr>
<tr>
<td>Description</td>
<td>Statement</td>
<td>Option</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>Specifies the input data set that contains the correlations matrix for elliptical copulas</td>
<td>DEFINE</td>
<td>CORR=</td>
</tr>
<tr>
<td>Specifies the input data set that contains the correlations matrix defined in Kendall’s tau for elliptical copulas</td>
<td>DEFINE</td>
<td>KENDALL=</td>
</tr>
<tr>
<td>Specifies the input data set that contains the correlations matrix defined in Spearman’s rho for elliptical copulas</td>
<td>DEFINE</td>
<td>SPEARMAN=</td>
</tr>
<tr>
<td><strong>Copula Fitting Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the method to use</td>
<td>FIT</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies the input data set that contains the correlations matrix for elliptical copulas</td>
<td>FIT</td>
<td>CORR=</td>
</tr>
<tr>
<td>Specifies the parameter value for Archimedean copulas</td>
<td>FIT</td>
<td>THETA=</td>
</tr>
<tr>
<td>Specifies the degrees of freedom for $t$ copulas</td>
<td>FIT</td>
<td>DF=</td>
</tr>
<tr>
<td><strong>Declaring the Role of Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the names of the variables to use in copula fitting or in simulation</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td><strong>Options to Control the Optimization Process</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selects the iterative minimization method to use</td>
<td>FIT</td>
<td>OPTMETHOD=</td>
</tr>
<tr>
<td>Specifies the maximum number of iterations for the optimization when you are fitting a $t$ or Archimedean copula and METHOD=MLE</td>
<td>FIT</td>
<td>MAXITER=</td>
</tr>
<tr>
<td>Sets initial values for parameters</td>
<td>FIT</td>
<td>INIT=</td>
</tr>
<tr>
<td><strong>Copula Estimation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the marginal distribution of the individual variables</td>
<td>FIT</td>
<td>MARGINALS=</td>
</tr>
<tr>
<td><strong>Copula Simulation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the random sample size</td>
<td>SIMULATE</td>
<td>NDRAWS=</td>
</tr>
<tr>
<td>Specifies the random number generator seed</td>
<td>SIMULATE</td>
<td>SEED=</td>
</tr>
<tr>
<td><strong>Output Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the output data set to contain the normalized pseudo-data used for model fitting</td>
<td>FIT</td>
<td>OUTPSEUDO=</td>
</tr>
<tr>
<td>Specifies the output item store to preserve the properties of the copula that is fit and the results of fitting the copula</td>
<td>FIT</td>
<td>STORE=</td>
</tr>
<tr>
<td>Specifies the output data set to contain the random samples from the simulation with uniform marginal distribution</td>
<td>SIMULATE</td>
<td>OUTUNIFORM=</td>
</tr>
</tbody>
</table>
Chapter 6: The CCOPULA Procedure

**Description Statement Option**

- 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 Set Options**

- DATA=SAS-data-set
  
  specifies the input data set to use to estimate parameters for the FIT statement. When the procedure is used for simulation only, the input data set 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 set and are excluded from the model fitting. You must provide a VAR statement as well. 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 with the RESTORE= option. The item store must contain model fitting results for the same BY-groups 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 179:
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

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 194. 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 By-
group1.Summary_SELECTIONSELECTSUM.

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 subsetted 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 194. 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 t 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:
Chapter 6: The CCOPULA Procedure

**CORR=SAS-data-set**
specifies the data set that contains the correlations matrix to use for elliptical copulas. 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=SAS-data-set**
specifies the data set that contains the correlations matrix defined in Kendall’s tau. 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=SAS-data-set**
specifies the data set that contains the correlations matrix defined in Spearman’s rho. 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 set 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=UNIFORM | EMPIRICAL**
specifies the marginal distribution of the individual variables. You can specify the following values:

- **EMPIRICAL** uses the marginal empirical CDF to transform the data and uses the transformed data to fit the copula.
- **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 \( \text{METHOD=CAL} \), 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 \( \text{METHOD=MLE} \) is supported and \( \text{METHOD=CAL} \) is ignored. By default for all copula types, \( \text{METHOD=MLE} \).

**OUTPSEUDO=SAS-data-set**

specifies the output data set for saving the pseudo-samples with uniform marginal distributions. The pseudo-samples are obtained by transforming the individual variables of the original data by using the empirical cumulative distribution functions (CDFs). The data set is not created if you omit this option.

**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, \( \text{NDRAWS}=100 \).
OUTUNIFORM=SAS-data-set
specifies the output data set to contain the result of the simulation in uniform margins. The data are not created if you do not specify this option.

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 provide the seed, a random number is used as the seed.

TOLERANCE=value
specifies the tolerance allowed for the simulation.

VAR Statement

VAR variables;

The VAR statement specifies the variable names in the input data set that is specified by the DATA= option in the PROC CCOPULA statement. The subset of variables in the data set is used for the copula models in the FIT statement. If there is no input data set, 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 \( C(u_1, \ldots, u_m) \).

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 \( F_j, j = 1, \ldots, m \), be the marginal distributions. Then
there exists a copula \( C : [0, 1]^m \to [0, 1] \) such that

\[
F(x_1, \ldots, x_m) = C(F_1(x_1), \ldots, F_m(x_m))
\]

for all \( x_1, \ldots, x_m \) in \([-\infty, \infty]\). Moreover, if the margins are continuous, then \( C \) is unique; otherwise \( C \)
is uniquely determined on \( \text{Ran} F_1 \times \cdots \times \text{Ran} F_m \), 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 \( F_1, \ldots, F_m \) are univariate distribution functions,
then the multivariate function that is defined in the preceding equation is a joint distribution function
with marginal distributions \( F_j, j = 1, \ldots, m \).
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 $(X_1, X_2)$:

$$\rho_r = E[\text{sign}(X_1 - \tilde{X}_1)(X_2 - \tilde{X}_2)]$$

The sign function (sometimes written as sgn) is defined as

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

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
  $$\lambda_u(X_1, X_2) \equiv \lim_{q \to 1^-} P(X_2 > F_2^{-1}(q)|X_1 > F_1^{-1}(q))$$
  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 a copula can be fit, the data must be transformed 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 transformation involved is the trivial identity transformation. However, if you specify MARGINALS=EMPIRICAL (or if no MARGINALS= option is specified), then the transformation is nontrivial. For relatively small data sets, each observation in the original input data set is transformed into a pseudo-uniform observation by using a simple value-comparison algorithm. Initial estimates of the parameters in \( \tau \) copulas and the Archimedean copulas are derived by computing a Kendall correlations matrix based on the transformed data.

However, for larger data sets, the simple value-comparison algorithm becomes computationally impractical as a means of transforming the original data into pseudo-uniform data. Thus, for larger data sets, a data binning algorithm is used to transform the original data into pseudo-uniform data. After the data have been transformed via binning, 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, for larger data sets, 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, as in the smaller data sets, the initial estimates for the parameters in \( \tau \) copulas and the Archimedean copulas are derived from the Kendall correlations matrix.

Normal Copula

Let \( u_j \sim U(0, 1) \) for \( j = 1, \ldots, m \), where \( U(0, 1) \) represents the uniform distribution on the \([0, 1]\) interval. Let \( \Sigma \) be the correlations matrix, where \( m(m-1)/2 \) parameters satisfy the positive semidefiniteness constraint. The normal copula can be written as

\[
C_\Sigma(u_1, u_2, \ldots, u_m) = \Phi_\Sigma\left(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_m)\right)
\]

where \( \Phi \) is the distribution function of a standard normal random variable and \( \Phi_\Sigma \) is the \( m \)-variate standard normal distribution with mean vector 0 and covariance matrix \( \Sigma \). That is, the distribution \( \Phi_\Sigma \) is \( N_m(0, \Sigma) \).

Simulation

For the normal copula, the input of the simulation is the correlations matrix \( \Sigma \). The normal copula can be simulated by the following steps, in which \( U = (U_1, \ldots, U_m) \) denotes one random draw from the copula:

1. Generate a multivariate normal vector \( Z \sim N(0, \Sigma) \), where \( \Sigma \) is an \( m \)-dimensional correlations matrix.

2. Transform the vector \( Z \) into \( U = (\Phi(Z_1), \ldots, \Phi(Z_m))^T \), where \( \Phi \) 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 \( \tilde{Z} \sim N(0, I) \), then \( L\tilde{Z} \sim N(0, \Sigma) \).
Fitting

To fit a normal copula is to estimate the covariance matrix $\Sigma$ from an input sample data set. Given a random sample $u_i = (u_{i,1}, \ldots, u_{i,m})^T$ where $i = 1, \ldots, n$, the log-likelihood function is

$$
\log L(\Sigma; u_1, \ldots, u_n) = \sum_{t=1}^{n} \log f_{\Sigma}(\Phi^{-1}(u_{t,1}), \ldots, \Phi^{-1}(u_{t,m})) - \sum_{t=1}^{n} \sum_{j=1}^{m} \log \phi(\Phi^{-1}(u_{t,j}))
$$

Here $f_{\Sigma}$ is the joint density of the multivariate normal with mean zero and variance $\Sigma$, and $\phi$ is the univariate density of the standard normal distribution. Note that the second term is not related to the parameters $\Sigma$ and, therefore, can be ignored during the optimization. The restriction that $\Sigma$ is a correlations matrix is very inconvenient, and it is common practice to circumvent this problem by first assuming that $\Sigma$ has the covariance form. Therefore, $\Sigma$ can be estimated by

$$
\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} \xi_i \xi_i^T
$$

where

$$
\xi_i = (\Phi^{-1}(u_{i,1}), \Phi^{-1}(u_{i,2}), \ldots, \Phi^{-1}(u_{i,m}))^T
$$

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:

$$
\Delta(\Sigma) = \text{diag}(\sigma_1^{1/2}, \ldots, \sigma_m^{1/2})
$$

$$
\mathcal{P}(\Sigma) = (\Delta(\Sigma))^{-1} \Sigma (\Delta(\Sigma))^{-1}
$$

Student’s $t$ copula

Let $\Theta = \{(v, \Sigma) : v \in (1, \infty), \Sigma \in \mathbb{R}^{m \times m}\}$, and let $t_v$ be a univariate $t$ distribution with $v$ degrees of freedom.

The Student’s $t$ copula can be written as

$$
C_{\Theta}(u_1, u_2, \ldots, u_m) = t_v, \Sigma (t_v^{-1}(u_1), t_v^{-1}(u_2), \ldots, t_v^{-1}(u_m))
$$

where $t_v, \Sigma$ is the multivariate Student’s $t$ distribution that has a correlations matrix $\Sigma$ with $v$ degrees of freedom.

Simulation

The input parameters for the simulation are $(v, \Sigma)$. The $t$ copula can be simulated by the following steps:

1. Generate a multivariate vector $X \sim t_m(v, 0, \Sigma)$ that follows the centered $t$ distribution with $v$ degrees of freedom and correlations matrix $\Sigma$. 
2. Transform the vector $X$ into $U = (t_v(X_1), \ldots, t_v(X_m))^T$, where $t_v$ is the distribution function of univariate $t$ distribution with $v$ degrees of freedom.

To simulate centered multivariate $t$ random variables, you can use the property that $X \sim t_m(v, 0, \Sigma)$ if $X = \sqrt{v/s}Z$, where $Z \sim N(0, \Sigma)$ and the univariate random variable $s \sim f_v^2$.

**Fitting**

To fit a $t$ copula is to estimate the covariance matrix $\Sigma$ and degrees of freedom $v$ from a given multivariate data set. Given a random sample $u_i = (u_{i,1}, \ldots, u_{i,m})^T, i = 1, \ldots, n$ that has uniform marginal distributions, the log likelihood is

$$
\log L(v, \Sigma; u_{i,1}, \ldots, u_{i,m}) = \sum_{i=1}^{n} \log g_v,\Sigma(t_v^{-1}(u_{i,1}), \ldots, t_v^{-1}(u_{i,m})) - \sum_{i=1}^{n} \sum_{j=1}^{m} \log g_v(t_v^{-1}(u_{i,j}))
$$

where $v$ denotes the degrees of freedom of the $t$ copula, $g_v,\Sigma$ denotes the joint density function of the centered multivariate $t$ distribution with parameters $(v, \Sigma)$, $t_v$ is the distribution function of a univariate $t$ distribution with $v$ degrees of freedom, $\Sigma$ is a correlations matrix, and $g_v$ is the density function of univariate $t$ distribution with $v$ degrees of freedom.

The log likelihood can be maximized with respect to the parameters $\theta = (v, \Sigma) \in \Theta$ using numerical optimization. If you allow the parameters in $\Sigma$ to be such that $\Sigma$ is symmetric and with ones on the diagonal, then the MLE estimate for $\Sigma$ 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 $\Sigma$ by calibration using Kendall’s tau. Then, using this fixed $\Sigma$, the single parameter $v$ can be estimated by MLE. By proposition 5.37 in McNeil, Frey, and Embrechts (2005),

$$
\rho_v(U_i, U_j) = \frac{2}{\pi} \arcsin \rho_{ij}
$$

where $\rho_v$ is the Kendall’s tau and $\rho_{ij}$ is the off-diagonal elements of the correlations matrix $\Sigma$ 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 $\Sigma$ be a symmetric matrix with ones on the diagonal, with off-diagonal entries in $[-1, 1]$. If $\Sigma$ 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 $P$ 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 $\zeta = (\zeta_1, \ldots, \zeta_m)$, with $\zeta_j = t_{ij}^{-1}(u_j)$, and $g$ is the derivative of the log $\Gamma$ function:

$$
   l = \log(c) = -\frac{1}{2} \log(|\Sigma|) + \log(\frac{v + m}{2}) + (m - 1) \log \left( \frac{v}{2} \right) - m \log \left( \frac{v + 1}{2} \right)
   - \frac{v + m}{2} \log(1 + \zeta^T \Sigma^{-1} \zeta / v) + \frac{v + 1}{2} \sum_{j=1}^{m} \log \left( 1 + \frac{\zeta_j^2}{v} \right)
$$

$$
   \frac{\partial l}{\partial v} = \frac{1}{2} g \left( \frac{v + m}{2} \right) + m - \frac{1}{2} g \left( \frac{v}{2} \right) - \frac{m}{2} g \left( \frac{v + 1}{2} \right)
   - \frac{1}{2} \log(1 + \zeta^T \Sigma^{-1} \zeta / v) + \frac{v + m}{2v^2} \frac{\zeta^T \Sigma^{-1} \zeta}{1 + \zeta^T \Sigma^{-1} \zeta / v}
   + \frac{1}{2} \sum_{j=1}^{m} \log(1 + \zeta_j^2 / v) - \frac{v + 1}{2v^2} \sum_{j=1}^{m} \frac{\zeta_j^2}{1 + \zeta_j^2 / v}
   - \frac{(v + m)}{v} \frac{\zeta^T \Sigma^{-1} (d \zeta / dv)}{1 + \zeta^T \Sigma^{-1} \zeta / v} + \frac{v + 1}{v} \sum_{j=1}^{m} \frac{\zeta_j (d \zeta_j / dv)}{1 + \zeta_j^2 / v}
$$

The derivative of the likelihood with respect to the correlations matrix $\Sigma$ follows:

$$
   \frac{\partial l}{\partial \Sigma} = -\frac{1}{2} (\Sigma^{-1})^T + \frac{v + m}{2} \frac{\Sigma^{-T} \zeta^T \Sigma^{-T} / v}{1 + \zeta^T \Sigma^{-1} \zeta / v}
   = -\frac{1}{2} (\Sigma^{-1})^T + \frac{v + m}{2} \frac{\Sigma^{-T} \zeta^T \Sigma^{-T}}{v + \zeta^T \Sigma^{-1} \zeta}
$$

### Archimedean Copulas

#### Overview of Archimedean Copulas

Let function $\phi : [0, 1] \to [0, \infty)$ be a strict Archimedean copula generator function, and suppose that its inverse $\phi^{-1}$ is completely monotonic on $[0, \infty)$. A strict generator is a decreasing function $\phi : [0, 1] \to [0, \infty)$ that satisfies $\phi(0) = \infty$ and $\phi(1) = 0$. A decreasing function $f(t) : [a, b] \to (-\infty, \infty)$ 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:

$$
   C(u_1, u_2, \ldots, u_m) = \phi^{-1}\left( \phi(u_1) + \cdots + \phi(u_m) \right)
$$

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 $\phi(u) = \theta^{-1} \left( u^{-\theta} - 1 \right)$. A Clayton copula is defined as

$$C_\theta(u_1, u_2, \ldots, u_m) = \left( \sum_{i=1}^{m} u_i^{-\theta} - m + 1 \right)^{-1/\theta}$$

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

$$C_\theta(u_1, u_2, \ldots, u_m) = \frac{1}{\theta} \log \left\{ 1 + \prod_{i=1}^{m} \left[ \frac{\exp(-\theta u_i) - 1}{\exp(-\theta) - 1} \right]^{m-1} \right\}$$

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

$$C_\theta(u_1, u_2, \ldots, u_m) = \exp \left\{ - \left[ \sum_{i=1}^{m} (-\log u_i)^\theta \right]^{1/\theta} \right\}$$

where $\theta > 1$.

Simulation

Suppose that the generator of the Archimedean copula is $\phi$. 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 $X_1, \ldots, X_m$.
3. Return $U = (\tilde{F}(-\log(X_1)/V), \ldots, \tilde{F}(-\log(X_m)/V))^T$.

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 $\theta^{-1}$ 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 $St(\theta^{-1}, 1, \gamma, 0)$, 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 $\theta$ and Kendall’s tau is summarized in Table 6.2 for the three Archimedean copulas.

<table>
<thead>
<tr>
<th>Copula Type</th>
<th>$\tau$</th>
<th>Formula for $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clayton</td>
<td>$\theta/(\theta + 2)$</td>
<td>$2\tau/(1 - \tau)$</td>
</tr>
<tr>
<td>Gumbel</td>
<td>$1 - 1/\theta$</td>
<td>$1/(1 - \tau)$</td>
</tr>
<tr>
<td>Frank</td>
<td>$1 - 4\theta^{-1}(1 - D_1(\theta))$</td>
<td>No closed form</td>
</tr>
</tbody>
</table>

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 $\theta$ has no closed form. The numerical algorithm for root finding can be used to invert the function $\tau(\theta)$ to obtain $\theta$ as a function of $\tau$.

Alternatively, you can use the MLE or CMLE method to estimate the parameter $\theta$ given the data $u = \{u_{i,j}\}$ and $i = 1, \ldots, n, j = 1, \ldots, m$. 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):

$$l = n \left[ m \log(\theta) + \log \left( \Gamma \left( \frac{1}{\theta} + m \right) \right) - \log \left( \Gamma \left( \frac{1}{\theta} \right) \right) \right] - (\theta + 1) \sum_{i,j} \log u_{i,j}$$

$$- \left( \frac{1}{\theta} + m \right) \sum_{i} \log \left( \sum_{j} u_{i,j}^{-\theta} - m + 1 \right)$$
Let $g(\cdot)$ be the derivative of $\log(\Gamma(\cdot))$. Then the first-order derivative is

$$
\frac{dl}{d\theta} = n \left[ \frac{m}{\theta} + g \left( \frac{1}{\theta} + m \right) \frac{-1}{\theta^2} - g \left( \frac{1}{\theta} \right) \frac{-1}{\theta^2} \right] \\
- \sum_{i,j} \log(u_{ij}) + \frac{1}{\theta^2} \sum_l \log \left( \sum_j u_{ij}^{-\theta} - m + 1 \right) \\
- \left( \frac{1}{\theta} + m \right) \sum_i - \frac{\sum_j u_{ij}^{-\theta} \log(u_{ij})}{\sum_j u_{ij}^{-\theta} - m + 1}
$$

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 $k = 1, 2, \ldots, m$,

$$
\phi^{-1(k)}(u) = (-1)^k \alpha \exp(-u^\alpha) u^{-k+\alpha} \Psi_{k-1}(u^\alpha)
$$

where $\Psi_{k-1}$ 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

$$
l = \log(c) \\
= \log(f_1) + \log(f_2) + \log(f_3) + \log(f_4) + \log((-1)^m f_5)
$$
Now the first-order derivative of the log density has the decomposition

\[
\frac{dl}{d\alpha} = \frac{1}{c} \frac{dc}{d\alpha} = 4 \sum_{j=1}^{4} \frac{1}{f_j} \frac{df_j}{d\alpha} + d \sum_k \log(-\phi'(u_k)) \frac{d \phi'(u_k)}{d\alpha}
\]

Some of the terms are given by

\[
\begin{align*}
\frac{1}{f_1} \frac{df_1}{d\alpha} &= \frac{1}{\alpha} \\
\frac{1}{f_2} \frac{df_2}{d\alpha} &= -x^\alpha \log(x) - \alpha x^{\alpha-1} \frac{dx}{d\alpha} \\
\frac{1}{f_3} \frac{df_3}{d\alpha} &= \log(x) + (-k + \alpha) x^{-1} \frac{dx}{d\alpha}
\end{align*}
\]

where

\[
\frac{dx}{d\alpha} = \sum \left( -\log u_k \right)^{1/\alpha} \log(-\log u_k) \left( \frac{-1}{\alpha^2} \right)
\]

The last term in the derivative of the \(dl/d\alpha\) is

\[
\begin{align*}
\log(-\phi'(u_k)) &= \log \left( \frac{1}{\alpha} \left( -\log u_k \right)^{1/\alpha-1} \frac{1}{u_k} \right) \\
&= -\log \alpha - \log(u_k) + \left( \frac{1}{\alpha} - 1 \right) \log(-\log(u_k)) \\
\frac{d}{d\alpha} \sum_k \log(-\phi'(u_k)) &= \sum_{k=1}^{m} \left( -\alpha - \frac{1}{\alpha^2} \right) \log(-\log(u_k)) \\
&= -\frac{m}{\alpha} - \frac{1}{\alpha^2} \sum_{k=1}^{m} \log(-\log(u_k))
\end{align*}
\]

Now the only remaining term is \(f_4\), which is related to \(\Psi_{m-1}\). 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 \(\alpha\). For simplicity, use the notation \(a_j(\alpha) \equiv a_j(m-1, \alpha)\). Therefore,

\[
f_4 = \Psi_{m-1}(x^\alpha) = \sum_{j=0}^{m-1} a_j(\alpha) x^{j \alpha}
\]
\[
\frac{df_4}{d\alpha} = \frac{d\Psi_{m-1}(x^\alpha)}{d\alpha} \\
= \sum_{j=0}^{m-1} \left[ \frac{da_j(\alpha)}{d\alpha} x^{j\alpha} + a_j(\alpha) x^{j\alpha} \log(x) + a_j(\alpha) (j\alpha) x^{j\alpha-1} \frac{dx}{d\alpha} \right]
\]

**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

\[
c = \varphi^{-1(m)}(x) \prod_k \varphi'(u_k)
\]

\[
= -\frac{1}{\theta} \Psi_{m-1} \left( \frac{1}{1 + e^{-x}(e^{-\theta} - 1)} \right) \prod_k \varphi'(u_k)
\]

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

\[
\frac{1}{\varphi'(u_k)} \frac{d\varphi'(u_k)}{d\theta} = \frac{1}{\theta(1 - e^{\theta u_k})} \left[ 1 - e^{\theta u_k} + \theta u_k e^{\theta u_k} \right]
\]

The function \( \Psi_{m-1} \) satisfies a recursive relation

\[
\Psi_k(x) = x(x - 1)\Psi'_{k-1}(x)
\]
with $\Psi_0(x) = x - 1$. Note that $\Psi_{m-1}$ is a polynomial whose coefficients do not depend on $\theta$; therefore,

$$
\frac{d \Psi_{m-1}}{d \theta} = \frac{d \Psi_{m-1}}{dy} \frac{dy}{d \theta} = \frac{d \Psi_{m-1}}{dy} \left[ \frac{dy}{d \theta} + \frac{dy}{dx} \frac{dx}{d \theta} \right] = \frac{d \Psi_{m-1}}{dy} \left[ \frac{e^{-x} e^{-\theta}}{1 + e^{-x} (e^{-\theta} - 1)^2} + \frac{e^{-x} (e^{-\theta} - 1)}{1 + e^{-x} (e^{-\theta} - 1)^2} \frac{dx}{d \theta} \right]
$$

where

$$
\frac{dx}{d \theta} = \sum_k \frac{d \varphi(u_k)}{d \theta} = \sum_k \left[ -\frac{u_k e^{-\theta u_k}}{1 - e^{-\theta u_k}} + \frac{e^{-\theta}}{1 - e^{-\theta}} \right] = \sum_k \left[ -\frac{u_k}{e^{\theta u_k} - 1} + \frac{1}{e^{\theta} - 1} \right]
$$

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 $x_i = (x_{i1}, x_{i2}, \ldots, x_{im})^T$, where $i = 1, \ldots, n$, have been transformed into uniform variates $\hat{u}_i = (\hat{u}_{i1}, \ldots, \hat{u}_{im})$, $i = 1, \ldots, n$. 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 $i = 1, \ldots, n$ in increasing order.

Let $c(u_1, u_2, \ldots, u_m; \theta)$ be the density function of a copula $C(u_1, u_2, \ldots, u_m; \theta)$, and let $\theta$ be the parameter vector to be estimated. The parameter $\theta$ is estimated by maximum likelihood:

$$
\hat{\theta} = \arg\max_{\theta \in \Theta} \sum_{i=1}^n \log c(\hat{u}_{i1}, \ldots, \hat{u}_{im}; \theta)
$$
Exact Maximum Likelihood Estimation (MLE)

Suppose that the marginal distributions of vector elements \( x_i = (x_{i1}, x_{i2}, \ldots, x_{im})^\top \), where \( i = 1, \ldots, n \), are already known to be uniform. Then the parameter \( \theta \) is estimated by exact maximum likelihood:

\[
\hat{\theta} = \arg \max_{\theta \in \Theta} \sum_{i=1}^{n} \log c(x_{i1}, x_{i2}, \ldots, x_{im}; \theta)
\]

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 \( \Sigma \) in a \( t \) copula and the parameter \( \theta \) in Archimedean copulas can be estimated in this manner. The most frequently used estimator of Kendall’s tau is the rank correlation coefficient:

\[
\hat{\tau}_t (X_i, X_j) = \left( \frac{n}{2} \right)^{-1} \sum_{1 \leq t < s \leq n} \text{sign} \left( (x_{t,i} - x_{s,i}) (x_{t,j} - x_{s,j}) \right)
\]

The preceding formula is analogous to its population counterpart,

\[
\tau_t (X_i, X_j) = E \left[ \text{sign} \left( (X_i - \bar{X}_i) (X_j - \bar{X}_j) \right) \right]
\]

where \( (\bar{X}_i, \bar{X}_j) \) has the same distribution but is independent of \( (X_i, X_j) \).

For Archimedean multivariate copulas, there is only one parameter to estimate, \( \tau \) (or its function \( \theta \)), although for \( m \) variables there are \( m(m-1)/2 \) unique pairwise correlation coefficients. Denote the map from \( \tau \) to \( \theta \) by \( \theta = \hat{\theta}(\tau) \). To aggregate the map, take a simple arithmetic average:

\[
\hat{\theta} = \frac{2}{m(m-1)} \sum_{1 \leq i < j \leq m} \hat{\theta} \left[ \hat{\tau}_t (X_i, X_j) \right]
\]

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 set, if any
- data set 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 Sets

The number of columns and the names of columns in the OUTPSEUDO= and OUTUNIFORM= data sets 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 sets. These table names are listed in Table 6.3.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ODS Tables Created by the FIT Statement</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status</td>
<td>Default</td>
</tr>
<tr>
<td>PearsonCorrelations</td>
<td>Pearson correlations matrix</td>
<td>Default with elliptical copulas</td>
</tr>
<tr>
<td>KendallCorrelations</td>
<td>Kendall correlations matrix</td>
<td>Default with elliptical copulas</td>
</tr>
<tr>
<td>SpearmanCorrelations</td>
<td>Spearman correlations matrix</td>
<td>Default with normal copula</td>
</tr>
<tr>
<td>FitSummary</td>
<td>Summary of nonlinear estimation</td>
<td>Default</td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Parameter estimates</td>
<td>Default</td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>InputOptions</td>
<td>Input options</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>IterHist</td>
<td>Iteration history</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>IterStart</td>
<td>Optimization start</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>IterStop</td>
<td>Optimization results</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>ParameterEstimatesResults</td>
<td>Parameter estimates</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>ParameterEstimatesStart</td>
<td>Parameter estimates</td>
<td>ITPRINT</td>
</tr>
<tr>
<td><strong>ODS Tables Created by the PRINTTIMING Option</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TimingDetails</td>
<td>Detailed summary of time used for all phases of execution</td>
<td>PRINTTIMING=(DETAILS)</td>
</tr>
<tr>
<td>TimingSummary</td>
<td>Summary of time used for main phases of execution</td>
<td>PRINTTIMING</td>
</tr>
</tbody>
</table>
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 Inparm. 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:

```sas
proc print data = inparm;
run;
```

**Output 6.1.1 CCopula Pearson Correlations Matrix**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Y1</th>
<th>Y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>1.0</td>
</tr>
</tbody>
</table>

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 inparm data set.

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 set for the result of simulation in uniforms.

```sas
/* simulate the data from bivariate normal copula */
proc ccopula;
  var Y1-Y2;
  define cop normal (corr=mycas.inparm);
  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 set as well. 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.

```sas
/* 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;
  end;
  surv = surv1 & surv2;
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:

```plaintext
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**

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Median</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>1000000</td>
<td>2.00324</td>
<td>2.00150</td>
<td>1.38948</td>
<td>4.74646E-6</td>
<td>29.97492</td>
</tr>
<tr>
<td>time2</td>
<td>1000000</td>
<td>2.00365</td>
<td>2.00000</td>
<td>1.39228</td>
<td>6.84299E-7</td>
<td>31.14652</td>
</tr>
</tbody>
</table>

**Pearson Correlation Coefficients, N = 1000000**

Prob > |r| under H0: Rho=0

<table>
<thead>
<tr>
<th></th>
<th>time1</th>
<th>time2</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>1.00000</td>
<td>0.77006</td>
</tr>
<tr>
<td></td>
<td>&lt;.0001</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

**Kendall Tau b Correlation Coefficients, N = 1000000**

Prob > |tau| under H0: Tau=0

<table>
<thead>
<tr>
<th></th>
<th>time1</th>
<th>time2</th>
</tr>
</thead>
<tbody>
<tr>
<td>time1</td>
<td>1.00000</td>
<td>0.59061</td>
</tr>
<tr>
<td></td>
<td>&lt;.0001</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

The second and final step is to empirically estimate the default probabilities of two companies. This is done using the FREQ procedure as follows:

```plaintext
proc freq data=default;
   table survive survive1-survive2;
run;
```

The result is shown in Output 6.1.3.
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*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.


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:

```sas
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:

```sas
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:

```sas
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.”
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 218.

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:

```plaintext
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:

```plaintext
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 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:

```plaintext
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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies data tables and options</td>
<td>PROC CESM</td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies variables to forecast</td>
<td>FORECAST</td>
<td></td>
</tr>
<tr>
<td>Specifies the time ID variable</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td><strong>Data Set Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data table</td>
<td>PROC CESM</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the output data table</td>
<td>PROC CESM</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies the parameter output data table</td>
<td>PROC CESM</td>
<td>OUTTEST=</td>
</tr>
<tr>
<td>Specifies the forecast output data table</td>
<td>PROC CESM</td>
<td>OUTFOR=</td>
</tr>
<tr>
<td>Specify statistics output data table</td>
<td>PROC CESM</td>
<td>OUTSTAT=</td>
</tr>
<tr>
<td>Specifies the summary output data table</td>
<td>PROC CESM</td>
<td>OUTSUM=</td>
</tr>
<tr>
<td><strong>Accumulation and Seasonality Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the accumulation frequency</td>
<td>ID</td>
<td>INTERVAL=</td>
</tr>
<tr>
<td>Specifies the length of seasonal cycle</td>
<td>PROC CESM</td>
<td>SEASONALITY=</td>
</tr>
<tr>
<td>Specifies the interval alignment</td>
<td>ID</td>
<td>ALIGN=</td>
</tr>
<tr>
<td>Specifies the starting time ID value</td>
<td>ID</td>
<td>START=</td>
</tr>
<tr>
<td>Specifies the ending time ID value</td>
<td>ID</td>
<td>END=</td>
</tr>
<tr>
<td>Specifies the accumulation statistic</td>
<td>ID, FORECAST</td>
<td>ACCUMULATE=</td>
</tr>
<tr>
<td>Specifies how to interpret missing values</td>
<td>ID, FORECAST</td>
<td>SETMISSING=</td>
</tr>
<tr>
<td><strong>Forecasting Horizon, Holdback Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the data to hold back</td>
<td>FORECAST</td>
<td>BACK=</td>
</tr>
<tr>
<td>Specifies the forecast horizon or lead</td>
<td>FORECAST</td>
<td>LEAD=</td>
</tr>
<tr>
<td><strong>Forecasting Model Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the confidence limit width</td>
<td>FORECAST</td>
<td>ALPHA=</td>
</tr>
<tr>
<td>Specifies the model selection criterion</td>
<td>FORECAST</td>
<td>CRITERION=</td>
</tr>
<tr>
<td>Specifies the forecast model</td>
<td>FORECAST</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies the median forecasts</td>
<td>FORECAST</td>
<td>MEDIAN</td>
</tr>
<tr>
<td>Specifies the model transformation</td>
<td>FORECAST</td>
<td>TRANSFORM=</td>
</tr>
<tr>
<td><strong>Miscellaneous Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the time ID format</td>
<td>ID</td>
<td>FORMAT=</td>
</tr>
<tr>
<td>Specifies the method for trimming the data in the BY groups</td>
<td>ID</td>
<td>TRIMID=</td>
</tr>
</tbody>
</table>
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 200.

*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 200.

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.

**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 200.
OUTSTAT=\textit{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. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 200.

OUTSUM=\textit{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.

\textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 200.

SEASONALITY=\textit{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.

\textbf{BY Statement}

\texttt{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.

\textbf{FORECAST Statement}

\texttt{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 \texttt{variable-list} that contains one or more numeric variables. For more information about the \texttt{variable-list}, see the section “SAS Variable Lists” in \textit{SAS Language Reference: Concepts}.

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 \texttt{options} to specify which forecast model to use:
FORECAST Statement  ♦  205

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.
MULTISEASONAL 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(\(n\)) performs Box-Cox transformation with parameter number (\(n\)), where \(n\) 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 212 and “Inverse Transformations” on page 213.

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 210.
**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:

- **n** interprets a missing value as having the value *n*. You can specify any number for *n*, 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.
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

<table>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
<th>Option</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accumulation</td>
<td>ACCUMULATE=</td>
<td>ID, FORECAST</td>
</tr>
<tr>
<td>2</td>
<td>Missing value interpretation</td>
<td>SETMISSING=</td>
<td>ID, FORECAST</td>
</tr>
<tr>
<td>3</td>
<td>Transformations</td>
<td>TRANSFORM=</td>
<td>FORECAST</td>
</tr>
<tr>
<td>4</td>
<td>Parameter estimation</td>
<td>METHOD=</td>
<td>FORECAST</td>
</tr>
<tr>
<td>5</td>
<td>Forecasting</td>
<td>METHOD=, LEAD=</td>
<td>FORECAST</td>
</tr>
<tr>
<td>6</td>
<td>Inverse transformation</td>
<td>TRANSFORM, MEDIAN</td>
<td>FORECAST</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>19MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>19MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>11MAY1999</td>
<td>50</td>
</tr>
<tr>
<td>12MAY1999</td>
<td>20</td>
</tr>
<tr>
<td>23MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>40</td>
</tr>
<tr>
<td>01APR1999</td>
<td>60</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>90</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>20</td>
</tr>
<tr>
<td>01APR1999</td>
<td>20</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>30</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>01APR1999</td>
<td>10</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>20</td>
</tr>
<tr>
<td>01APR1999</td>
<td>20</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>01APR1999</td>
<td>30</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>50</td>
</tr>
</tbody>
</table>
• If the ACCUMULATE=FIRST option is specified, the resulting time series is

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>50</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>14.14</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>17.32</td>
</tr>
</tbody>
</table>

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 213. 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_: t values of parameter estimate
- _PVALUE_: 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 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
### Chapter 7: The CESM Procedure

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URMSE</td>
<td>unbiased root mean square error</td>
</tr>
<tr>
<td>MAPE</td>
<td>mean absolute percentage error</td>
</tr>
<tr>
<td>MAE</td>
<td>mean absolute error</td>
</tr>
<tr>
<td>RSQUARE</td>
<td>R-square</td>
</tr>
<tr>
<td>ADJRSQ</td>
<td>adjusted R-square</td>
</tr>
<tr>
<td>AADJRSQ</td>
<td>Amemiya’s adjusted R-square</td>
</tr>
<tr>
<td>RWRSQ</td>
<td>random walk R-square</td>
</tr>
<tr>
<td>AIC</td>
<td>Akaike’s information criterion</td>
</tr>
<tr>
<td>AICC</td>
<td>finite sample corrected AIC</td>
</tr>
<tr>
<td>SBC</td>
<td>Schwarz Bayesian information criterion</td>
</tr>
<tr>
<td>APC</td>
<td>Amemiya’s prediction criterion</td>
</tr>
<tr>
<td>MAXERR</td>
<td>maximum error</td>
</tr>
<tr>
<td>MINERR</td>
<td>minimum error</td>
</tr>
<tr>
<td>MAXPE</td>
<td>maximum percentage error</td>
</tr>
<tr>
<td>MINPE</td>
<td>minimum percentage error</td>
</tr>
<tr>
<td>ME</td>
<td>mean error</td>
</tr>
<tr>
<td>MPE</td>
<td>mean percentage error</td>
</tr>
<tr>
<td>MDAPE</td>
<td>median absolute percentage error</td>
</tr>
<tr>
<td>GMAPE</td>
<td>geometric mean absolute percentage error</td>
</tr>
<tr>
<td>MINPPE</td>
<td>minimum predictive percentage error</td>
</tr>
<tr>
<td>MAXPPE</td>
<td>maximum predictive percentage error</td>
</tr>
<tr>
<td>MPPE</td>
<td>mean predictive percentage error</td>
</tr>
<tr>
<td>MAPPE</td>
<td>symmetric mean absolute predictive percentage error</td>
</tr>
<tr>
<td>MDAPPE</td>
<td>median absolute predictive percentage error</td>
</tr>
<tr>
<td>GMAPPE</td>
<td>geometric mean absolute predictive percentage error</td>
</tr>
<tr>
<td>MINSPE</td>
<td>minimum symmetric percentage error</td>
</tr>
<tr>
<td>MAXSPE</td>
<td>maximum symmetric percentage error</td>
</tr>
<tr>
<td>MSPE</td>
<td>mean symmetric percentage error</td>
</tr>
<tr>
<td>SMAPE</td>
<td>symmetric mean absolute percentage error</td>
</tr>
<tr>
<td>MDASPE</td>
<td>median absolute symmetric percentage error</td>
</tr>
<tr>
<td>GMASPE</td>
<td>geometric mean absolute symmetric percentage error</td>
</tr>
<tr>
<td>MINRE</td>
<td>minimum relative error</td>
</tr>
<tr>
<td>MAXRE</td>
<td>maximum relative error</td>
</tr>
<tr>
<td>MRE</td>
<td>mean relative error</td>
</tr>
<tr>
<td>MRAE</td>
<td>mean relative absolute error</td>
</tr>
</tbody>
</table>
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.

```plaintext
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:

```plaintext
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**

<table>
<thead>
<tr>
<th>Summary of time series processing for SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of analysis variables</td>
</tr>
<tr>
<td>Number of rows read</td>
</tr>
<tr>
<td>Number of groups read</td>
</tr>
<tr>
<td>Memory for group packages (KB)</td>
</tr>
<tr>
<td>Time to load groups (seconds)</td>
</tr>
<tr>
<td>Minimum time ID</td>
</tr>
<tr>
<td>Maximum time ID</td>
</tr>
<tr>
<td>Minimum time periods</td>
</tr>
<tr>
<td>Maximum time periods</td>
</tr>
<tr>
<td>Number of nodes run</td>
</tr>
<tr>
<td>Number of nodes with data</td>
</tr>
<tr>
<td>Number of nodes with groups</td>
</tr>
<tr>
<td>Number of threads budgeted</td>
</tr>
<tr>
<td>Minimum thread group count</td>
</tr>
<tr>
<td>Maximum thread group count</td>
</tr>
<tr>
<td>Minimum threads active</td>
</tr>
<tr>
<td>Maximum threads active</td>
</tr>
<tr>
<td>Number of groups processed by submitted code</td>
</tr>
<tr>
<td>Number of groups failing</td>
</tr>
<tr>
<td>Elapsed time to process groups (seconds)</td>
</tr>
</tbody>
</table>

The following statements plot the forecasts for the sale of belts, blouses, and laces:

```plaintext
%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);
   reline &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');
```
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:

```plaintext
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:
Example 7.1: Forecasting of Time Series Data

%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)
The following statements print the methods that are selected for each variable, as shown in Output 7.1.8:

```plaintext
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>belts</td>
<td>SEASONAL</td>
</tr>
<tr>
<td>blouses</td>
<td>WINTERS</td>
</tr>
<tr>
<td>coats</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>dresses</td>
<td>SEASONAL</td>
</tr>
<tr>
<td>hats</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>laces</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>shirts</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>shoes</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>socks</td>
<td>ADDWINTERS</td>
</tr>
<tr>
<td>ties</td>
<td>SEASONAL</td>
</tr>
</tbody>
</table>

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;
Example 7.2: Forecasting of Transactional Data

/*- 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:

```r
%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)
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:

```ruby
%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 7.3.2  Internet Data Forecast Plots (Cars)
Output 7.3.3 Internet Data Forecast Plots (Planes)
Chapter 8
The CNTSELECT Procedure

Contents

Overview: CNTSELECT Procedure ....................................................... 236
PROC CNTSELECT Features ........................................................... 236
PROC CNTSELECT Compared with Other SAS Procedures ..................... 237
Using CAS Sessions and CAS Engine Librefs .................................. 238
Getting Started: CNTSELECT Procedure ......................................... 239
Syntax: CNTSELECT Procedure ....................................................... 241
  Functional Summary ..................................................................... 242
  PROC CNTSELECT Statement ....................................................... 244
  BOUNDS Statement ..................................................................... 246
  BY Statement .............................................................................. 247
  CLASS Statement ......................................................................... 247
  DISPLAY Statement ...................................................................... 248
  DISPLAYOUT Statement ................................................................ 249
  DISPMODEL Statement ................................................................ 249
  EFFECT Statement ....................................................................... 250
  FREQ Statement .......................................................................... 251
  INIT Statement ........................................................................... 252
  MODEL Statement ......................................................................... 252
  OUTPUT Statement ....................................................................... 254
  RESTRICT Statement ................................................................... 255
  SELECTION Statement .................................................................. 257
  TEST Statement ........................................................................... 259
  VIEWSTORE Statement ................................................................ 260
  WEIGHT Statement ....................................................................... 261
  ZEROMODEL Statement ................................................................ 261
Details: CNTSELECT Procedure ....................................................... 262
  Missing Values ........................................................................... 262
  Poisson Regression ....................................................................... 262
  Conway-Maxwell-Poisson Regression ............................................ 263
  Negative Binomial Regression ....................................................... 267
  Zero-Inflated Count Regression Overview ...................................... 269
  Zero-Inflated Poisson Regression .................................................. 270
  Zero-Inflated Conway-Maxwell-Poisson Regression ......................... 271
  Zero-Inflated Negative Binomial Regression .................................... 273
  Parameter Naming Conventions for the RESTRICT, TEST, BOUNDS, and INIT Statements ......................................................... 274
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 $E(y_i|x_i)$ of the dependent variable $y_i$ is assumed to be a function of a vector of covariates $x_i$.

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 $\varphi_i$, which is determined by a separate model, $\varphi_i = F(z'_i \gamma)$, where $F$ is the normal or logistic distribution function that results in a probit or logistic model and $z_i$ is a set of covariates.

PROC CNTSELECT Features

The CNTSELECT procedure estimates the parameters of a count regression model by maximum likelihood techniques.

The CNTSELECT procedure supports the following models for count data:
PROC CNTSELECT Compared with Other SAS Procedures

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 set that contains predicted probabilities and other observationwise statistics
- supports effect-selection methods

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 provides all the functionality of the HPCOUNTREG procedure.
Chapter 8: The CNTSELECT 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:

```sas
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:

```sas
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:

```sas
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.”
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:

```sas
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](image)

<table>
<thead>
<tr>
<th>Obs</th>
<th>art</th>
<th>fem</th>
<th>mar</th>
<th>kid5</th>
<th>phd</th>
<th>ment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1.38000</td>
<td>8.0000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3.59000</td>
<td>19.0000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.12000</td>
<td>10.0000</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.80000</td>
<td>4.0000</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3.41000</td>
<td>10.0000</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.10000</td>
<td>2.0000</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.26000</td>
<td>5.0000</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3.85000</td>
<td>16.0000</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.26000</td>
<td>2.0000</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4.29000</td>
<td>10.0000</td>
</tr>
</tbody>
</table>

The following SAS statements estimate the Poisson regression model.

```sas
/*--- 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

<table>
<thead>
<tr>
<th>Model Fit Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependent Variable: art</td>
</tr>
<tr>
<td>Number of Observations: 915</td>
</tr>
<tr>
<td>Data Set: LONG97DATA</td>
</tr>
<tr>
<td>Model: Poisson</td>
</tr>
<tr>
<td>Log Likelihood: -1651.06</td>
</tr>
<tr>
<td>Maximum Absolute Gradient: 0.001295</td>
</tr>
<tr>
<td>Number of Iterations: 11</td>
</tr>
<tr>
<td>Optimization Method: Quasi-Newton</td>
</tr>
<tr>
<td>AIC: 3314.113</td>
</tr>
<tr>
<td>SBC: 3343.026</td>
</tr>
<tr>
<td>Covariance Estimation: Hessian</td>
</tr>
</tbody>
</table>

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 | 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.
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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Set Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data set</td>
<td>PROC CNTSELECT</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the identification variable for panel data analysis</td>
<td>PROC CNTSELECT</td>
<td>GROUPID=</td>
</tr>
<tr>
<td>Writes estimates to an output data set</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies the classification variables</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>Specifies an optional frequency variable</td>
<td>FREQ</td>
<td></td>
</tr>
<tr>
<td>Specifies an optional weight variable</td>
<td>WEIGHT</td>
<td></td>
</tr>
<tr>
<td><strong>Printing Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prints the correlation matrix of the estimates</td>
<td>PROC CNTSELECT</td>
<td>CORRB</td>
</tr>
<tr>
<td>Prints the covariance matrix of the estimates</td>
<td>PROC CNTSELECT</td>
<td>COVB</td>
</tr>
<tr>
<td>Suppresses the normal printed output</td>
<td>PROC CNTSELECT</td>
<td>NOPRINT</td>
</tr>
<tr>
<td>Requests all printing options</td>
<td>PROC CNTSELECT</td>
<td>PRINTALL</td>
</tr>
</tbody>
</table>
### Table 8.1 continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prints timing information</td>
<td>PROC CNTSELECT</td>
<td>PRINTTIMING</td>
</tr>
<tr>
<td>Prints the names used internally for the parameters</td>
<td>PROC CNTSELECT</td>
<td>PRINTINTERNALNAMES</td>
</tr>
<tr>
<td><strong>Options to Control the Optimization Process</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selects the iterative minimization method to use</td>
<td>PROC CNTSELECT</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies maximum number of iterations allowed</td>
<td>PROC CNTSELECT</td>
<td>MAXITER=</td>
</tr>
<tr>
<td>Specifies maximum number of function calls</td>
<td>PROC CNTSELECT</td>
<td>MAXFUNC=</td>
</tr>
<tr>
<td>Specifies the upper limit of CPU time in seconds</td>
<td>PROC CNTSELECT</td>
<td>MAXTIME=</td>
</tr>
<tr>
<td>Sets boundary restrictions on parameters</td>
<td>BOUNDS</td>
<td></td>
</tr>
<tr>
<td>Sets initial values for parameters</td>
<td>INIT</td>
<td></td>
</tr>
<tr>
<td>Sets linear restrictions on parameters</td>
<td>RESTRICT</td>
<td></td>
</tr>
<tr>
<td><strong>Model Estimation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the dispersion variables</td>
<td>DISPMODEL</td>
<td></td>
</tr>
<tr>
<td>Specifies the type of model</td>
<td>PROC CNTSELECT</td>
<td>DIST=</td>
</tr>
<tr>
<td>Specifies the type of covariance matrix</td>
<td>PROC CNTSELECT</td>
<td>COVEST=</td>
</tr>
<tr>
<td>Specifies the type of error components model for panel data</td>
<td>MODEL</td>
<td>ERRORCOMP=</td>
</tr>
<tr>
<td>Suppresses the intercept parameter</td>
<td>MODEL</td>
<td>NOINT</td>
</tr>
<tr>
<td>Specifies the offset variable</td>
<td>MODEL</td>
<td>OFFSET=</td>
</tr>
<tr>
<td>Specifies the parameterization for the Conway-Maxwell-Poisson (CMP) model</td>
<td>MODEL</td>
<td>PARAMETER=</td>
</tr>
<tr>
<td>Specifies the zero-inflated offset variable</td>
<td>ZEROMODEL</td>
<td>OFFSET=</td>
</tr>
<tr>
<td>Specifies the zero-inflated link function</td>
<td>ZEROMODEL</td>
<td>LINK=</td>
</tr>
<tr>
<td>Specifies the constructed regression effects</td>
<td>EFFECT</td>
<td></td>
</tr>
<tr>
<td><strong>Regression Effect-Selection Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the selection method</td>
<td>SELECTION</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies how to apply the STOP= criterion</td>
<td>SELECTION</td>
<td>STOPHORIZON=</td>
</tr>
<tr>
<td><strong>Regression Effect-Selection Method Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies a criterion for choosing the best model at each step</td>
<td>SELECTION</td>
<td>CHOOSE=</td>
</tr>
<tr>
<td>Specifies that the competitive form of the stepwise selection method be used</td>
<td>SELECTION</td>
<td>COMPETITIVE</td>
</tr>
<tr>
<td>Specifies the maximum number of effects in the model</td>
<td>SELECTION</td>
<td>MAXEFFECTS=</td>
</tr>
<tr>
<td>Specifies the maximum number of selection steps</td>
<td>SELECTION</td>
<td>MAXSTEPS=</td>
</tr>
<tr>
<td>Specifies the minimum number of effects in the model</td>
<td>SELECTION</td>
<td>MINEFFECTS=</td>
</tr>
<tr>
<td>Specifies a criterion to decide which effects enter or leave the model at each step</td>
<td>SELECTION</td>
<td>SELECT=</td>
</tr>
<tr>
<td>Specifies a criterion for stopping the selection process</td>
<td>SELECTION</td>
<td>STOP=</td>
</tr>
</tbody>
</table>
### Table 8.1  continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td>Specifies the output item store to preserve the properties of the model and the results of fitting the model</td>
<td>PROC CNTSELECT STORE=</td>
<td></td>
</tr>
<tr>
<td>Outputs SAS variables to the output data set</td>
<td>OUTPUT</td>
<td>COPYVAR=</td>
</tr>
<tr>
<td>Outputs the estimates of dispersion for the CMP model</td>
<td>OUTPUT</td>
<td>DISPERSION</td>
</tr>
<tr>
<td>Outputs the estimates of $G\Delta = g'_i\delta$ for the CMP model</td>
<td>OUTPUT</td>
<td>GDELTA=</td>
</tr>
<tr>
<td>Outputs the estimates of $\lambda$ for the CMP model</td>
<td>OUTPUT</td>
<td>LAMBDA=</td>
</tr>
<tr>
<td>Outputs the estimates of $\nu$ for the CMP model</td>
<td>OUTPUT</td>
<td>NU=</td>
</tr>
<tr>
<td>Outputs the estimates of $\mu$ for the CMP model</td>
<td>OUTPUT</td>
<td>MU=</td>
</tr>
<tr>
<td>Outputs the estimates of mode for the CMP model</td>
<td>OUTPUT</td>
<td>MODE=</td>
</tr>
<tr>
<td>Outputs the probability that the response variable will take the current value</td>
<td>OUTPUT</td>
<td>PROB=</td>
</tr>
<tr>
<td>Outputs probabilities for particular response values</td>
<td>OUTPUT</td>
<td>PROBCOUNT( )</td>
</tr>
<tr>
<td>Outputs expected value of response variable</td>
<td>OUTPUT</td>
<td>PRED=</td>
</tr>
<tr>
<td>Outputs the estimates of variance for the CMP model</td>
<td>OUTPUT</td>
<td>VARIANCE=</td>
</tr>
<tr>
<td>Outputs estimates of $X\beta = x'_i\beta$</td>
<td>OUTPUT</td>
<td>XBETA=</td>
</tr>
<tr>
<td>Outputs estimates of $Z\gamma = z'_i\gamma$</td>
<td>OUTPUT</td>
<td>ZGAMMA=</td>
</tr>
<tr>
<td>Outputs probability of a zero value as a result of the zero-generating process</td>
<td>OUTPUT</td>
<td>PROBZERO=</td>
</tr>
</tbody>
</table>

---

**PROC CNTSELECT Statement**

PROC CNTSELECT <options> ;

The following options can be used in the PROC CNTSELECT statement.

**Input Data Set Options**

**DATA=SAS-data-set**

specifies the input SAS data set. If the DATA= option is not specified, PROC CNTSELECT uses the most recently created SAS data set.
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 274.

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 255.

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 \( x_1 \) through \( x_{10} \) to be between 0 and 1, and the parameter for \( x_1 \) in the zero-inflation model to be less than 1.

\[
\text{bounds } z < 0, \\
0 < x_1-x_{10} < 1, \\
\text{Inf}_x_1 < 1;
\]

**BY Statement**

```
BY variables ;
```

A BY statement can be used with PROC CNTSELECT to obtain separate analyses on observations in groups defined by the BY variables. When a BY statement appears, the input data set should be sorted in order of 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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Allows design columns for a variable to enter or leave the model independently</td>
</tr>
</tbody>
</table>
Chapter 8: The CNTSELECT Procedure

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 280. 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 subsetted 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 280. 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 \( g_i' \) be the ith 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 \( g_i' \delta \), where \( \delta \) is the \( (q + 1) \times 1 \) vector of parameters to be estimated, the dispersion model intercept is \( \delta_0 \), and the coefficients for the q dispersion covariates are \( \delta_1, \ldots, \delta_q \). If you specify DISP=CMPOISSON in the MODEL statement but do not include a
DISPMODEL statement, then only the intercept term $\delta_0$ 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 set) 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:

```plaintext
proc cntselect data=docvisit;
   model doctorvisits=sex illness income / dist=cmpoisson;
   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 50 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collection Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the constituents of the collection effect</td>
</tr>
<tr>
<td><strong>Multimember Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the levels of the multimember effect</td>
</tr>
<tr>
<td>NOEFFECT</td>
<td>Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns</td>
</tr>
</tbody>
</table>
Table 8.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STDIZE</td>
<td>Standardizes the design matrix entries so that each observation has a sum of 1</td>
</tr>
<tr>
<td>WEIGHT=</td>
<td>Specifies the weight variable for the contributions of each classification effect</td>
</tr>
</tbody>
</table>

**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

**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**

\[
\text{INIT } \text{initialization1 } < , \text{ initialization2 } \ldots > ;
\]

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:

\[
\text{parameter } <= \text{ number}
\]

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 '*, '-', '('), 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 274.

**MODEL Statement**

\[
\text{MODEL } \text{dependent-variable } = \text{ regressors } / \text{ options } ;
\]

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 set. 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 (/):

\[
\text{DIST=} \text{value}
\]

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 set 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 set 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 \( \mu \) 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 SAS data set that includes variables created by the output-options. These variables include the estimates of $x_i^\beta$, 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 set contain the estimates of $z_i^\gamma$ 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 $g_i^\delta$, $\lambda$, $\nu$, $\mu$, 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 set, 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=SAS-data-set**
  names the output data set

- **COPYVAR=SAS-variable-names**

- **COPYVARS=SAS-variable-names**
  adds SAS variables to the output data set.

- **XBETA=name**
  names the variable to contain estimates of $x_i^\beta$.

- **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.

- **ZGAMMA=name**
  names the variable to contain estimates of $z_i^\gamma$. 


PROBZERO=name
names the variable to contain the value of $\varphi_i$, 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.

GDELTA=name
assigns a name to the variable that contains estimates of $\gamma_i \delta$ for the
Conway-Maxwell-Poisson distribution.

LAMBDA=name
assigns a name to the variable that contains the estimate of $\lambda$ for the
Conway-Maxwell-Poisson distribution.

NU=name
assigns a name to the variable that contains the estimate of $\nu$ for the
Conway-Maxwell-Poisson distribution.

MU=name
assigns a name to the variable that contains the estimate of $\mu$ for the
Conway-Maxwell-Poisson distribution.

MODE=name
assigns a name to the variable that contains the integral part of $\mu$ (mode)
for the Conway-Maxwell-Poisson distribution.

VARIANCE=name
assigns a name to the variable that contains the estimate of variance for the
Conway-Maxwell-Poisson distribution.

DISPERSION=name
assigns a name to the variable that contains the value of dispersion for the
Conway-Maxwell-Poisson distribution.

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 set 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:

```plaintext
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 274.

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:

```plaintext
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:

```plaintext
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 $\alpha$ 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 35 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 58.

PROC CNTSELECT supports a specific set of values for the following method-options:
**Chapter 8: The CNTSELECT Procedure**

**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 35 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 35 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:

\[ \text{expression} = \text{expression} \]

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 256.

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.

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**

```plaintext
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.
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 (/):

NONORMALIZE

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 NONORMALIZE 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 (\( \psi_i \)) of a zero count. Each of these \( q \) variables has a parameter to be estimated in the regression. For example, let \( z_{i0} \) 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 \( \psi_i \) is a function of \( z_{i0} \psi \), where \( \psi \) is the \((q + 1) \times 1\) vector of parameters to be estimated. (The zero-inflated intercept is \( \gamma_0 \); the coefficients for the \( q \) zero-inflated covariates are \( \gamma_1, \ldots, \gamma_q \).) If \( q \) is equal to 0 (no ZI explanatory variables are provided), then only the intercept term \( \gamma_0 \) 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 set) 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 (/):

NONE

suppresses printing.

OPTIMIZERSETTINGS

prints the optimizer settings that were used to fit the model.
LINK=LOGISTIC | NORMAL
specifies the distribution function used to compute probability of zeros. The supported distribution
functions are as follows:

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 set 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 ($\psi_i$) 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 set 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 set 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 set contains any observations that have missing response values but nonmissing regressors,
PROC CNTSELECT can compute several statistics and store them in an output data set by using the OUTPUT
statement. For example, you can request that the output data set contain the estimates of $x_i^T \beta$, 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 set contain the estimates
of $z_i^T \gamma$, 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 $x_i$, is independently Poisson distributed with

$$P(Y_i = y_i | x_i) = \frac{e^{-\mu_i} \mu_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \ldots$$

and the mean parameter—that is, the mean number of events per period—is given by

$$\mu_i = \exp(x_i^T \beta)$$
Conway-Maxwell-Poisson Regression

where $\beta$ is a $(k + 1) \times 1$ parameter vector. (The intercept is $\beta_0$; the coefficients for the $k$ regressors are $\beta_1, \ldots, \beta_k$.) Taking the exponential of $x'_i \beta$ ensures that the mean parameter $\mu_i$ is nonnegative. It can be shown that the conditional mean is given by

$$E(y_i|x_i) = \mu_i = \exp(x'_i \beta)$$

Note that the conditional variance of the count random variable is equal to the conditional mean in the Poisson regression model:

$$V(y_i|x_i) = E(y_i|x_i) = \mu_i$$

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

$$L = \sum_{i=1}^{N} (-\mu_i + y_i \ln \mu_i - \ln y_i!) = \sum_{i=1}^{N} (-e^{x'_i \beta} + y_i x'_i \beta - \ln y_i!$$

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 $x_i$, is independently Poisson-distributed as

$$P(Y_i = y_i|x_i) = \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}, \quad y_i = 0, 1, 2, \ldots$$

The Conway-Maxwell-Poisson distribution is defined as

$$P(Y_i = y_i|x_i, z_i) = \frac{1}{Z(\lambda_i, v_i)} \left( \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}} \right), \quad y_i = 0, 1, 2, \ldots$$

where the normalization factor is

$$Z(\lambda_i, v_i) = \sum_{n=0}^{\infty} \frac{\lambda_i^n}{(n!)^{v_i}}$$
and
\[ \lambda_i = \exp(x_i' \beta) \]
\[ v_i = -\exp(g_i' \delta) \]

The \( \beta \) vector is a \((k + 1) \times 1 \) parameter vector. (The intercept is \( \beta_0 \), and the coefficients for the \( k \) regressors are \( \beta_1, \ldots, \beta_k \).) The \( \delta \) vector is an \((m + 1) \times 1 \) parameter vector. (The intercept is represented by \( \delta_0 \), and the coefficients for the \( m \) regressors are \( \delta_1, \ldots, \delta_m \).) The covariates are represented by \( x_i \) and \( g_i \) vectors.

One of the restrictive properties of the Poisson model is that the conditional mean and variance must be equal:
\[ E(y_i | x_i) = V(y_i | x_i) = \lambda_i = \exp(x_i' \beta) \]

The CMP distribution overcomes this restriction by defining an additional parameter, \( \nu \), which governs the rate of decay of successive ratios of probabilities such that
\[ P(Y_i = y_i - 1) / P(Y_i = y_i) = (y_i)_{v_i} / \lambda_i \]

The introduction of the additional parameter, \( \nu \), 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 \( \lambda \) and \( \nu \). 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 \( y_1, y_2, \ldots, y_n \) is written as

\[
L(y_1, y_2, \ldots, y_n | \lambda, \nu) = \frac{\prod_{i=1}^{n} \lambda^{y_i} Z(\lambda, \nu)^{-n}}{(\prod_{i=1}^{n} y_i!)} \exp \left( -\nu \sum_{i=1}^{n} \ln(y_i!) \right) Z(\lambda, \nu)^{-n} = \lambda^S_1 \exp (-\nu S_2) Z(\lambda, \nu)^{-n} \]

where \( S_1 \) and \( S_2 \) are sufficient statistics for \( y_1, y_2, \ldots, y_n \). 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

\[
\mathcal{L}_\beta = \left( \sum_{k=1}^{N} y_k - n \frac{\lambda Z(\lambda, \nu) \lambda}{Z(\lambda, \nu)} \right) x \\
\mathcal{L}_\delta = \left( \sum_{k=1}^{N} \ln(y_k!) - n \frac{Z(\lambda, \nu) \nu}{Z(\lambda, \nu)} \right) v z \]
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 $\lambda = \mu^v$, the Guikema and Coffelt (2008) formulation is written as

$$P(Y = y_i; \mu, v) = \frac{1}{S(\mu, v)} \left( \frac{\mu^v y_i}{y_i!} \right)^v$$

where the new normalization factor is defined as

$$S(\mu, v) = \sum_{j=0}^{\infty} \left( \frac{\mu^v}{j!} \right)^v$$

In terms of their new formulations, the mean and variance of $Y$ are given as

$$E[Y] = \frac{1}{v} \frac{\partial \ln S}{\partial \ln \mu}$$

$$V[Y] = \frac{1}{v^2} \frac{\partial^2 \ln S}{\partial^2 \ln \mu}$$

They can be approximated as

$$E[Y] \approx \mu + \frac{1}{2} v - \frac{1}{2}$$

$$V[Y] \approx \frac{\mu}{v}$$

In the CNTSELECT procedure, the mean and variance are calculated according to the following formulas, respectively, for the Guikema and Coffelt (2008) model:

$$E(Y) = \frac{1}{Z(\lambda, \mu)} \sum_{j=0}^{\infty} \frac{j \mu^v}{(j!)^v}$$

$$V(Y) = \frac{1}{Z(\lambda, \mu)} \sum_{j=0}^{\infty} \frac{j^2 \mu^v}{(j!)^v} - E(Y)^2$$

In terms of the new parameter $\mu$, the log-likelihood function is specified as

$$\mathcal{L} = \ln(S(\mu, v)) + v \sum_{i=1}^{N} (y_i \ln(\mu) - \ln(y_i!))$$

and the gradients are calculated as

$$\mathcal{L}_\beta = \left( v \sum_{i=1}^{N} y_i - \frac{\mu S(\mu, v)}{S(\mu, v)} \right) x$$
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 \( \mu \) 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 \( \mu \) representing the mode, which is a reasonable approximation for the mean. The dispersion can be written as

\[
D(Y) = \frac{V(Y)}{E(Y)} \approx \frac{\mu}{\mu + \frac{1}{2}v - \frac{1}{2}} \approx \frac{1}{v}
\]

When \( v < 1 \), 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 \( v = 1 \) and the mean and variance are equal, the dispersion is equal to 1 (Poisson model). When \( v > 1 \), 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

\[
-\ln(v) = \delta_0 + \sum_{n=1}^{q} \delta_n g_n
\]

Negative values of \( \ln(v) \) indicate that the data are approximately overdispersed, and positive values of \( \ln(v) \) 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

\[
E(y_i|\mathbf{x}_i, \tau_i) = \mu_i \tau_i = e^{x_i'\beta + \epsilon_i}
\]

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 \( \tau_i \) 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 $\tau_i$. Then, the distribution $f(y_i|x_i)$ (no longer conditional on $\tau_i$) is obtained by integrating $f(y_i|x_i, \tau_i)$ with respect to $\tau_i$:

$$f(y_i|x_i) = \int_0^\infty f(y_i|x_i, \tau_i)g(\tau_i)d\tau_i$$

An analytical solution to this integral exists when $\tau_i$ 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 $E(e^{\mu_i}) = E(\tau_i) = 1$. Thus, it is assumed that $\tau_i$ follows a gamma($\theta$, $\theta$) distribution with $E(\tau_i)$ 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 $\theta$ is a positive parameter. Then, the density of $y_i$ given $x_i$ is derived as

$$f(y_i|x_i) = \int_0^\infty f(y_i|x_i, \tau_i)g(\tau_i)d\tau_i$$

$$= \frac{\theta^\theta \mu_i^{y_i}}{y_i!\Gamma(\theta)} \int_0^\infty e^{-(\mu_i+\theta)\tau_i} \tau_i^{\theta+y_i-1} d\tau_i$$

$$= \frac{\theta^\theta \mu_i^{y_i}}{y_i!\Gamma(\theta)(\theta + \mu_i)^{\theta+y_i}}$$

$$= \frac{\Gamma(y_i + \theta)}{y_i!\Gamma(\theta)} \left( \frac{\theta}{\theta + \mu_i} \right)^\theta \left( \frac{\mu_i}{\theta + \mu_i} \right)^{y_i}$$

If you make the substitution $\alpha = \frac{1}{\theta}$ ($\alpha > 0$), the negative binomial distribution can then be rewritten as

$$f(y_i|x_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i!\Gamma(\alpha^{-1})} \left( \frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left( \frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i = 0, 1, 2, \ldots$$

Thus, the negative binomial distribution is derived as a gamma mixture of Poisson random variables. It has the conditional mean

$$E(y_i|x_i) = \mu_i = e^{\mu_i}$$

and the conditional variance

$$V(y_i|x_i) = \mu_i[1 + \frac{1}{\theta} \mu_i] = \mu_i[1 + \alpha \mu_i] > E(y_i|x_i)$$

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 $V(y_i|x_i) = \mu_i + \alpha \mu_i^2$, 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 $\alpha = \frac{1}{\theta} = 0$. A Wald test of this hypothesis is provided (it is the reported $t$ statistic for the estimated $\alpha$ in the negative binomial model).
The log-likelihood function of the negative binomial regression model (NEGBIN2) is given by

\[
\mathcal{L} = \sum_{i=1}^{N} \left\{ \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) - \ln(y_i!) \right.
\]
\[
-(y_i + \alpha^{-1}) \ln(1 + \exp(x_i' \beta)) + y_i \ln(\alpha) + y_i x_i' \beta \left. \right\}
\]

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 \( \mu_i \) 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 \( f(y_i|x_i) \), except that \( \alpha^{-1} \) is replaced everywhere by \( \alpha^{-1} \mu_i^{2-p} \). The negative binomial model NEGBIN1, which sets \( p = 1 \), has the variance function \( V(y_i|x_i) = \mu_i + \alpha \mu_i \), 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

\[
\mathcal{L} = \sum_{i=1}^{N} \left\{ \sum_{j=0}^{y_i-1} \ln \left( j + \alpha^{-1} \exp(x_i' \beta) \right) \right. 
\]
\[
- \ln(y_i!) - \left( y_i + \alpha^{-1} \exp(x_i' \beta) \right) \ln(1 + \alpha) + y_i \ln(\alpha) \left. \right\}
\]

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 \( \varphi_i \) 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

\[
P(y_i = 0|x_i) = \varphi_i + (1 - \varphi_i)g(0) \\
P(y_i|x_i) = (1 - \varphi_i)g(y_i), \quad y_i > 0
\]
where \( g(y_i) \) follows either the Poisson or the negative binomial distribution.

If the probability \( \varphi_i \) depends on the characteristics of observation \( i \), then \( \varphi_i \) is written as a function of \( z'_i \gamma \), where \( z'_i \) is the \( 1 \times (q + 1) \) vector of zero-inflated covariates and \( \gamma \) is the \( (q + 1) \times 1 \) vector of zero-inflated coefficients to be estimated. (The zero-inflated intercept is \( \gamma_0 \); the coefficients for the \( q \) zero-inflated covariates are \( \gamma_1, \ldots, \gamma_q \).) The function \( F \) that relates the product \( z'_i \gamma \) (which is a scalar) to the probability \( \varphi_i \) is called the zero-inflated link function,

\[
\varphi_i = F_i = F(z'_i \gamma)
\]

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,

\[
F(z'_i \gamma) = \Lambda(z'_i \gamma) = \frac{\exp(z'_i \gamma)}{1 + \exp(z'_i \gamma)}
\]

or the standard normal cumulative distribution function (also called the probit function),

\[
F(z'_i \gamma) = \Phi(z'_i \gamma) = \int_0^{z'_i \gamma} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-u^2}{2}\right) du
\]

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 \( \mu_i = \exp(z'_i \beta) \). Thus the ZIP model is defined as

\[
P(y_i = 0|x_i, z_i) = F_i + (1 - F_i) \exp(-\mu_i)
\]

\[
P(y_i|x_i, z_i) = (1 - F_i) \frac{\exp(-\mu_i) \mu_i^{y_i}}{y_i!}, \quad y_i > 0
\]

The conditional expectation and conditional variance of \( y_i \) are given by

\[
E(y_i|x_i, z_i) = \mu_i (1 - F_i)
\]

\[
V(y_i|x_i, z_i) = E(y_i|x_i, z_i)(1 + \mu_i F_i)
\]

Note that the ZIP model (in addition to the ZINB model) exhibits overdispersion because \( V(y_i|x_i, z_i) > E(y_i|x_i, z_i) \).

In general, the log-likelihood function of the ZIP model is

\[
L = \sum_{i=1}^{N} \ln \left[ P(y_i|x_i, z_i) \right]
\]

After a specific link function (either logistic or standard normal) for the probability \( \varphi_i \) 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 $\psi_i$ is expressed by a logistic link function, namely

$$
\psi_i = \frac{\exp(z_i' \gamma)}{1 + \exp(z_i' \gamma)}
$$

The log-likelihood function is

$$
\mathcal{L} = \sum_{\{i: y_i=0\}} \ln \left[ \exp(z_i' \gamma) + \exp(-\exp(x_i' \beta)) \right] 
+ \sum_{\{i: y_i>0\}} \left[ y_i x_i' \beta - \exp(x_i' \beta) - \sum_{k=2}^{y_i} \ln(k) \right] 
- \sum_{i=1}^{N} \ln \left[ 1 + \exp(z_i' \gamma) \right]
$$

ZIP Model with Standard Normal Link Function

Next, consider the ZIP model in which the probability $\psi_i$ is expressed by a standard normal link function: $\psi_i = \Phi(z_i' \gamma)$. The log-likelihood function is

$$
\mathcal{L} = \sum_{\{i: y_i=0\}} \ln \left\{ \Phi(z_i' \gamma) + \left[ 1 - \Phi(z_i' \gamma) \right] \exp(-\exp(x_i' \beta)) \right\} 
+ \sum_{\{i: y_i>0\}} \left\{ \ln \left[ (1 - \Phi(z_i' \gamma)) \right] - \exp(x_i' \beta) + y_i x_i' \beta - \sum_{k=2}^{y_i} \ln(k) \right\}
$$

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

$$
P(Y_i = y_i | x_i, z_i) = \frac{1}{Z(\lambda_i, v_i)} \frac{\lambda_i^{y_i}}{(y_i!)^{v_i}}, \quad y_i = 0, 1, 2, \ldots
$$

where the normalization factor is

$$
Z(\lambda, v) = \sum_{n=0}^{\infty} \frac{\lambda^n}{(n!)^v}
$$

and

$$
\lambda_i = \exp(x_i' \beta) \\
v_i = -\exp(g_i' \delta)
$$
Chapter 8: The CNTSELECT Procedure

The zero-inflated Conway-Maxwell-Poisson model can be written as

\[
P(y_i | x_i, z_i) = \begin{cases} 
F_i & y_i = 0 \\
(1 - F_i) \frac{1}{Z(\lambda_i, v_i)} \lambda_i^{y_i} & y_i > 0
\end{cases}
\]

The conditional expectation and conditional variance of \( y_i \) are given respectively by

\[
E(y_i | x_i, z_i) = (1 - F_i) \frac{1}{Z(\lambda_i, v_i)} \sum_{j=0}^{\infty} \frac{j^\lambda}{(j!)^v} \\
V(y_i | x_i, z_i) = (1 - F_i) \frac{1}{Z(\lambda_i, v_i)} \sum_{j=0}^{\infty} \frac{j^2 \lambda^j}{(j!)^v} - E(y_i | x_i, z_i)^2
\]

The general form of the log-likelihood function for the Conway-Maxwell-Poisson zero-inflated model is

\[
\mathcal{L} = \sum_{i=1}^{N} w_i \ln [P(y_i | x_i, z_i)]
\]

Zero-Inflated Conway-Maxwell-Poisson Model with Logistic Link Function

For this model, the probability \( \varphi_i \) is expressed by using a logistic link function as

\[
\varphi_i = \Lambda(z_i' \gamma) = \frac{\exp(z_i' \gamma)}{1 + \exp(z_i' \gamma)}
\]

The log-likelihood function is

\[
\mathcal{L} = \sum_{\{i: y_i = 0\}} w_i \ln \left\{ \Lambda(z_i' \gamma) + \left[ 1 - \Lambda(z_i' \gamma) \right] \frac{1}{Z(\lambda_i, v_i)} \right\} \\
+ \sum_{\{i: y_i > 0\}} w_i \{ \ln \left[ (1 - \Lambda(z_i' \gamma)) \right] - \ln(Z(\lambda, v)) + (y_i \ln(\lambda) - v \ln(y_i!)) \}
\]

Zero-Inflated Conway-Maxwell-Poisson Model with Normal Link Function

For this model, the probability \( \varphi_i \) is specified by using the standard normal distribution function (probit function): \( \varphi_i = \Phi(z_i' \gamma) \).

The log-likelihood function is written as

\[
\mathcal{L} = \sum_{\{i: y_i = 0\}} w_i \ln \left\{ \Phi(z_i' \gamma) + \left[ 1 - \Phi(z_i' \gamma) \right] \frac{1}{Z(\lambda_i, v_i)} \right\} \\
+ \sum_{\{i: y_i > 0\}} w_i \{ \ln \left[ (1 - \Phi(z_i' \gamma)) \right] - \ln(Z(\lambda, v)) + (y_i \ln(\lambda) - v \ln(y_i!)) \}
\]
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

\[
P(y_i = 0|x_i, z_i) = F_i + (1 - F_i) (1 + \alpha \mu_i)^{-\alpha^{-1}}
\]

\[
P(y_i|x_i, z_i) = (1 - F_i) \frac{\Gamma(y_i + \alpha^{-1})}{y_i!\Gamma(\alpha^{-1})} \left( \frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \times \left( \frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}, \quad y_i > 0
\]

In this case, the conditional expectation \( E \) and conditional variance \( V \) of \( y_i \) are

\[
E(y_i|x_i, z_i) = \mu_i (1 - F_i)
\]

\[
V(y_i|x_i, z_i) = E(y_i|x_i, z_i) [1 + \mu_i (F_i + \alpha)]
\]

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 \( \varphi_i \) is given by the logistic function, namely

\[
\varphi_i = \frac{\exp(z'_i \gamma)}{1 + \exp(z'_i \gamma)}
\]

The log-likelihood function is

\[
\mathcal{L} = \sum_{\{i: y_i = 0\}} \ln \left[ \exp(z'_i \gamma) + (1 + \alpha \exp(z'_i \beta))^{-\alpha^{-1}} \right] + \sum_{\{i: y_i > 0\}} \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) + \sum_{\{i: y_i > 0\}} \left\{ -\ln(y_i!) - (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(z'_i \beta)) + y_i \ln(\alpha) + y_i z'_i \beta \right\} - \sum_{i=1}^{N} \ln \left[ 1 + \exp(z'_i \gamma) \right]
\]
ZINB Model with Standard Normal Link Function

For this model, the probability \( \varphi_i \) is expressed by the standard normal distribution function (probit function):

\[ \varphi_i = \Phi(z_i' \gamma) \]

The log-likelihood function is

\[
\begin{align*}
\mathcal{L} &= \sum_{\{i: y_i = 0\}} \ln \left\{ \Phi(z_i' \gamma) + \left[ 1 - \Phi(z_i' \gamma) \right] (1 + \alpha \exp(x_i' \beta))^{-\alpha^{-1}} \right\} \\
&+ \sum_{\{i: y_i > 0\}} \ln \left[ 1 - \Phi(z_i' \gamma) \right] \\
&+ \sum_{\{i: y_i > 0\}} \sum_{j=0}^{y_i-1} \{ \ln(j + \alpha^{-1}) \} \\
&- \sum_{\{i: y_i > 0\}} \ln(y_i!) \\
&- \sum_{\{i: y_i > 0\}} (y_i + \alpha^{-1}) \ln(1 + \alpha \exp(x_i' \beta)) \\
&+ \sum_{\{i: y_i > 0\}} y_i \ln(\alpha) \\
&+ \sum_{\{i: y_i > 0\}} y_i x_i' \beta
\end{align*}
\]

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, which 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 to refer to its associated parameter. Suppose your model is

\[
\text{model } y = x1 \ x2 \ x5;
\]

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:

\[
\text{RESTRICT } x5 > 1.7;
\]
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```latex
\begin{verbatim}
class C;
model y = x1 x2 C*x3;
\end{verbatim}
```

The interaction between the continuous variable \( x_3 \) and the classification variable \( C \) introduces three additional parameters, which are named \( x_3_{-}C_0 \), \( x_3_{-}C_1 \), and \( x_3_{-}C_2 \). Note how, although the order of the terms in the interaction is \( C \) followed by \( x_3 \), the name of the parameter associated with the interaction is formed by placing the name of the continuous variable \( x_3 \) 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:

```latex
\begin{verbatim}
class C;
model y = x1 x2 x3(C);
\end{verbatim}
```

The nested interaction between the continuous variable \( x_3 \) and the classification variable \( C \) introduces three additional parameters, which are named \( x_3_{-}C_{-}0 \), \( x_3_{-}C_{-}1 \), and \( x_3_{-}C_{-}2 \). 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:

```latex
\begin{verbatim}
class D;
model y = x1 x2 D;
\end{verbatim}
```

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:

```latex
\begin{verbatim}
RESTRICT D_{-}1 < 0.4;
\end{verbatim}
```

### 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 COUNTREG 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:

```plaintext
   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:

```plaintext
   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 COUNTREG automatically
adds the _Alpha parameter to your model.

If no ERRORCOMP= option is specified, for zero-inflated binomial and negative binomial models, PROC
COUNTREG adds the _Alpha parameter to the model. If ERRORCOMP=RANDOM is specified for the
zero-inflated binomial and negative binomial models, then PROC COUNTREG 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:

```plaintext
   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 set 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 set 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 \( 8 \times (p + p \times (p + 1)/2) \) 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 245.

### 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 set, if any
- data set 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 Set

The OUTPUT statement creates a new SAS data set that contains various estimates that you specify. You can request that the output data set contain the estimates of $x_i^\beta$, 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 set contain the estimates of $z_i^\gamma$, 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 set contains estimates of $g_i^\delta$, $\lambda$, $\nu$, $\mu$, 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 set, 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 set that is created by the OUTPUT statement does not contain the variables in the input data set.

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 sets. These table names are listed in Table 8.4.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassInfo</td>
<td>Level information from the CLASS statement</td>
<td>CLASS</td>
<td>Default</td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>Correlation</td>
<td>Correlation of parameter estimates</td>
<td>MODEL</td>
<td>CORRB</td>
</tr>
<tr>
<td>Covariance</td>
<td>Covariance of parameter estimates</td>
<td>MODEL</td>
<td>COVB</td>
</tr>
<tr>
<td>DepVarSummary</td>
<td>Summary of dependent variable statistics</td>
<td>PROC</td>
<td>DEPVARSUMMARY</td>
</tr>
<tr>
<td>FitModelSummary</td>
<td>Summary of nonlinear estimation</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Parameter estimates</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>TestResults</td>
<td>Hypothesis test statistics</td>
<td>TEST</td>
<td>Default</td>
</tr>
<tr>
<td>TimingDetails</td>
<td>Detailed summary of time taken for all phases of execution</td>
<td>PROC</td>
<td>PRINTTIMING=(DETAILS)</td>
</tr>
<tr>
<td>TimingSummary</td>
<td>Summary of time taken for main phases of execution</td>
<td>PROC</td>
<td>PRINTTIMING</td>
</tr>
</tbody>
</table>
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.

```sas
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:

```plaintext
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**

<table>
<thead>
<tr>
<th>Class Level Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>group</td>
</tr>
</tbody>
</table>

**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.

**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         | 0.000000       | 0.0000  | <.0001      |          |
| x1            | 1  | 0.29663  | 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      |          |
| ln Intercept  | 1  | -0.969211| 0.028573       | -33.92  | <.0001      |          |
| ln x1         | 1  | -0.609492| 0.029128       | -20.92  | <.0001      |          |
| ln x2         | 1  | 0.298572 | 0.027039       | 11.04   | <.0001      |          |
| ln x3         | 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**


# Chapter 9
## The CPANEL Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: CPANEL Procedure</td>
<td>286</td>
</tr>
<tr>
<td>PROC CPANEL Features</td>
<td>287</td>
</tr>
<tr>
<td>PROC CPANEL Compared with Other SAS Procedures</td>
<td>288</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>290</td>
</tr>
<tr>
<td>Getting Started: CPANEL Procedure</td>
<td>290</td>
</tr>
<tr>
<td>Syntax: CPANEL Procedure</td>
<td>292</td>
</tr>
<tr>
<td>Functional Summary</td>
<td>293</td>
</tr>
<tr>
<td>PROC CPANEL Statement</td>
<td>294</td>
</tr>
<tr>
<td>BY Statement</td>
<td>295</td>
</tr>
<tr>
<td>CLASS Statement</td>
<td>295</td>
</tr>
<tr>
<td>COMPARE Statement</td>
<td>296</td>
</tr>
<tr>
<td>CORRELATED Statement</td>
<td>298</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>298</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>299</td>
</tr>
<tr>
<td>ENDOGENOUS Statement</td>
<td>300</td>
</tr>
<tr>
<td>ID Statement</td>
<td>301</td>
</tr>
<tr>
<td>INSTRUMENTS Statement</td>
<td>302</td>
</tr>
<tr>
<td>MODEL Statement</td>
<td>303</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>307</td>
</tr>
<tr>
<td>PREDETERMINED Statement</td>
<td>308</td>
</tr>
<tr>
<td>RESTRICT Statement</td>
<td>309</td>
</tr>
<tr>
<td>TEST Statement</td>
<td>310</td>
</tr>
<tr>
<td>Details: CPANEL Procedure</td>
<td>311</td>
</tr>
<tr>
<td>Specifying the Input Data</td>
<td>311</td>
</tr>
<tr>
<td>Specifying the Regression Model</td>
<td>311</td>
</tr>
<tr>
<td>Missing Values</td>
<td>312</td>
</tr>
<tr>
<td>Unbalanced Data</td>
<td>312</td>
</tr>
<tr>
<td>Common Notation</td>
<td>312</td>
</tr>
<tr>
<td>Pooled Regression (POOLED Option)</td>
<td>313</td>
</tr>
<tr>
<td>Between-Groups Regression (BTWNG Option)</td>
<td>313</td>
</tr>
<tr>
<td>One-Way Fixed-Effects Model (FIXONE Option)</td>
<td>313</td>
</tr>
<tr>
<td>One-Way Fixed-Effects Model, First Differencing (FDONE Option)</td>
<td>314</td>
</tr>
<tr>
<td>Two-Way Fixed-Effects Model (FIXTWO Option)</td>
<td>315</td>
</tr>
<tr>
<td>One-Way Random-Effects Model (RANONE Option)</td>
<td>316</td>
</tr>
<tr>
<td>Two-Way Random-Effects Model (RANTWO Option)</td>
<td>318</td>
</tr>
<tr>
<td>Hausman-Taylor Estimation (HTAYLOR Option)</td>
<td>322</td>
</tr>
</tbody>
</table>
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, and the one-way random-effects model. For the one-way random-effects model, PROC CPANEL uses methods that were developed by Baltagi (1981).

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 63 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. The PANEL procedure also provides alternative estimators of variance such as heteroscedastic and autocorrelation-consistent (HAC). Such functionality is 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.

Modeling Only Time Effects

The PANEL procedure and the HPPANEL procedure provide the following MODEL statement options for estimating models that contain time effects and no cross-sectional effects:

- FIXONETIME for one-way fixed time effects
- BTWNT for between-groups regression, grouped by time value

You can obtain the same functionality with the CPANEL procedure by setting your time variable as the cross section ID variable, and by leaving the time series ID variable unspecified. The following statements perform estimation equivalent to the FIXONETIME option:

```plaintext
proc cpanel data = mycas.a;
   id time;
   model y = x1 x2 x3 / fixone;
run;
```
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:

```sas
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:

```sas
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:

```sas
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 1965 7.40245 2.09519 2 1970 7.82644 2.39480
  3 1955 8.07153 2.94628 3 1960 8.47679 3.25967
;

Consider the model
\[ C_{it} = \beta_0 + \beta_1 P_{it} + v_i + e_{it} \]
for \( i = 1, \ldots, N \) and \( t = 1, \ldots, T \)

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:

```plaintext
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

<table>
<thead>
<tr>
<th>Model Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Estimation Method</strong></td>
</tr>
<tr>
<td><strong>Data Set</strong></td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
</tr>
<tr>
<td>Time Series Length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE</td>
</tr>
<tr>
<td>MSE</td>
</tr>
<tr>
<td>R-Square</td>
</tr>
</tbody>
</table>
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:

```plaintext
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 ... > ;
```
Table 9.1 summarizes the statements and options that the CPANEL procedure supports.

### Table 9.1  Functional Summary

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Set Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data set</td>
<td>PROC CPANEL</td>
<td>DATA=</td>
</tr>
<tr>
<td>Prevents partitioning of data by cross sections</td>
<td>PROC CPANEL</td>
<td>NOPART</td>
</tr>
<tr>
<td>Specifies the name of an output SAS data table</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td><strong>Variable Role Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies the classification variables</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>Specifies the cross section and time ID variables</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>Declares correlated variables or effects</td>
<td>CORRELATED</td>
<td></td>
</tr>
<tr>
<td>Declares endogenous variables or effects</td>
<td>ENDOGENOUS</td>
<td></td>
</tr>
<tr>
<td>Declares instrumental variables or effects</td>
<td>INSTRUMENTS</td>
<td></td>
</tr>
<tr>
<td>Declares predetermined variables or effects</td>
<td>PREDETERMINED</td>
<td></td>
</tr>
<tr>
<td><strong>Printing Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td>Prints correlations of the estimates</td>
<td>MODEL</td>
<td>CORRB</td>
</tr>
<tr>
<td>Prints covariances of the estimates</td>
<td>MODEL</td>
<td>COVB</td>
</tr>
<tr>
<td>Suppresses parameter labels from output tables</td>
<td>MODEL</td>
<td>NOLABEL</td>
</tr>
<tr>
<td>Suppresses printed output</td>
<td>PROC CPANEL</td>
<td>NOPRINT</td>
</tr>
<tr>
<td>Prints all available output</td>
<td>PROC CPANEL</td>
<td>PRINTALL</td>
</tr>
<tr>
<td>Prints fixed effects</td>
<td>MODEL</td>
<td>PRINTFIXED</td>
</tr>
<tr>
<td><strong>Model Estimation Technique Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the Amemiya-MacCurdy model</td>
<td>MODEL</td>
<td>AMACURDY</td>
</tr>
<tr>
<td>Specifies the between-groups model</td>
<td>MODEL</td>
<td>BTWNG</td>
</tr>
<tr>
<td>Specifies the dynamic panel model via first-differencing GMM</td>
<td>MODEL</td>
<td>DYNDIFF</td>
</tr>
<tr>
<td>Specifies the dynamic panel model via system GMM</td>
<td>MODEL</td>
<td>DYN SYS</td>
</tr>
<tr>
<td>Specifies the one-way fixed-effects model via first-differencing</td>
<td>MODEL</td>
<td>FDONE</td>
</tr>
<tr>
<td>Specifies the one-way fixed-effects model</td>
<td>MODEL</td>
<td>FIXONE</td>
</tr>
<tr>
<td>Specifies the two-way fixed-effects model</td>
<td>MODEL</td>
<td>FIXTWO</td>
</tr>
<tr>
<td>Specifies the Hausman-Taylor model</td>
<td>MODEL</td>
<td>HTAYLOR</td>
</tr>
</tbody>
</table>
Table 9.1 continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the between-groups model with instrumental variables</td>
<td>MODEL</td>
<td>IVBTWNG</td>
</tr>
<tr>
<td>Specifies the one-way fixed-effects model with instrumental variables</td>
<td>MODEL</td>
<td>IVFIXONE</td>
</tr>
<tr>
<td>Specifies the pooled regression model with instrumental variables</td>
<td>MODEL</td>
<td>IVPOOLED</td>
</tr>
<tr>
<td>Specifies the one-way random-effects model with instrumental variables</td>
<td>MODEL</td>
<td>IVRANONE</td>
</tr>
<tr>
<td>Specifies the pooled regression model</td>
<td>MODEL</td>
<td>POOLED</td>
</tr>
<tr>
<td>Specifies the one-way random-effects model</td>
<td>MODEL</td>
<td>RANONE</td>
</tr>
<tr>
<td>Specifies the two-way random-effects model</td>
<td>MODEL</td>
<td>RANTWO</td>
</tr>
</tbody>
</table>

Model Estimation Control Options

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets the number of AR($m$) tests</td>
<td>MODEL</td>
<td>ARTESTS=</td>
</tr>
<tr>
<td>Specifies bias-corrected variances for two-step GMM</td>
<td>MODEL</td>
<td>BIASCORRECTED</td>
</tr>
<tr>
<td>Sets the number of dependent variable lags</td>
<td>MODEL</td>
<td>DLAGS=</td>
</tr>
<tr>
<td>Specifies the generalized inverse type</td>
<td>MODEL</td>
<td>GINV=</td>
</tr>
<tr>
<td>Specifies one-step or two-step GMM</td>
<td>MODEL</td>
<td>GMM=</td>
</tr>
<tr>
<td>Limits the number of GMM instruments</td>
<td>MODEL</td>
<td>MAXBAND=</td>
</tr>
<tr>
<td>Suppresses the intercept term</td>
<td>MODEL</td>
<td>NOINT</td>
</tr>
<tr>
<td>Specifies robust variances for GMM</td>
<td>MODEL</td>
<td>ROBUST</td>
</tr>
<tr>
<td>Specifies the method for the variance components estimator</td>
<td>MODEL</td>
<td>VCOMP=</td>
</tr>
</tbody>
</table>

Tests and Restrictions

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies linear equality restrictions on the parameters</td>
<td>RESTRICT</td>
</tr>
<tr>
<td>Performs tests of linear hypotheses</td>
<td>TEST</td>
</tr>
</tbody>
</table>

Model Comparison Statements

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create tables that display side-by-side model comparisons</td>
<td>COMPARE</td>
</tr>
</tbody>
</table>

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 290.

`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 303.

---

**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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
</tbody>
</table>

**COMPARE Statement**

```plaintext
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 303. 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:

```plaintext
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:
**COMPARE Statement**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>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.</td>
</tr>
<tr>
<td>DFE</td>
<td>Displays the error degrees of freedom. This statistic is displayed by default.</td>
</tr>
<tr>
<td>F</td>
<td>Displays the $F$ statistic of the overall test for no fixed effects.</td>
</tr>
<tr>
<td>FNUMDF</td>
<td>Displays the numerator degrees of freedom of the overall test for no fixed effects.</td>
</tr>
<tr>
<td>FDENDF</td>
<td>Displays the denominator degrees of freedom of the overall test for no fixed effects.</td>
</tr>
<tr>
<td>M</td>
<td>Displays the Hausman test $m$ statistic.</td>
</tr>
<tr>
<td>MDF</td>
<td>Displays the Hausman test degrees of freedom.</td>
</tr>
<tr>
<td>MSE</td>
<td>Displays the model mean square error. This statistic is displayed by default.</td>
</tr>
<tr>
<td>NCS</td>
<td>Displays the number of cross sections. This statistic is displayed by default.</td>
</tr>
<tr>
<td>NOBS</td>
<td>Displays the number of observations. This statistic is displayed by default.</td>
</tr>
<tr>
<td>NONE</td>
<td>Suppresses the table of model fit statistics when specified alone, and is ignored when specified with other options.</td>
</tr>
<tr>
<td>NTS</td>
<td>Displays the maximum length of the time series. This statistic is displayed by default.</td>
</tr>
<tr>
<td>PROBF</td>
<td>Displays the significance level of the overall test for no fixed effects.</td>
</tr>
<tr>
<td>PROBM</td>
<td>Displays the significance level of the Hausman test.</td>
</tr>
<tr>
<td>RMSE</td>
<td>Displays the model root mean square error.</td>
</tr>
<tr>
<td>RSQUARE</td>
<td>Displays the model R-square fit statistic. This statistic is displayed by default.</td>
</tr>
<tr>
<td>SSE</td>
<td>Displays the model sum of squares.</td>
</tr>
<tr>
<td>VARCS</td>
<td>Displays the variance component that corresponds to cross sections in random-effects models.</td>
</tr>
<tr>
<td>VARERR</td>
<td>Displays the variance component that corresponds to error in random-effects models.</td>
</tr>
<tr>
<td>VARTS</td>
<td>Displays the variance component that corresponds to time series in random-effects models.</td>
</tr>
</tbody>
</table>

**OUTPARM=** `CAS-libref.data-table`

Names an output data set 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 290.

**OUTSTAT=** `CAS-libref.data-table`

Names an output data set 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 290.
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 \( X_1, X_2, Z_1, \) and \( Z_2 \). The
effects \( X_2 \) and \( Z_2 \) are correlated with the cross-sectional (firm-level) error term.

```plaintext
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 47
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 341. 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 subsetted 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 341. 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`, `IVPOOLED`, and `IVRANONE` options in the `MODEL` statement.

When you use the `ENDOGENOUS` statement for IV variables 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 DYN SYS 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 (/):

\textbf{EQ=BOTH | DIFF | LEVEL}

specifies which equations include the endogenous instruments. You can specify the following values:

\textbf{BOTH} includes the endogenous instruments in both the difference and level equations.
\textbf{DIFF} includes the endogenous instruments only in the difference equations.
\textbf{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 \texttt{Sales} and standard instruments for the exogenous variable \texttt{Price}. Both sets of instruments are included only in the difference equations.

\begin{verbatim}
proc cpanel data = mycas.a;
  id firm year;
  model Sales = Price / dyndiff;
  endogenous Sales / eq = diff;
  instruments Price / eq = diff;
run;
\end{verbatim}

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 47 in Chapter 3, “Shared Concepts.”

ID Statement

\textbf{ID cross-section-id < time-series-id> ;}

The ID statement specifies variables in the input data set that identify the cross section and time period for each observation. Sometimes estimation does not require a time variable, in which case specifying the \texttt{time-series-id} variable is optional.
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, IVPOOLED, and IVRANONE 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 DYNSYS 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 47 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 47 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Estimation Technique Options</strong></td>
<td></td>
</tr>
<tr>
<td>AMACURDY</td>
<td>Fits a one-way model by using the Amemiya-MacCurdy estimator</td>
</tr>
<tr>
<td>BTWNG</td>
<td>Fits the between-groups model</td>
</tr>
<tr>
<td>DYNDIFF</td>
<td>Fits a dynamic panel model by using first-differences GMM</td>
</tr>
<tr>
<td>DYNNSYS</td>
<td>Fits a dynamic panel model by using system GMM</td>
</tr>
<tr>
<td>FDONE</td>
<td>Fits a one-way model by using first differences</td>
</tr>
<tr>
<td>FIXONE</td>
<td>Fits a one-way fixed-effects model</td>
</tr>
<tr>
<td>FIXTWO</td>
<td>Fits a two-way fixed-effects model</td>
</tr>
<tr>
<td>HTAYLOR</td>
<td>Fits a one-way model by using the Hausman-Taylor estimator</td>
</tr>
</tbody>
</table>
### Table 9.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVBTWNG</td>
<td>Fits a between-groups regression model by using instrumental variables</td>
</tr>
<tr>
<td>IVFIXONE</td>
<td>Fits a one-way fixed-effects model by using instrumental variables</td>
</tr>
<tr>
<td>IVPOOLED</td>
<td>Fits the pooled regression model by using instrumental variables</td>
</tr>
<tr>
<td>IVRANONE</td>
<td>Fits a one-way random-effects model by using instrumental variables</td>
</tr>
<tr>
<td>POOLED</td>
<td>Fits the pooled regression model</td>
</tr>
<tr>
<td>RANONE</td>
<td>Fits a one-way random-effects model</td>
</tr>
<tr>
<td>RANTWO</td>
<td>Fits a two-way random-effects model</td>
</tr>
</tbody>
</table>

#### Estimation Control Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOINT</td>
<td>Suppresses the intercept</td>
</tr>
<tr>
<td>VCOMP=</td>
<td>Specifies the type of variance component estimation for random-effects estimation</td>
</tr>
</tbody>
</table>

#### Dynamic Panel Estimation Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTESTS=</td>
<td>Specifies the maximum lag of AR(m) tests</td>
</tr>
<tr>
<td>BIASCORRECTED</td>
<td>Computes bias-corrected two-step GMM variances</td>
</tr>
<tr>
<td>DLAGS=</td>
<td>Specifies the number of dependent variable lags to include</td>
</tr>
<tr>
<td>GINV=</td>
<td>Specifies the generalized inverse method</td>
</tr>
<tr>
<td>GMM=</td>
<td>Specifies one-step or two-step GMM</td>
</tr>
<tr>
<td>MAXBAND=</td>
<td>Limits the number of instruments</td>
</tr>
<tr>
<td>ROBUST</td>
<td>Computes robust variances</td>
</tr>
</tbody>
</table>

#### Printed Output Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORRB</td>
<td>Prints the parameter correlation matrix</td>
</tr>
<tr>
<td>COVB</td>
<td>Prints the parameter covariance matrix</td>
</tr>
<tr>
<td>NOLABEL</td>
<td>Suppresses variable labels</td>
</tr>
<tr>
<td>PRINTFIXED</td>
<td>Estimates and prints the fixed effects</td>
</tr>
</tbody>
</table>

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 311.
AMACURDY
requests Amemiya-MacCurdy 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.

DYNDIFF
estimates a dynamic panel by using the generalized method of moments (GMM) on equations that are formed by first differencing.

DYNYSYS
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.

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 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.

IVFIXONE
requests instrumental variables 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.

IVPOOLED
requests instrumental variables pooled regression. You specify the endogenous effects by using the ENDOGENOUS statement, and you specify external instruments by using the INSTRUMENTS statement.

IVRANONE
requests instrumental variables 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.

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).

**NOINT**

suppresses the intercept parameter from the model.

**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.

**BIASCORRECTED**

computes the bias-corrected covariance matrix of the two-step generalized method of moments (GMM) estimator.

**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.

**GMM=ONESTEP | TWOSTEP**

specifies the number of GMM stages to use in the estimation. 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.

**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.

**ROBUST**

computes the robust covariance matrix of the generalized method of moments (GMM) estimator.

**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

- **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 290.

- **data-table** 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 set.

- **RESIDUAL=**name
  **R=**name
  writes the residuals to the output data set.

---

**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 **DYN SYS** 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.

```r
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 47 in Chapter 3, “Shared Concepts.”

---

**RESTRICT Statement**

```r
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:

```r
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:

```r
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.
Chapter 9: The CPANEL Procedure

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 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;
    id csid tsid;
    model y = x1 x2 x3;
    test x1 = 0, x2 * .5 + 2 * x3 = 0 / all;
    test intercept = 0, x3 = 0;
run;
```

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, DYNSYS, HTAYLOR, IVBTWNG, IVFIXONE, IVPOOLED, and IVRANONE, 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, FIXONE, FDONE, FIXTWO, RANONE, RANTWO, HTAYLOR, AMACURDY, IVPOOLED, IVBTWNG, IVFIXONE, IVRANONE, DYNDIFF, and DYNYSYS. The methods that underlie these estimation options are described, in order, beginning with the section “Pooled Regression (POOLED Option)” on page 313.

The following statements fit a one-way random-effects model with variance components estimated by the Fuller-Battese (FB) method:
**Missing Values**

Any observation in the input data set 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 set 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:

\[ y_{it} = \alpha + \sum_{k=1}^{K} x_{itk} \beta_k + u_{it} \quad i = 1, \ldots, N; \ t = 1, \ldots, T_i \]

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

$$y_{it} = \alpha + x_{it} \beta + u_{it}$$

where $x_{it}$ is a $1 \times K$ row vector of independent variables and $\beta$ is the $K \times 1$ vector of coefficients. Let $y$ and $X$ be matrices that are formed by arranging the dependent and independent variables by cross section, and by time within each cross section. Let $X_\alpha$ be the $X$ matrix augmented by a first column of ones, which corresponds to the intercept term $\alpha$.

Define the following utility matrices:

- $I_p$ is an identity matrix of dimension $p$.
- $j_p$ is a $p \times 1$ column vector of ones.
- $J_p = j_p j_p'$ is a matrix of ones of dimension $p$.
- $\bar{J}_p = p^{-1} J_p$.
- $E_p = I_p - \bar{J}_p$.

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 $\sigma_e^2$.

---

**Between-Groups Regression (BTWNG Option)**

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.

---

**One-Way Fixed-Effects Model (FIXONE Option)**

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 \( \sigma_e^2 \).

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 \( \beta \) and not \( v_i \).

Let \( Q_0 = \text{diag}(E_{T_i}) \). The matrix \( Q_0 \) represents the within transformation, the conversion of the raw data to deviations from a cross section’s mean. Let \( X_w = Q_0 \mathbf{X} \) and \( y_w = Q_0 \mathbf{y} \). The within estimator of \( \beta \) is

\[
\hat{\beta}_w = (X_w'X_w)^{-1}X_w'y_w
\]

The previous estimation does not involve the intercept term because \( \hat{\beta}_w \) is the same whether or not the intercept \( \alpha \) 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 \( y_w \) on \( X_w \). 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

\[
\hat{\alpha} = \bar{y}_i - \bar{x}_i \hat{\alpha}_w
\]

where \( \bar{y}_i \) is the overall mean of \( y_{it} \) and \( \bar{x}_i \) 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

\[
\hat{v}_i = \bar{y}_i - \bar{x}_i \hat{\alpha}_w
\]

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 \( \hat{\alpha} \).

---

### 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 \( i = 1, \ldots, N \) and \( t = 2, \ldots, T_i \). You obtain the first-differenced estimator, \( \hat{\beta}_d \), and its variance by performing OLS regression of \( \Delta y_{it} \) on \( \Delta x_{it} \).

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 \( \sigma_e^2 \) 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 \( \alpha \) and \( v_i \) are identical to that described in the section “One-Way Fixed-Effects Model (FIXONE Option)” on page 313, 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 \( \lambda_t \) 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 \( X^* \) and \( y^* \) be versions of \( X \) and \( y \) whose rows are sorted by time period, and by cross section within each time period. With the data sorted in this manner, define \( D_N \) to be the \( M \times N \) design matrix for cross sections. Each row of \( D_N \) contains a 1 in the column that corresponds to that observation’s cross section, and 0s in the remaining columns. Similarly, define \( D_T \) to be the \( M \times T \) design matrix for time periods. In balanced data, \( D_N D_T = I_T \otimes J_N \).

Define the following:

\[
\Delta_N = D_N' D_N \quad \quad (N \times N) \\
\Delta_T = D_T' D_T \quad \quad (T \times T) \\
A = D_T' D_N \quad \quad (T \times N) \\
\hat{D} = D_T - D_N \Delta_N^{-1} A' \quad \quad (M \times T) \\
Q = \Delta_T - A \Delta_N^{-1} A' \quad \quad (T \times T) \\
P = I_M - D_N \Delta_N^{-1} D_N' - \hat{D} Q^{-1} \hat{D}' \quad \quad (M \times M)
\]

The matrix \( P \) 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 \( Py^* \) on \( PX^* \) to obtain \( \hat{\beta}_f \), \( \text{Var}(\hat{\beta}_f) \), 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 \( r^* = y^* - X^* \hat{\beta}_f \). Estimates of the time effects are \( \hat{\lambda} = Q^{-1} \hat{D}' r^* \), and estimates of the cross-sectional effects are \( \hat{v} = (\Theta_1 - \Theta_2 + \Theta_3) r^* \), where

\[
\Theta_1 = \Delta_N^{-1} D_N' \\
\Theta_2 = \Delta_N^{-1} A' Q^{-1} D_T' \\
\Theta_3 = \Delta_N^{-1} A' Q^{-1} A \Delta_N^{-1} D_N'
\]

If the model includes an intercept, then \( \hat{\alpha} = \bar{y} - \bar{x} \hat{\beta}_f \), and the \( \hat{v}_i \) are shifted to exclude \( \hat{\alpha} \).

Variance and covariance estimates for \( \hat{\alpha}, \hat{v} \), and \( \hat{\lambda} \) are obtained by the delta method, because each of these quantities is a linear transformation of \( y^* \) and \( \hat{\beta}_f \).
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 \( \sigma_v^2 \), and the \( e_{it} \) are iid with zero mean and variance \( \sigma_e^2 \). Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with \( X \).

Estimation proceeds in two steps. First, you obtain estimates of the variance components \( \sigma_v^2 \) and \( \sigma_e^2 \). 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,
\[ w_i = \frac{1}{\hat{\sigma}_e^2 + \hat{\sigma}_e^2} \]
where \( \hat{\sigma}_e^2 = T_i \hat{\sigma}_v^2 + \hat{\sigma}_e^2 \). Taking \( \hat{\theta}_i \), you form the partial deviations:
\[ \tilde{y}_{it} = y_{it} - \hat{\theta}_i \bar{y}_i \]
\[ \tilde{x}_{\alpha, it} = x_{\alpha, it} - \hat{\theta}_i \tilde{x}_{\alpha, i} \]

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:
\[ q_e = u' Q_0 u \]
\[ q_v = u' P_0 u \]

In these equations, \( Q_0 = \text{diag}(E_T) \), \( P_0 = \text{diag}(J_T) \), and \( u \) is the vector of true residuals.

The ANOVA methods differ only in how they estimate \( u \). 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
\[ E\left( \hat{u}_w' Q_0 \hat{u}_w \right) = (M - N - K) \sigma_v^2 \]
\[ E\left( \hat{u}_b' P_0 \hat{u}_b \right) = \left[ M - \text{tr}\left\{ (X'_\alpha P_0 X_\alpha)^{-1} X'_\alpha Z_0 Z'_\alpha X_\alpha \right\} \right] \sigma_v^2 + (N - K - 1) \sigma_e^2 \]
where \( Z_0 = \text{diag}(J_T) \).

Estimates of \( \sigma_v^2 \) and \( \sigma_e^2 \) 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, \( \hat{u}_p \), in both quadratic forms.

The expected values of the quadratic forms are

\[
E \left( \hat{u}_p^* Q_0 \hat{u}_p \right) = (d_1 - d_3) \sigma_v^2 + (M - N - K - 1 + d_2) \sigma_e^2 \\
E \left( \hat{u}_p^* P_0 \hat{u}_p \right) = (M - 2d_1 + d_3) \sigma_v^2 + (N - d_2) \sigma_e^2
\]

where

\[
d_1 = \text{tr} \left\{ \left( X'_\alpha X_\alpha \right)^{-1} X'_\alpha Z_0 Z'_0 X_\alpha \right\} \\
d_2 = \text{tr} \left\{ \left( X'_\alpha X_\alpha \right)^{-1} X'_\alpha P_0 X_\alpha \right\} \\
d_3 = \text{tr} \left\{ \left( X'_\alpha X_\alpha \right)^{-1} X'_\alpha P_0 X_\alpha \left( X'_\alpha X_\alpha \right)^{-1} X'_\alpha Z_0 Z'_0 X_\alpha \right\}
\]

**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, \( \hat{u}_w \), in both quadratic forms.

The expected values of the quadratic forms are

\[
E \left( \hat{u}_w^* Q_0 \hat{u}_w \right) = (M - N - K) \sigma_v^2 \\
E \left( \hat{u}_w^* P_0 \hat{u}_w \right) = (N - 1 + d) \sigma_v^2 + \left( M - M^{-1} \sum_{i=1}^{N} T_i^2 \right) \sigma_e^2
\]

where

\[
d = \text{tr} \left\{ (X'Q_0 X)^{-1} X' P_0 \right\} - \text{tr} \left\{ (X'Q_0 X)^{-1} X' \tilde{J}_M X \right\}
\]

**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

\[
\hat{\sigma}_v^2 = \frac{R(v|\beta) - (N - 1) \hat{\sigma}_e^2}{M - \text{tr}\{Z_0 X_\alpha (X'_\alpha X_\alpha)^{-1} X'_\alpha Z_0\}}
\]
where
\[ R(v|\beta) = R(\beta|v) + R(v) - R(\beta) \]

for
\[ R(v) = y'Z_0(Z_0'Z_0)^{-1}Z_0'y \]
\[ R(\beta|v) = y'_wX'_w(X'_wX_w)^{-1}X'_wv_w \]
\[ R(\beta) = y'X'_\alpha(X'_\alphaX_\alpha)^{-1}X'_\alpha\beta \]

**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 \( \sigma^2_v \) as the sample variance of the cross-sectional effects, estimated from a one-way fixed-effects regression. Specifically, \( \hat{\sigma}^2_v = (N - 1)^{-1} \sum_{i=1}^{N} (\hat{\nu}_i - \bar{\nu})^2 \), where \( \bar{\nu} \) is the mean of the estimated fixed effects. You estimate \( \sigma^2_e \) 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 \( \sigma^2_e \). If your goal is inference on \( \beta \), then the variance-component method will matter little.

The methods have varying performance in how they estimate \( \sigma^2_v \), the cross-sectional variance. All five methods tend to perform poorly if either the data are severely unbalanced or the ratio \( \sigma^2_v / \sigma^2_e \) is much greater than 1.

The Nerlove method is the only method that guarantees a nonnegative estimate of \( \sigma^2_v \); the other four methods reset a negative estimate to 0. However, the Nerlove method is particularly unsuitable for unbalanced data because the sample variance 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 \( \sigma^2_v \), the \( \lambda_t \) are iid with zero mean and variance \( \sigma^2_\lambda \), and the \( e_{it} \) are iid with zero mean and variance \( \sigma^2_e \). Furthermore, a random-effects specification assumes that the error terms are mutually uncorrelated and that each error term is uncorrelated with \( X \).

Estimation proceeds in two steps. First, you obtain estimates of the variance components \( \sigma^2_v, \sigma^2_\lambda, \) and \( \sigma^2_e \). 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, y and X are treated as being sorted first by time, and then by cross-section within time. For the definitions of the design matrices $D_N$ and $D_T$, see the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 315. The variance of $y$ is

$$\Omega = \sigma^2_I M + \sigma^2_v D_N D_N' + \sigma^2_\lambda D_T D_T'$$

and estimation proceeds as OLS regression of $\hat{\sigma}^2_\Omega^{-1/2} y$ on $\hat{\sigma}^2_\Omega^{-1/2} x_\alpha$.

Rather than invert the $M \times M$ matrix $\hat{\Omega}$ directly, Wansbeek and Kapteyn (1989) provide the more convenient form

$$\hat{\sigma}^2_\Omega^{-1} = V - V D_T \tilde{P}^{-1} D_T' V$$

where

$$V = I_M - D_N \hat{\Delta}_N^{-1} D_N'$$

$$\tilde{P} = \tilde{\Delta}_T - D_T' D_N \hat{\Delta}_N^{-1} D_N' D_T$$

with $\hat{\Delta}_N = D_N' D_N + (\hat{\sigma}^2_\nu / \hat{\sigma}^2_\lambda) I_N$ and $\hat{\Delta}_T = D_T' D_T + (\hat{\sigma}^2_\nu / \hat{\sigma}^2_\lambda) I_T$.

If the data are balanced, then the calculations are simplified considerably—the data are transformed from $z_{i,t}$ to $z_{i,t} = \hat{\theta}_1 \hat{z}_i - \hat{\theta}_2 \hat{z}_t + \hat{\theta}_3 \hat{z}_{i,t}$, where

$$\hat{\theta}_1 = 1 - \hat{\sigma}_e (T \hat{\sigma}^2_\nu + \hat{\sigma}^2_\epsilon) ^{-1/2}$$

$$\hat{\theta}_2 = 1 - \hat{\sigma}_e (N \hat{\sigma}^2_\lambda + \hat{\sigma}^2_\epsilon) ^{-1/2}$$

$$\hat{\theta}_3 = \hat{\theta}_1 + \hat{\theta}_2 + \hat{\sigma}_e (T \hat{\sigma}^2_\nu + N \hat{\sigma}^2_\lambda + \hat{\sigma}^2_\epsilon) ^{-1/2} - 1$$

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:

$$q_e = u' Pu$$

$$q_\lambda = u' D_T \Delta_T^{-1} D_T' u$$

$$q_\nu = u' D_N \Delta_N^{-1} D_N' u$$

The matrix $P$ is the two-way within transformation defined in the section “Two-Way Fixed-Effects Model (FIXTWO Option)” on page 315, $\Delta_T = D_T' D_T$, $\Delta_N = D_N' D_N$, and $u$ is the vector of true residuals.
The ANOVA methods differ only in how they estimate $u$. The Swamy-Arora method substitutes into $q_e$ the within residuals from two-way fixed-effects estimation, substitutes into $q_\lambda$ 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

$$E\left(\hat{u}_j' P \hat{u}_j \right) = (M - N - T - K + 1)\sigma_e^2$$

$$E\left(\hat{u}_\lambda' P \lambda \hat{u}_\lambda \right) = \left[ M - \text{tr} \left( (X'_\lambda P \lambda X_\lambda)^{-1} X'_\lambda D_T D_T' X_\lambda \right) \right]\sigma_\lambda^2 + (T - K - 1)\sigma_e^2$$

$$E\left(\hat{u}_v' P \nu \hat{u}_v \right) = \left[ M - \text{tr} \left( (X'_\nu P \nu X_\nu)^{-1} X'_\nu D_N D_N' X_\nu \right) \right]\sigma_v^2 + (N - K - 1)\sigma_e^2$$

where $P \lambda = D_T \Delta_T^{-1} D_T'$ and $P \nu = D_N \Delta_N^{-1} D_N'$.

Estimates of $\sigma_v^2$, $\sigma_\lambda^2$, and $\sigma_e^2$ 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 VCMP=WH option in the MODEL statement. The Wallace-Hussain method is an ANOVA method that uses the residuals from pooled (OLS) regression, $\hat{u}_p$, in all three quadratic forms.

The expected values of the quadratic forms are

$$E\left(\hat{u}_p' P \hat{u}_p \right) = d_{11}\sigma_e^2 + d_{12}\sigma_v^2 + d_{13}\sigma_\lambda^2$$

$$E\left(\hat{u}_\lambda' P \lambda \hat{u}_\lambda \right) = d_{21}\sigma_e^2 + d_{22}\sigma_v^2 + d_{23}\sigma_\lambda^2$$

$$E\left(\hat{u}_v' P \nu \hat{u}_v \right) = d_{31}\sigma_e^2 + d_{32}\sigma_v^2 + d_{33}\sigma_\lambda^2$$

Define $\Sigma = (X'_\alpha X_\alpha)^{-1}$, which is the inverse crossproducts matrix from pooled regression. Also define $S_v = X'_\nu D_N D_N' X_\alpha$ and $S_\lambda = X'_\lambda D_T D_T' X_\alpha$, which are the individual-level sum of squares and the time-level sum of squares, respectively. The coefficients are

$$d_{11} = M - N - T + 1 - \text{tr} \left( X'_\alpha P X_\alpha \Sigma \right)$$

$$d_{12} = \text{tr} \left( S_v \Sigma X'_\alpha P X_\alpha \Sigma \right)$$

$$d_{13} = \text{tr} \left( S_\lambda \Sigma X'_\alpha P X_\alpha \Sigma \right)$$

$$d_{21} = T - \text{tr} \left( X'_\lambda P \lambda X_\lambda \Sigma \right)$$

$$d_{22} = T - 2\text{tr} \left( X'_\lambda P \lambda D_N D_N' X_\alpha \Sigma \right) + \text{tr} \left( X'_\lambda P \lambda X_\alpha \Sigma S_\nu \Sigma \right)$$

$$d_{23} = M - 2\text{tr} \left( S_\lambda \Sigma \right) + \text{tr} \left( X'_\alpha P \lambda X_\alpha \Sigma S_\lambda \Sigma \right)$$

$$d_{31} = N - \text{tr} \left( X'_\alpha P \nu X_\alpha \Sigma \right)$$

$$d_{32} = M - 2\text{tr} \left( S_\nu \Sigma \right) + \text{tr} \left( X'_\nu P \alpha X_\alpha \Sigma S_\nu \Sigma \right)$$

$$d_{33} = N - 2\text{tr} \left( X'_\alpha P \nu D_T D_T' X_\alpha \Sigma \right) + \text{tr} \left( X'_\alpha P \alpha X_\alpha \Sigma S_\lambda \Sigma \right)$$
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 Wansbeek and Kapteyn method is an ANOVA method that uses the within residuals from two-way fixed effects, \( \hat{u}_f \), in all three quadratic forms.

The expected values of the quadratic forms are

\[
E\left( \hat{u}'_f P \hat{u}_f \right) = (M - N - T - K + 1)\sigma^2_e
\]

\[
E\left( \hat{u}'_f P_\lambda \hat{u}_f \right) = (T + k_N - k_0) \sigma^2_e + (T - \delta_N) \sigma^2_\nu + (M - \delta_T) \sigma^2_\lambda
\]

\[
E\left( \hat{u}'_f P_\nu \hat{u}_f \right) = (N + k_T - k_0) \sigma^2_e + (M - \delta_N) \sigma^2_\nu + (N - \delta_T) \sigma^2_\lambda
\]

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

\[
k_0 = 1 + M^{-1} j_M X(X'P)X^{-1}X'j_M
\]

\[
k_N = \text{tr}\{(X'P)X^{-1}X'P_\lambda X\}
\]

\[
k_T = \text{tr}\{(X'P)X^{-1}X'P_\nu X\}
\]

When the NOINT option is specified, the variance-component equations change slightly: \( k_0, \delta_N, \) and \( \delta_T \) 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 \( u \) is estimated by the two-way within residuals \( \hat{u}_f \). The other two quadratic forms, \( q_\lambda \) and \( q_\nu \), are replaced by the error sums of squares from one-way fixed-effects estimations.

The resulting system of equations is

\[
E\left( \hat{u}'_f P \hat{u}_f \right) = (M - N - T - K + 1)\sigma^2_e
\]

\[
E\left( \hat{u}'_\lambda \hat{u}_\lambda \right) = (M - T - K)\sigma^2_\nu + \left[ M - T - \text{tr}\left\{ X'W_\lambda D_N D'_N W_\lambda X(X'W_\lambda X)^{-1} \right\} \right] \sigma^2_\nu
\]

\[
E\left( \hat{u}'_\nu \hat{u}_\nu \right) = (M - N - K)\sigma^2_\nu + \left[ M - N - \text{tr}\left\{ X'W_\nu D_T D'_T W_\nu X(X'W_\nu X)^{-1} \right\} \right] \sigma^2_\lambda
\]

where \( W_\lambda = I_M - P_\lambda, W_\nu = I_M - P_\nu, \hat{u}_\lambda \) are the residuals from a one-way model with time fixed effects, and \( \hat{u}_\nu \) 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

\[ \hat{\sigma}_e^2 = M^{-1} \hat{u}_f' P \hat{u}_f \]

You obtain \( \hat{\sigma}_e^2 \) 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:

\[ y_{it} = x_{1it} \beta_1 + x_{2it} \beta_2 + z_{1i} \gamma_1 + z_{2i} \gamma_2 + v_i + e_{it} \]

The regressors are subdivided so that \( x_{1it} \) and \( x_{2it} \) vary within cross sections whereas \( z_{1i} \) and \( z_{2i} \) 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 \( z_{1i} \).

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 \( y_w, X_w, \) and \( \hat{\beta}_w \) from the section “One-Way Fixed-Effects Model (FIXONE Option)” on page 313. Then \( \hat{\sigma}_e^2 = SSE/(M - N) \), where

\[ SSE = (y_w - X_w \hat{\beta}_w)' (y_w - X_w \hat{\beta}_w) \]

To estimate the cross-sectional error variance, form the mean-residual vector \( r = P_0'(y - X_w \hat{\beta}_w) \), where \( P_0 = \text{diag}(J_{T_i}) \). You can use the mean residuals to obtain intermediate estimates of the coefficients for \( z_1 \) and \( z_2 \) via two-stage least squares (2SLS) estimation. At the first stage, use \( x_1 \) and \( z_1 \) as instrumental variables to predict \( z_2 \). At the second stage, regress \( r \) on both \( z_1 \) and the predicted \( z_2 \) to obtain \( \hat{y}_1^m \) and \( \hat{y}_2^m \).

To estimate the cross-sectional variance, compute \( \hat{\sigma}_v^2 = \{R(v)/N - \hat{\sigma}_\lambda^2\}/\hat{T} \), where \( \hat{T} = N/(\sum_{i=1}^N T_i^{-1}) \) and

\[ R(v) = (r - Z_1 \hat{y}_1^m - Z_2 \hat{y}_2^m)' (r - Z_1 \hat{y}_1^m - Z_2 \hat{y}_2^m) \]

The design matrices \( Z_1 \) and \( Z_2 \) are formed by stacking the data observations of \( z_{1i} \) and \( z_{2i} \), 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 \( x_{1it}^*, x_{2it}^*, z_{1i}^*, \) and \( z_{2i}^* \). The partial weights \( \hat{\theta}_i \) are determined by \( \hat{\theta}_i = 1 - \hat{\sigma}_e / \hat{\sigma}_i \), with \( \hat{\sigma}_i^2 = T_i \hat{\sigma}_e^2 + \hat{\sigma}_c^2 \).

Finally, you obtain the Hausman-Taylor estimates by performing 2SLS regression of \( y_{it}^* \) on \( x_{1it}^*, x_{2it}^*, z_{1i}^* \), and \( z_{2i}^* \). For the first-stage regression, use the following instruments:

- \( \bar{x}_{1it} \), 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
- \( (1 - \hat{\theta}_i) z_{1i} \)

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 \( x_1 \), and let \( g_2 \) equal the number of regressors in \( z_2 \). 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 336. However, the degrees of freedom of the test are not based on matrix rank but instead are equal to \( k_1 - g_2 \).

---

**Amemiya-MacCurdy Estimation (AMACURDY Option)**

You perform Amemiya-MacCurdy estimation by specifying the AMACURDY option in the MODEL statement. The Amemiya and MacCurdy (1986) model is similar to the Hausman-Taylor model. Following the development in the section “Hausman-Taylor Estimation (HTAYLOR Option)” on page 322, 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:

- \( x_{1i1}, x_{1i2}, \ldots, x_{1iT} \)

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-MacCurdy estimator can be applied only to balanced data.

The Amemiya-MacCurdy 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 \( x_1 \) variables. Although the Hausman-Taylor model requires only that the cross-sectional means of \( x_1 \) be orthogonal to \( \nu_i \), the Amemiya-MacCurdy 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 \( \alpha' = (\beta_1', \beta_2', \gamma_1', \gamma_2') \), its Hausman-Taylor estimate as \( \hat{\alpha}_{HT} \), and its Amemiya-MacCurdy estimate as \( \hat{\alpha}_{AM} \). Under the null hypothesis, both estimators are consistent and \( \hat{\alpha}_{AM} \) is efficient. The Hausman test statistic is

\[
    m = (\hat{\alpha}_{HT} - \hat{\alpha}_{AM})' \left( \hat{\Sigma}_{HT} - \hat{\Sigma}_{AM} \right)^{-1} (\hat{\alpha}_{HT} - \hat{\alpha}_{AM})
\]
Instrumental Variables Regression

Recall the original specification of the regression model

$$y_{it} = \alpha + \sum_{k=1}^{K} x_{itk}\beta_k + u_{it} \quad i = 1, \ldots, N; \; t = 1, \ldots, T_i$$

Consider that a subset of the regressors are endogenous—that is, these regressors are correlated with the observation-level errors $u_{it}$.

Formally, divide $x$ and $\beta$ so that

$$y_{it} = x_{1it}\beta_1 + x_{2it}\beta_2 + u_{it}$$

where $x_{1it}$ are $k_1$ uncorrelated (exogenous) regressors, $x_{2it}$ are $k_2$ endogenous regressors, and the intercept (if your model has one) is contained in $x_{1it}$.

In matrix notation, the model is

$$y = X_1\beta_1 + X_2\beta_2 + \mathbf{u}$$

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 $Z$ represent a set of $p$ instruments such that $p \geq k_2$. The set of endogenous regressors, $X_2$, are specified as effects in the ENDOGENOUS statement. The set of external instruments, $Z$, are specified as effects in the INSTRUMENTS statement.

The CPANEL procedure performs instrumental variables regression by the method of two-stage least squares (2SLS). Depending on the error structure of $u_{it}$, there are four 2SLS methods available, as described in the following subsections.

### Instrumental Variables Pooled Regression (IVPOOLED Option)

You perform IV pooled regression by specifying the IVPOOLED 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 iid with zero mean and variance $\sigma^2_e$. The errors $e_{it}$ are uncorrelated with $x_{1it}$, but correlated with $x_{2it}$.

Estimation is performed by two-stage least squares (2SLS), where you form the full instrument set by combining the external instruments $Z$ and the exogenous regression variables $X_1$. At the first stage, the predictions $\hat{x}_{2it}$ are obtained from regressing $x_{2it}$ on $x_{1it}$ and $z_{it}$. At the second stage, $\hat{\beta} = (\hat{\beta}_1, \hat{\beta}_2)'$ is estimated by regressing $y$ on $x_{1it}$ and $\hat{x}_{2it}$.

Form $X_*$ by stacking the data observations of $(x_{1it}, \hat{x}_{2it})$. The variance of $\hat{\beta}$ is

$$\text{Var} \left( \hat{\beta} \right) = \sigma^2_e \left( X_*' X_* \right)^{-1}$$
The error variance $\sigma^2_e$ is estimated by using residuals that are based on the original data:

$$\hat{\sigma}^2_e = \left( M - K - 1 \right)^{-1} \sum_{i=1}^N \sum_{t=1}^{T_i} \left( y_{it} - x_{1it} \hat{\beta}_1 - x_{2it} \hat{\beta}_2 \right)^2$$

**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 regression is two-stage least squares (2SLS) 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 $w_i = \sqrt{T_i/T}$, where $T = M/N$ is the average cross-sectional 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 $\sigma^2_e$. 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 $\beta$ and not $v_i$.

Let $Q_0 = \text{diag}(E_T)$. The matrix $Q_0$ is the within transformation that removes means (and $v_i$) from the estimation. Define $y_w = Q_0 y$, and use the same transformation to form $X_{1w}, X_{2w}$ and $Z_w$. Because the within transformation also removes the intercept term, the intercept is not included as part of $X_1$.

The estimates $\hat{\beta}_{W2SLS}$ and $\text{Var}(\hat{\beta}_{W2SLS})$ are obtained by performing 2SLS regression of $y_w$ on $X_{1w}$ and $X_{2w}$, using $Z_w$ as external instruments. When forming the error variance $\hat{\sigma}^2_e$, the error degrees of freedom are adjusted to equal $M - N - K$ to account for the $N$ fixed effects.

The estimate of the intercept is

$$\hat{\alpha} = \bar{y}_- - \bar{x}_- \hat{\beta}_{W2SLS}$$

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 is estimated as

$$\hat{v}_i = \bar{y}_i - \bar{x}_i \hat{\beta}_{W2SLS}$$

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 $\hat{\alpha}$. 


A poolability test is an $F$ test of 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

$$\text{SSE}_u = \left( y_w - X_{1w} \hat{\beta}_1, W2SLS - X_{2w} \hat{\beta}_2, W2SLS \right)' \left( y_w - X_{1w} \hat{\beta}_1, W2SLS - X_{2w} \hat{\beta}_2, W2SLS \right)$$

SSE$_{2u}$ is the second-stage unrestricted error sum of squares

$$\text{SSE}_{2u} = \left( y_w - X_{1w} \hat{\beta}_1, W2SLS - \hat{X}_{2w} \hat{\beta}_2, W2SLS \right)' \left( y_w - X_{1w} \hat{\beta}_1, W2SLS - \hat{X}_{2w} \hat{\beta}_2, W2SLS \right)$$

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 $y$ on $X_1$ and $X_2^* = \hat{X}_{2w} + P_0 X_2$.

**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 $\sigma_v^2$, and the $e_{it}$ are iid with zero mean and variance $\sigma_e^2$. The regressors in $x_{1it}$ are orthogonal to $e_{it}$, but the regressors in $x_{2it}$ are correlated with $e_{it}$. This form of endogeneity is different from that assumed by Hausman and Taylor (1981), where $x_{2it}$ 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

$$E \left( \hat{u}_F Q_0 \hat{u}_F \right) = (M - N - K)\sigma_e^2$$

$$E \left( \hat{u}_B P_0 \hat{u}_B \right) = \left[ M - \text{tr} \left( (X_{2w} P_0 X_0)^{-1} X_{2w} Z_0 Z_0' X_0 \right) \right] \sigma_v^2 + (N - K - 1)\sigma_e^2$$

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 316. Estimates of $\sigma_v^2$ and $\sigma_e^2$ 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 regression of $y_{it}^*$ on $x_{1it}^*$ and $x_{2it}^*$. For the first-stage regression, use the following instruments: $x_{1it}^*$, $\bar{x}_{1i}$, $z_{it}^*$, and $\bar{z}_i$.

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 336. Failure to reject the null hypothesis favors the random-effects specification.
Dynamic Panel Estimation (DYNDIFF and DYNSYS 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 DYNSYS 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 $\eta_{it}$. 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

$$ y_i^d = X_i^d \gamma + \eta_i^d $$

where

$$ y_i^d = \begin{pmatrix} \Delta y_{i3} \\ \Delta y_{i4} \\ \vdots \\ \Delta y_{iT} \end{pmatrix}, \quad X_i^d = \begin{pmatrix} \Delta y_{i2} & \Delta y_{i3} \\ \Delta y_{i3} & \Delta x_{i3} \\ \vdots & \vdots \\ \Delta y_{i,T-1} & \Delta x_{iT} \end{pmatrix}, \quad \gamma = \begin{pmatrix} \phi \\ \beta \end{pmatrix}, \quad \eta_i^d = \begin{pmatrix} \eta_{i3} \\ \eta_{i4} \\ \vdots \\ \eta_{iT} \end{pmatrix} $$
Proceeding with the idea that you can use \((y_{i1}, \ldots, y_{i, t-1})\) as instruments for \(\Delta y_{it}\), the instrument matrix for the lagged dependent variables is

\[
Z^d_i = \begin{pmatrix}
y_{i1} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & y_{i1} & y_{i2} & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 0 & y_{i1} & y_{i2} & y_{i3} & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & y_{i1} & \cdots & y_{i, T-2}
\end{pmatrix}
\]

This extends naturally to \(L > 1\) and \(K > 1\); simply add columns to \(X^d_i\) and elements to \(\gamma\) as appropriate. When an observation is either missing or lost because of missing lags, delete the corresponding rows of \(y^d_i\), \(X^d_i\), \(\eta^d_i\), and \(Z^d_i\). 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 \(Z_i = (Z^d_i, D_i)\), where

\[
D_i = \begin{pmatrix}
\Delta x_{i31} & \Delta x_{i32} & \cdots & \Delta x_{i3K} \\
\Delta x_{i41} & \Delta x_{i42} & \cdots & \Delta x_{i4K} \\
\vdots & \vdots & \vdots & \vdots \\
\Delta x_{iT1} & \Delta x_{iT2} & \cdots & \Delta x_{iTK}
\end{pmatrix}
\]

When \(L = 1\), the default \(Z_i\) has \((T-1)(T-2)/2 + K\) columns. Each column \(z_c\) of \(Z_i\) satisfies the moment condition \(E(z_c' \eta^d_i) = 0\).

**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 \(\phi\) is near 1 or \(\sigma_{\epsilon}^2 / \sigma_v^2\) is large, the lagged dependent variables \(y_{i, t-1}\) are weak instruments for the differenced variables \(\Delta y_{it}\). 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

\[
y^\ell_i = X^\ell_i \gamma + \epsilon^\ell_i
\]

where

\[
y^\ell_i = \begin{pmatrix}
y_{i2} \\
y_{i3} \\
\vdots \\
y_{iT}
\end{pmatrix}, \quad X^\ell_i = \begin{pmatrix}
y_{i1} & x_{i2} \\
y_{i2} & x_{i3} \\
\vdots & \vdots \\
y_{i, T-1} & x_{iT}
\end{pmatrix}, \quad \epsilon^\ell_i = \begin{pmatrix}
\nu_i + \epsilon_{i2} \\
\nu_i + \epsilon_{i3} \\
\vdots \\
\nu_i + \epsilon_{iT}
\end{pmatrix}
\]
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

$$
Z_i^l = \begin{pmatrix}
0 & 0 & 0 & \cdots & 0 \\
0 & \Delta y_{i2} & 0 & \cdots & 0 \\
0 & 0 & \Delta y_{i3} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \Delta y_{iT-1}
\end{pmatrix}
$$

where the first row corresponds to time $t = 2$. You can extend this to $L > 1$ and $K > 1$ by adding columns to $X_i^\ell$ and elements to $\eta$ as appropriate. Higher-order lags require deletion of the leading rows of $y_i^\ell$, $X_i^\ell$, $\epsilon_i^\ell$, and $Z_i^l$.

Regression on the full system is obtained by stacking $y_i^d$ and $y_i^\ell$ to form $y_i^s$, stacking $X_i^d$ and $X_i^\ell$ to form $X_i^s$, and stacking $\eta_i^d$ and $\epsilon_i^\ell$ to form $\epsilon_i^s$.

When you specify the DYNSYS model option, the default instrument matrix for the full system is

$$
Z_i = \begin{pmatrix}
Z_i^d & 0 & D_i \\
0 & Z_i^\ell & 0
\end{pmatrix}
$$

**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 $H_{1i} = \text{diag}(G_{1i}, G_{2i})$, where

$$
G_{1i} = \begin{pmatrix}
1 & -0.5 & 0 & \cdots & 0 & 0 & 0 \\
-0.5 & 1 & -0.5 & \cdots & 0 & 0 & 0 \\
0 & -0.5 & 1 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & -0.5 & 0 \\
0 & 0 & 0 & \cdots & -0.5 & 1 & -0.5 \\
0 & 0 & 0 & \cdots & 0 & -0.5 & 1
\end{pmatrix}
$$

and $G_{2i}$ is 0.5 times the identity matrix.

Define the weighting matrix as

$$
W_1 = \left( \sum_{i=1}^{N} Z_i^d H_{1i} Z_i \right)^{-1}
$$

and define the projections as

$$
P_y = \sum_{i=1}^{N} Z_i^\ell y_i^s; \quad P_x = \sum_{i=1}^{N} Z_i^\ell X_i^s
$$
The one-step GMM estimate of $\gamma$ is the weighted OLS estimator

$$\hat{\gamma}_1 = \left( P'_x W_1 P_x \right)^{-1} P'_x W_1 y$$

The variance of $\hat{\gamma}_1$ is

$$\text{Var}(\hat{\gamma}_1) = \hat{\sigma}_e^2 \left( P'_x W_1 P_x \right)^{-1}$$

where $\hat{\sigma}_e^2$ is the mean square error (MSE) that is derived solely from the difference equations, namely

$$\hat{\sigma}_e^2 = (M - K)^{-1} \sum_{i=1}^N \left( y'_i - X'_i \hat{\gamma}_1 \right) \left( y'_i - X'_i \hat{\gamma}_1 \right)'$$

The total number of observations, $M$, is equal to the number of observations for which the difference equations hold.

A disadvantage of $\hat{\gamma}_1$ is its reliance on the theoretical basis of $H_{1i}$. The two-step GMM estimate of $\gamma$ replaces $H_{1i}$ with a version that is obtained from the observed one-step residuals. Let $H_{2i}$ be the outer product of $\hat{e}_i = y'_i - X'_i \hat{\gamma}_1$. Then

$$\hat{\gamma}_2 = \left( P'_x W_2 P_x \right)^{-1} P'_x W_2 y$$

where

$$W_2 = \left( \sum_{i=1}^N Z'_i H_{2i} Z_i \right)^{-1}$$

The variance of $\hat{\gamma}_2$ is

$$\text{Var}(\hat{\gamma}_2) = \left( P'_x W_2 P_x \right)^{-1}$$

Robust variances are calculated by the sandwich method. The robust variance of $\hat{\gamma}_1$ is

$$\text{Var}^r(\hat{\gamma}_1) = \left( P'_x W_1 P_x \right)^{-1} P'_x W_1 W_2^{-1} W_1 P_x \left( P'_x W_1 P_x \right)^{-1}$$

The robust variance of $\hat{\gamma}_2$ is

$$\text{Var}^r(\hat{\gamma}_2) = \left( P'_x W_2 P_x \right)^{-1} P'_x W_2 W_3^{-1} W_2 P_x \left( P'_x W_2 P_x \right)^{-1}$$

where

$$W_3 = \left( \sum_{i=1}^N Z'_i H_{3i} Z_i \right)^{-1}$$

and $H_{3i}$ is the outer product of $y'_i - X'_i \hat{\gamma}_2$.

Arellano and Bond (1991), among others, note that robust two-step variance estimators are biased. Windmeijer (2005) derived a bias-corrected variance of $\hat{\gamma}_2$; you can obtain this correction by specifying the BIASCORRECTED option in the MODEL statement.
Define the one-step and two-step residuals as 
\[ \hat{e}_{1i} = y_t - X_t \hat{\gamma}_1 \] and 
\[ \hat{e}_{2i} = y_t - X_t \hat{\gamma}_2, \] respectively. Also define the projected two-step residual as
\[ P_e = \sum_{i=1}^{N} Z_i' \hat{e}_{2i} \]
Formulate the matrix \( D \) such that its \( k \)th column is 
\[ D_k = V_2 P_e W_2 F_k W_2 P_e, \] where \( V_2 = \text{Var}(\hat{\gamma}_2). \) The matrix \( F_k \) is the quadratic form
\[ F_k = \sum_{i=1}^{N} Z_i' \left( x_{ik} \hat{e}_{1i} + \hat{e}_{1i} x_{ik}' \right) Z_i \]
where \( x_{ik} \) is the \( k \)th column of \( X_t. \)
The Windmeijer (2005) bias-corrected variance is
\[ \text{Var}^w(\hat{\gamma}_2) = V_2 + DV_2 + V_2 D' + DV_1 D' \]
where \( V_1 \) is the robust variance estimate of \( \hat{\gamma}_1. \)

**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
\[ Z_i = \begin{pmatrix} Z_i^d & D_i & 0 \\ 0 & 0 & j_i \end{pmatrix} \]
where \( j_i \) 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 DYNSYS option in the MODEL statement, then there is no issue regarding the intercept. Under the default instrument specification, if \( X_t^\ell \) includes an intercept, then the level instruments include an added column of ones. That is,
\[ Z_i = \begin{pmatrix} Z_i^d & 0 & D_i & 0 \\ 0 & Z_i^\ell & 0 & j_i \end{pmatrix} \]

**Customizing Instruments**

When you specify the DYNSYS option to perform system GMM, the default instrument matrix is
\[ Z_i = \begin{pmatrix} Z_i^d & 0 & D_i & 0 \\ 0 & Z_i^\ell & 0 & c_i \end{pmatrix} \]
where \( c_i \) is a column of ones or, if you specify the NOINT option, \( c_i \) is 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 DYNSYS option in the MODEL statement. When you specify the DYNDIFF option instead, any specification (except the constant \( c_i \) 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 $Z_i^d$ 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:

$$G_i^d = \begin{pmatrix}
  x_{i1} & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
  0 & x_{i1} & x_{i2} & 0 & 0 & \cdots & 0 & 0 & 0 \\
  0 & 0 & 0 & x_{i1} & x_{i2} & x_{i3} & \cdots & 0 & 0 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
  0 & 0 & 0 & 0 & 0 & 0 & 0 & x_{i1} & \cdots & x_{i, T-2}
\end{pmatrix}$$

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 $Z_i^\ell$ 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:

$$G_i^\ell = \begin{pmatrix}
  0 & 0 & 0 & \cdots & 0 \\
  0 & \Delta x_{i2} & 0 & \cdots & 0 \\
  0 & 0 & \Delta x_{i3} & \cdots & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & 0 & \cdots & \Delta x_{i, T-1}
\end{pmatrix}$$

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.

```plaintext
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 $E(x_{it} \epsilon_{is}) = 0$ for $s < t$ and 0 otherwise.

The PREDETERMINED statement specifies a list of predetermined variables that are used to form predetermined 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:

$$P_i^d = \begin{pmatrix} x_{i1} & x_{i2} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & x_{i1} & x_{i2} & x_{i3} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & x_{i1} & \cdots & x_{i, T-1} \end{pmatrix}$$

The first row corresponds to time $t = 3$.

For the level equations, the PREDETERMINED statement produces instruments that are similar to those in $Z_i^l$, 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:

$$P_i^l = \begin{pmatrix} \Delta x_{i2} & 0 & 0 & \cdots & 0 \\ 0 & \Delta x_{i3} & 0 & \cdots & 0 \\ 0 & 0 & \Delta x_{i4} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \Delta x_{i,T} \end{pmatrix}$$

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 $D_i$, as described in the section “First Differencing” on page 327. 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:

\[
L_{ij} = \begin{pmatrix}
X_{i1} & X_{i2} \\
X_{i3} & X_{i4} \\
\vdots & \vdots \\
X_{iT1} & X_{iT2}
\end{pmatrix}
\]

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:

```sas
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

\[
J = \frac{1}{\sigma^2} \left( \sum_{i=1}^{N} Z_{i1}' \hat{\epsilon}_{1i} \right)' W_1 \left( \sum_{i=1}^{N} Z_{i1}' \hat{\epsilon}_{1i} \right)
\]

The Sargan test statistic for two-step GMM is

\[
J = \left( \sum_{i=1}^{N} Z_{i2}' \hat{\epsilon}_{2i} \right)' W_2 \left( \sum_{i=1}^{N} Z_{i2}' \hat{\epsilon}_{2i} \right)
\]
The null hypothesis of the Sargan test is that the moment conditions (as defined by the columns \( Z_i \)) hold, and thus \( Z_i \) form an adequate set of instruments. Under the null, \( J \) is distributed as \( \chi^2 \) with degrees of freedom equal to the rank of \( W_c \) \((c = 1, 2)\) minus the number of parameters \( K \). The nominal rank of \( W_c \) 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 \( W_2 \) 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 \( R^s_i \) be the working variance of the residuals from the full system. The precise definition of \( R^s_i \) depends on the GMM stage and whether robust variances are specified. Table 9.4 provides detailed definitions.

<table>
<thead>
<tr>
<th>Estimator</th>
<th>( R^s_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-step</td>
<td>( \hat{\sigma}^2_i H_{1i} )</td>
</tr>
<tr>
<td>One-step, robust</td>
<td>( H_{2i} )</td>
</tr>
<tr>
<td>Two-step</td>
<td>( H_{2i} )</td>
</tr>
<tr>
<td>Two-step, robust</td>
<td>( H_{3i} )</td>
</tr>
</tbody>
</table>

Define the residual vector

\[
\hat{e}_i = \begin{pmatrix} \hat{\eta}_i^d \\ 0 \end{pmatrix}
\]

where \( \hat{\eta}_i^d = y_i^d - X_i^d \hat{y}_c \) are the residuals from the difference equations, evaluated at the final estimate of \( \hat{y}_c \). The trailing zeros correspond to the level equations. Define \( \hat{\omega}_{mi} \) as a lagged version of \( \hat{e}_i \) such that the following statements are true:

1. The first \( m \) elements of \( \hat{\omega}_{mi} \) are 0.
2. The next \( p - m \) elements of \( \hat{\omega}_{mi} \) are the first \( p - m \) elements of \( \hat{e}_i \), where \( p \) is the number of difference equations.
3. The trailing elements of \( \hat{\omega}_{mi} \) that correspond to the level equations are 0.

Define the following:

\[
P_m = \sum_{i=1}^{N} Z_i^s R^s_i \hat{\omega}_{mi}
\]

\[
Q_m = \sum_{i=1}^{N} \hat{\omega}_{mi}^s X_i^s
\]
The AR\((m)\) test statistic is \(Z_m = k_0m \{k_1m + k_2m + k_3m\}^{-1/2}\), where

\[
\begin{align*}
k_0m &= \sum_{i=1}^{N} \hat{\omega}_{mi} \hat{e}_i \\
k_1m &= \sum_{i=1}^{N} \hat{\omega}_{mi} R_i \hat{\omega}_{mi} \\
k_2m &= -2Q_m \left( P'_x W_c P_x \right)^{-1} P'_x W_c P_m \\
k_3m &= Q_m VQ_m'
\end{align*}
\]

The matrix \(V\) 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 \(\text{SSE}_r\) is the error sum of squares from the restricted model (pooled regression) and \(\text{SSE}_u\) 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 \(\chi^2\) 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 $\hat{\beta}_e - \beta_e$, 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 $R\beta = q$, where $R$ is $J \times K$ and $q$ 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

$$L = (y - X\beta)'(y - X\beta) + 2\lambda(R\beta - q)$$

The Lagrangian is minimized by the restricted estimator $\beta^*$, and it can be shown that

$$\beta^* = \hat{\beta} - (X'X)^{-1}R'\lambda$$

where $\hat{\beta}$ is the unrestricted estimator.

Because $R\beta^* = q$, you can solve for $\lambda$ to obtain the Lagrange multipliers

$$\lambda^* = [R(X'X)^{-1}R']^{-1}(R\hat{\beta} - q)$$

The standard errors of the Lagrange multipliers are the square roots of the diagonal elements of the variance matrix

$$\text{Var}(\lambda^*) = \hat{\sigma}^2 [R(X'X)^{-1}R']^{-1}$$

where $\hat{\sigma}^2$ 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 $R\beta = q$, where $R$ is $J \times K$ and $q$ is $J \times 1$. The Wald test statistic is

$$\chi^2_W = (R\hat{\beta} - q)'(R\hat{V}R')^{-1}(R\hat{\beta} - q)$$

where $\hat{V}$ 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 $\text{SSE}_r$ is the restricted error sum of squares, $\text{SSE}_u$ 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

$$\chi^2_{LR} = M \ln \left[ 1 + \frac{JF}{M-K} \right]$$

The LM test statistic is

$$\chi^2_{LM} = M \left[ \frac{JF}{M-K} + JF \right]$$

The distribution of these test statistics is $\chi^2$ 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).

**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),

$$R^2 = 1 - \frac{\text{SSE}}{\text{y}^\prime D \hat{\Omega}^{-1} D \text{y}}$$

where $\text{SSE}$ is the error sum of squares from the final model fit, $\hat{\Omega}^{-1/2}$ represents the GLS transform, and $D = I_M - a^{-1} J_M \hat{\Omega}^{-1}$, for $a = j_M^j \hat{\Omega}^{-1} j_M$. In GLS models that do not have an intercept, the alternative R-square measure, which is attributed to Theil (1961), is calculated as follows:

$$R^2 = 1 - \frac{\text{SSE}}{\text{y}^\prime \hat{\Omega}^{-1} \text{y}}$$

In fixed-effects models, the R-square measure is

$$R^2 = 1 - \frac{\text{SSE}}{y_w^\prime y_w}$$

where $y_w$ 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 296.

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 9.5.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTest</td>
<td>Autoregression tests for the residuals</td>
<td>MODEL</td>
<td>DYNDIFF or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DYNNSYS</td>
</tr>
<tr>
<td>ClassInfo</td>
<td>Level information from the CLASS statement</td>
<td>CLASS</td>
<td>Default</td>
</tr>
<tr>
<td>CorrB</td>
<td>Correlation matrix of parameter estimates</td>
<td>MODEL</td>
<td>CORRB</td>
</tr>
<tr>
<td>CovB</td>
<td>Covariance matrix of parameter estimates</td>
<td>MODEL</td>
<td>COVB</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Fit statistics</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>FixedEffectsTest</td>
<td>$F$ test for poolability</td>
<td>MODEL</td>
<td>FIXONE, FIXTWO, or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IVFIXONE RANONE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RANTWO, or IVRANONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HTAYLOR</td>
</tr>
<tr>
<td>HausmanTestForRandom</td>
<td>Hausman test for random effects</td>
<td>MODEL</td>
<td></td>
</tr>
<tr>
<td>HausmanTestVsFixed</td>
<td>Hausman test for Hausman-Taylor models</td>
<td>MODEL</td>
<td></td>
</tr>
</tbody>
</table>
Table 9.5  continued

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>HausmanTestVsHausmanTaylor</td>
<td>Hausman test for Amemiya-MaCurdy models</td>
<td>MODEL</td>
<td>AMACURDY</td>
</tr>
<tr>
<td>Instruments</td>
<td>Instruments summary</td>
<td>MODEL</td>
<td>DYNDIFF or DYNSYS</td>
</tr>
<tr>
<td>ModelDescription</td>
<td>Model summary and data counts</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>ParameterComparisonTable</td>
<td>Comparison of model parameter estimates, standard errors, and $t$ tests</td>
<td>COMPARE</td>
<td></td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Solutions for the parameter estimates associated with effects in MODEL</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>ParameterEstimatesLegend</td>
<td>Legend that defines the abbreviations that were used in the parameter</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>SarganTest</td>
<td>Sargan test for overidentifying restrictions</td>
<td>MODEL</td>
<td>DYNDIFF or DYNSYS</td>
</tr>
<tr>
<td>StatComparisonTable</td>
<td>Comparison of model summary statistics</td>
<td>COMPARE</td>
<td></td>
</tr>
<tr>
<td>StructuralSummary</td>
<td>Endogenous and instrumental variable lists</td>
<td>MODEL</td>
<td>IVPOOLED, IVBTWNG, IVFIXONE, or IVRANONE</td>
</tr>
<tr>
<td>TestResults</td>
<td>Hypothesis tests of linear restrictions</td>
<td>TEST</td>
<td></td>
</tr>
<tr>
<td>VarianceComponents</td>
<td>Variance component estimates</td>
<td>MODEL</td>
<td>RANONE, RANTWO, HTAYLOR, AMACURDY, or IVRANONE</td>
</tr>
</tbody>
</table>
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,

$$\log (C_{it}) = \alpha + \beta_1 \log (Q_{it}) + \beta_2 \log (PF_{it}) + \beta_3 LF_{it} + \nu_i + \epsilon_{it}$$

where the $\nu_i$ are airline effects. The actual model in the original, untransformed variables is highly nonlinear:

$$C_{it} = \exp(\alpha + \beta_3 LF_{it} + \nu_i + \epsilon_{it}) Q_{it}^{\beta_1} PF_{it}^{\beta_2}$$

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.

```plaintext
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 ...
```

Examples: CPANEL Procedure
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 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

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
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

<table>
<thead>
<tr>
<th>Variable</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>t</th>
<th></th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>9.71356</td>
<td>0.2296</td>
<td>42.30</td>
<td>&lt;.0001</td>
<td>Intercept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
<td>0.919293</td>
<td>0.0299</td>
<td>30.76</td>
<td>&lt;.0001</td>
<td>Log Transformation of Quantity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPF</td>
<td>1</td>
<td>0.417492</td>
<td>0.0152</td>
<td>27.47</td>
<td>&lt;.0001</td>
<td>Log Transformation of Price of Fuel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LF</td>
<td>1</td>
<td>-1.07044</td>
<td>0.2017</td>
<td>-5.31</td>
<td>&lt;.0001</td>
<td>Load Factor (utilization index)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 1</td>
<td>1</td>
<td>-0.00759</td>
<td>0.0465</td>
<td>-0.16</td>
<td>0.8707</td>
<td>Cross-Sectional Effect: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 2</td>
<td>1</td>
<td>-0.04883</td>
<td>0.0390</td>
<td>-1.25</td>
<td>0.2144</td>
<td>Cross-Sectional Effect: 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 3</td>
<td>1</td>
<td>-0.21651</td>
<td>0.0184</td>
<td>-11.77</td>
<td>&lt;.0001</td>
<td>Cross-Sectional Effect: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 4</td>
<td>1</td>
<td>0.176971</td>
<td>0.0214</td>
<td>8.27</td>
<td>&lt;.0001</td>
<td>Cross-Sectional Effect: 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 5</td>
<td>1</td>
<td>0.016475</td>
<td>0.0378</td>
<td>0.44</td>
<td>0.6638</td>
<td>Cross-Sectional Effect: 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS 6</td>
<td>1</td>
<td>0.079484</td>
<td>0.0415</td>
<td>1.92</td>
<td>0.0588</td>
<td>Cross-Sectional Effect: 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

```plaintext
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)

<table>
<thead>
<tr>
<th>F Test for No Fixed Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num DF</td>
</tr>
<tr>
<td>19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>Intercept</td>
</tr>
<tr>
<td>IQ</td>
</tr>
<tr>
<td>IPF</td>
</tr>
<tr>
<td>LF</td>
</tr>
<tr>
<td>CS 1</td>
</tr>
<tr>
<td>CS 2</td>
</tr>
<tr>
<td>CS 3</td>
</tr>
<tr>
<td>CS 4</td>
</tr>
<tr>
<td>CS 5</td>
</tr>
<tr>
<td>CS 6</td>
</tr>
<tr>
<td>Time 1970</td>
</tr>
<tr>
<td>Time 1971</td>
</tr>
<tr>
<td>Time 1972</td>
</tr>
<tr>
<td>Time 1973</td>
</tr>
<tr>
<td>Time 1974</td>
</tr>
<tr>
<td>Time 1975</td>
</tr>
<tr>
<td>Time 1976</td>
</tr>
<tr>
<td>Time 1977</td>
</tr>
<tr>
<td>Time 1978</td>
</tr>
<tr>
<td>Time 1979</td>
</tr>
<tr>
<td>Time 1980</td>
</tr>
<tr>
<td>Time 1981</td>
</tr>
<tr>
<td>Time 1982</td>
</tr>
<tr>
<td>Time 1983</td>
</tr>
<tr>
<td>Time 1984</td>
</tr>
</tbody>
</table>

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.

```plaintext
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 -0.9916 0.4662 1.2110
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:

```plaintext
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)

<table>
<thead>
<tr>
<th>Model Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
</tr>
<tr>
<td>Time Series Length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE</td>
</tr>
<tr>
<td>MSE</td>
</tr>
<tr>
<td>R-Square</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Variance</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Sections</td>
<td>0.046394</td>
<td>0.215393</td>
</tr>
<tr>
<td>Error</td>
<td>0.00134</td>
<td>0.036608</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hausman Test For Random Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>Intercept</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>rd</td>
</tr>
<tr>
<td>rt</td>
</tr>
<tr>
<td>rs</td>
</tr>
</tbody>
</table>

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:

```sas
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

<table>
<thead>
<tr>
<th>Dependent Variable: d (Per Capita Demand Deposits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison of Model Statistics</td>
</tr>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>Var due to Cross Sections</td>
</tr>
<tr>
<td>Var due to Error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comparison of Model Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>Estimate</td>
</tr>
<tr>
<td>Std Err</td>
</tr>
</tbody>
</table>

You conclude that how you estimate variance components has little bearing on the regression results.


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.

```plaintext
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  9
  1  2  5.7203102112 43  1  0  1  4 16  0  0  0  0  0  9
  1  3  5.9964499474 40  1  0  1  5 25  0  0  0  0  0  9
  1  4  5.9964499474 39  1  0  1  6 36  0  0  0  0  0  9
  1  5  6.0614600182 42  1  0  1  7 49  0  1  0  0  0  9
  1  6  6.1737899780 35  1  0  1  8 64  0  1  0  0  0  9
  1  7  6.2441701889 32  1  0  1  9 81  0  1  0  0  0  9
  2  1  6.1633100510 34  0  0  1  30 900 1  0  0  0  0 11
  2  2  6.2146100998 27  0  0  1  31 961 1  0  0  0  0 11
  2  3  6.2634000778 33  0  0  1  32 1024 1  1  1  0  0 11
  2  4  6.5439100266 30  0  0  1  33 1089 1  1  0  0  0 11
  2  5  6.6970300674 30  0  0  1  34 1156 1  1  0  0  0 11
  2  6  6.7912201881 37  0  0  1  35 1225 1  1  0  0  0 11
  2  7  6.8156399727 30  0  0  1  36 1296 1  1  0  0  0 11
  ... more lines ...
```

You begin by fitting a one-way random-effects model:

```plaintext
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)

<table>
<thead>
<tr>
<th>Model Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
<td>RanOne</td>
</tr>
<tr>
<td>Data Set</td>
<td>PSID</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>4165</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
<td>595</td>
</tr>
<tr>
<td>Time Series Length</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variance Component Estimates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Variance</td>
</tr>
<tr>
<td>Cross Sections</td>
<td>0.068972</td>
</tr>
<tr>
<td>Error</td>
<td>0.023102</td>
</tr>
</tbody>
</table>

Hausman Test For Random Effects
Coefficients DF m Value Pr > m
9 9 5074.87 <.0001

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>DF</td>
</tr>
<tr>
<td>Intercept</td>
<td>1</td>
</tr>
<tr>
<td>wks</td>
<td>1</td>
</tr>
<tr>
<td>south</td>
<td>1</td>
</tr>
<tr>
<td>smsa</td>
<td>1</td>
</tr>
<tr>
<td>ms</td>
<td>1</td>
</tr>
<tr>
<td>exp</td>
<td>1</td>
</tr>
<tr>
<td>exp2</td>
<td>1</td>
</tr>
<tr>
<td>occ</td>
<td>1</td>
</tr>
<tr>
<td>ind</td>
<td>1</td>
</tr>
<tr>
<td>union</td>
<td>1</td>
</tr>
<tr>
<td>fem</td>
<td>1</td>
</tr>
<tr>
<td>blk</td>
<td>1</td>
</tr>
<tr>
<td>ed</td>
<td>1</td>
</tr>
</tbody>
</table>

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:

```plaintext
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)

<table>
<thead>
<tr>
<th>Variance Component Estimates</th>
<th>Variance</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Sections</td>
<td>0.886993</td>
<td>0.941803</td>
</tr>
<tr>
<td>Error</td>
<td>0.023044</td>
<td>0.151803</td>
</tr>
</tbody>
</table>

**Hausman Test vs. Fixed Effects**

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>DF</th>
<th>m Value</th>
<th>Pr &gt; m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>5.26</td>
<td>0.1539</td>
</tr>
</tbody>
</table>

**Parameter Estimates**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>TI</td>
<td>1</td>
<td>2.912726</td>
<td>0.2837</td>
<td>10.27</td>
<td>&lt;0.001</td>
<td>Intercept</td>
</tr>
<tr>
<td>wks</td>
<td>C</td>
<td>1</td>
<td>0.000837</td>
<td>0.000600</td>
<td>1.40</td>
<td>0.1627</td>
<td>Weeks worked</td>
</tr>
<tr>
<td>south</td>
<td></td>
<td>1</td>
<td>0.00744</td>
<td>0.0320</td>
<td>0.23</td>
<td>0.8159</td>
<td>1 if resides in the South</td>
</tr>
<tr>
<td>smsa</td>
<td>C</td>
<td>1</td>
<td>-0.04183</td>
<td>0.0190</td>
<td>-2.21</td>
<td>0.0274</td>
<td>1 if resides in SMSA</td>
</tr>
<tr>
<td>ms</td>
<td>C</td>
<td>1</td>
<td>-0.02985</td>
<td>0.0190</td>
<td>-1.57</td>
<td>0.1159</td>
<td>1 if married</td>
</tr>
<tr>
<td>exp</td>
<td>C</td>
<td>1</td>
<td>0.113133</td>
<td>0.00247</td>
<td>45.79</td>
<td>&lt;0.0001</td>
<td>Years full-time experience</td>
</tr>
<tr>
<td>exp2</td>
<td>C</td>
<td>1</td>
<td>-0.00042</td>
<td>0.000055</td>
<td>-7.67</td>
<td>&lt;0.0001</td>
<td>exp squared</td>
</tr>
<tr>
<td>occ</td>
<td></td>
<td>1</td>
<td>-0.0207</td>
<td>0.0138</td>
<td>-1.50</td>
<td>0.1331</td>
<td>1 if blue-collar occupation</td>
</tr>
<tr>
<td>ind</td>
<td></td>
<td>1</td>
<td>0.013604</td>
<td>0.0152</td>
<td>0.89</td>
<td>0.3720</td>
<td>1 if manufacturing</td>
</tr>
<tr>
<td>union</td>
<td>C</td>
<td>1</td>
<td>0.032771</td>
<td>0.0149</td>
<td>2.20</td>
<td>0.0280</td>
<td>1 if union contract</td>
</tr>
<tr>
<td>fem</td>
<td>TI</td>
<td>1</td>
<td>-0.13092</td>
<td>0.1267</td>
<td>-1.03</td>
<td>0.3014</td>
<td>1 if female</td>
</tr>
<tr>
<td>blk</td>
<td>TI</td>
<td>1</td>
<td>-0.28575</td>
<td>0.1557</td>
<td>-1.84</td>
<td>0.0665</td>
<td>1 if black</td>
</tr>
<tr>
<td>ed</td>
<td>C</td>
<td>1</td>
<td>0.137944</td>
<td>0.0212</td>
<td>6.49</td>
<td>&lt;0.0001</td>
<td>Years of education</td>
</tr>
</tbody>
</table>

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-MacCurdy model is a close relative of the Hausman-Taylor model. The only difference between the two is that the Amemiya-MacCurdy 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-MacCurdy model can take advantage of a more efficient set of instrumental variables.

The following statements fit the Amemiya-MacCurdy model:

```plaintext
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-MacCurdy 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-MacCurdy Estimation

The CPANEL Procedure
Amemiya and MacCurdy Model for Correlated Individual Effects
Dependent Variable: lwage (Log of wages)

<table>
<thead>
<tr>
<th>Variance Component Estimates</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Variance</td>
</tr>
<tr>
<td>Cross Sections</td>
<td>0.886993</td>
</tr>
<tr>
<td>Error</td>
<td>0.023044</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hausman Test vs. Hausman-Taylor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Finally, you should realize that the Hausman-Taylor and Amemiya-MacKurdy 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
        lcrmrte lprbarr lprbconv lprbpris lavgsen lpolpc
        ldensity ltaxpc lwcon 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, LCrnRte, 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 lcrmrte = 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: icrmte (log crimes committed per person)

<table>
<thead>
<tr>
<th>Model Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
<td>IVRanOne</td>
</tr>
<tr>
<td>Data Set</td>
<td>CRIME</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>630</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
<td>90</td>
</tr>
<tr>
<td>Time Series Length</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fit Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE</td>
<td>12.1250</td>
</tr>
<tr>
<td>DFE</td>
<td>603</td>
</tr>
<tr>
<td>MSE</td>
<td>0.0201</td>
</tr>
<tr>
<td>Root MSE</td>
<td>0.1418</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variance Component Estimates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Variance</td>
</tr>
<tr>
<td>Cross Sections</td>
<td>0.046036</td>
</tr>
<tr>
<td>Error</td>
<td>0.022272</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hausman Test For Random Effects</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>DF</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
</tbody>
</table>

**Structural Summary**

<table>
<thead>
<tr>
<th>Endogenous Variables</th>
<th>iprbarr ipolpc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruments</td>
<td>iprbconv iprbpris lavgisen idensity lwcon lwptc lwtrd lwfir lwser lwmg lwfed lwsta lwloc lpctymlc lpctmin west central urban d82 d83 d84 d85 d86 d87 llaxpc lmix</td>
</tr>
</tbody>
</table>
Example 9.4: Crime Rates in North Carolina: Instrumental Variables Regression

Output 9.4.1 continued

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td></td>
<td>1</td>
<td>-0.9538</td>
<td>1.2840</td>
<td>-0.74</td>
<td>0.4579</td>
<td>Intercept</td>
</tr>
<tr>
<td>lprbarr</td>
<td>EN</td>
<td>1</td>
<td>0.41293</td>
<td>0.9794</td>
<td>-4.24</td>
<td>&lt;.0001</td>
<td>log probability of arrest</td>
</tr>
<tr>
<td>lpolpc</td>
<td>EN</td>
<td>1</td>
<td>0.434749</td>
<td>0.0897</td>
<td>4.85</td>
<td>&lt;.0001</td>
<td>log police per capita</td>
</tr>
<tr>
<td>lprbconv</td>
<td></td>
<td>1</td>
<td>-0.32289</td>
<td>0.0536</td>
<td>-6.03</td>
<td>&lt;.0001</td>
<td>log probability of conviction</td>
</tr>
<tr>
<td>lprbpris</td>
<td></td>
<td>1</td>
<td>-0.18632</td>
<td>0.0419</td>
<td>-4.44</td>
<td>&lt;.0001</td>
<td>log probability of prison sentence</td>
</tr>
<tr>
<td>lavgsen</td>
<td></td>
<td>1</td>
<td>-0.01018</td>
<td>0.0270</td>
<td>-0.38</td>
<td>0.7066</td>
<td>log avg. sentence in days</td>
</tr>
<tr>
<td>ldensity</td>
<td></td>
<td>1</td>
<td>0.429028</td>
<td>0.0548</td>
<td>7.82</td>
<td>&lt;.0001</td>
<td>log people per sq. mile</td>
</tr>
<tr>
<td>lwcon</td>
<td></td>
<td>1</td>
<td>-0.00748</td>
<td>0.0396</td>
<td>-0.19</td>
<td>0.8503</td>
<td>log weekly wage, construction</td>
</tr>
<tr>
<td>lwteu</td>
<td></td>
<td>1</td>
<td>0.045445</td>
<td>0.0198</td>
<td>2.30</td>
<td>0.0220</td>
<td>log wkly wge, tms, util, commun</td>
</tr>
<tr>
<td>lwtrd</td>
<td></td>
<td>1</td>
<td>-0.00814</td>
<td>0.0414</td>
<td>-0.20</td>
<td>0.8441</td>
<td>log wkly wge, whlesle, retail trade</td>
</tr>
<tr>
<td>lwfr</td>
<td></td>
<td>1</td>
<td>-0.00364</td>
<td>0.0289</td>
<td>-0.13</td>
<td>0.8999</td>
<td>log wkly wge, fin, ins, real est</td>
</tr>
<tr>
<td>lwser</td>
<td></td>
<td>1</td>
<td>0.00561</td>
<td>0.0201</td>
<td>0.28</td>
<td>0.7805</td>
<td>log wkly wge, service industry</td>
</tr>
<tr>
<td>lwmgf</td>
<td></td>
<td>1</td>
<td>-0.20414</td>
<td>0.0804</td>
<td>-2.54</td>
<td>0.0114</td>
<td>log wkly wge, manufacturing</td>
</tr>
<tr>
<td>lwfed</td>
<td></td>
<td>1</td>
<td>-0.16351</td>
<td>0.1594</td>
<td>-1.03</td>
<td>0.3056</td>
<td>log wkly wge, fed employees</td>
</tr>
<tr>
<td>lwsta</td>
<td></td>
<td>1</td>
<td>-0.05405</td>
<td>0.1057</td>
<td>-0.51</td>
<td>0.6092</td>
<td>log wkly wge, state employees</td>
</tr>
<tr>
<td>lwloc</td>
<td></td>
<td>1</td>
<td>0.163052</td>
<td>0.1196</td>
<td>1.36</td>
<td>0.1734</td>
<td>log wkly wge, local gov emps</td>
</tr>
<tr>
<td>lpctymle</td>
<td></td>
<td>1</td>
<td>-0.10811</td>
<td>0.1397</td>
<td>-0.77</td>
<td>0.4393</td>
<td>log percent young male</td>
</tr>
<tr>
<td>lpctmin</td>
<td></td>
<td>1</td>
<td>0.189037</td>
<td>0.0415</td>
<td>4.56</td>
<td>&lt;.0001</td>
<td>log perc. minority, 1980</td>
</tr>
<tr>
<td>west</td>
<td></td>
<td>1</td>
<td>-0.22684</td>
<td>0.0996</td>
<td>-2.28</td>
<td>0.0231</td>
<td>1 if in western N.C.</td>
</tr>
<tr>
<td>central</td>
<td></td>
<td>1</td>
<td>-0.19404</td>
<td>0.0598</td>
<td>-3.24</td>
<td>0.0012</td>
<td>1 if in central N.C.</td>
</tr>
<tr>
<td>urban</td>
<td></td>
<td>1</td>
<td>-0.22515</td>
<td>0.1156</td>
<td>-1.95</td>
<td>0.0520</td>
<td>1 if in SMSA</td>
</tr>
<tr>
<td>d82</td>
<td></td>
<td>1</td>
<td>0.010745</td>
<td>0.0258</td>
<td>0.42</td>
<td>0.6772</td>
<td>1 if year =&lt; 82</td>
</tr>
<tr>
<td>d83</td>
<td></td>
<td>1</td>
<td>0.08379</td>
<td>0.0307</td>
<td>-2.73</td>
<td>0.0065</td>
<td>1 if year =&lt; 83</td>
</tr>
<tr>
<td>d84</td>
<td></td>
<td>1</td>
<td>-0.1035</td>
<td>0.0371</td>
<td>-2.79</td>
<td>0.0054</td>
<td>1 if year =&lt; 84</td>
</tr>
<tr>
<td>d85</td>
<td></td>
<td>1</td>
<td>0.0957</td>
<td>0.0495</td>
<td>1.94</td>
<td>0.0534</td>
<td>1 if year =&lt; 85</td>
</tr>
<tr>
<td>d86</td>
<td></td>
<td>1</td>
<td>0.0689</td>
<td>0.0596</td>
<td>-1.16</td>
<td>0.2481</td>
<td>1 if year =&lt; 86</td>
</tr>
<tr>
<td>d87</td>
<td></td>
<td>1</td>
<td>0.03141</td>
<td>0.0705</td>
<td>0.45</td>
<td>0.6562</td>
<td>1 if year =&lt; 87</td>
</tr>
</tbody>
</table>

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.
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 DYNYSYS Options)” on page 327. 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.

```plaintext
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:

```plaintext
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)

<table>
<thead>
<tr>
<th>Model Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
<td>DynDiff</td>
</tr>
<tr>
<td>Estimation Technique</td>
<td>One-Step GMM</td>
</tr>
<tr>
<td>Variance Estimation</td>
<td>Model Based</td>
</tr>
<tr>
<td>Data Set</td>
<td>CIGAR</td>
</tr>
<tr>
<td>Number of Observations</td>
<td>1380</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
<td>46</td>
</tr>
<tr>
<td>Time Series Length</td>
<td>30</td>
</tr>
<tr>
<td>GMM Bandwidth</td>
<td>30</td>
</tr>
</tbody>
</table>

**Fit Statistics**

<table>
<thead>
<tr>
<th>SSE</th>
<th>DFE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1373</td>
<td>1283</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSE</th>
<th>Root MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0024</td>
<td>0.0494</td>
</tr>
</tbody>
</table>

**Instruments**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variables</th>
<th>Number of Instruments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference Equations, Endogenous</td>
<td>LSales</td>
<td>406</td>
</tr>
<tr>
<td>Difference Equations, Standard</td>
<td>LPrice LDisp LMin</td>
<td>3</td>
</tr>
<tr>
<td>Level Equations, Standard</td>
<td>Intercept</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>410</td>
</tr>
</tbody>
</table>

**Sargan Test**

<table>
<thead>
<tr>
<th>DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>405</td>
<td>712.45</td>
<td>&lt;.0001</td>
</tr>
</tbody>
</table>

**Parameter Estimates**

<table>
<thead>
<tr>
<th>Variable</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>0.769092</td>
<td>0.0658</td>
<td>11.69</td>
<td>&lt;.0001</td>
<td>Intercept</td>
</tr>
<tr>
<td>LSales (Lag 1)</td>
<td>1</td>
<td>0.732212</td>
<td>0.0178</td>
<td>41.07</td>
<td>&lt;.0001</td>
<td>Lag 1: Log cigarette sales in packs per capita</td>
</tr>
<tr>
<td>LPrice</td>
<td>1</td>
<td>-0.26328</td>
<td>0.0255</td>
<td>-10.31</td>
<td>&lt;.0001</td>
<td>Log price per pack of cigarettes</td>
</tr>
<tr>
<td>LDisp</td>
<td>1</td>
<td>0.166116</td>
<td>0.0105</td>
<td>15.88</td>
<td>&lt;.0001</td>
<td>Log per capita disposable income</td>
</tr>
<tr>
<td>LMin</td>
<td>1</td>
<td>0.032726</td>
<td>0.0233</td>
<td>1.40</td>
<td>0.1604</td>
<td>Log minimum price in adjoining states per pack of cigarettes</td>
</tr>
</tbody>
</table>

**AR(m) Tests**

<table>
<thead>
<tr>
<th>Lag</th>
<th>Z</th>
<th>Pr &gt;</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-15.44</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.47</td>
<td>0.0134</td>
<td></td>
</tr>
</tbody>
</table>

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:

```plaintext
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 CPANDEL Procedure
Dynamic-Panel Estimation using Difference Equations (DynDiff)
Dependent Variable: LSales (Log cigarette sales in packs per capita)

<table>
<thead>
<tr>
<th>Model Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
</tr>
<tr>
<td>Estimation Technique</td>
</tr>
<tr>
<td>Variance Estimation</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
</tr>
<tr>
<td>Time Series Length</td>
</tr>
<tr>
<td>GMM Bandwidth</td>
</tr>
</tbody>
</table>

Fit Statistics

<table>
<thead>
<tr>
<th>Type</th>
<th>Variables</th>
<th>Number of Instruments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE</td>
<td>3.1277</td>
<td>DFE 1283</td>
</tr>
<tr>
<td>MSE</td>
<td>0.0024</td>
<td>Root MSE 0.0494</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Variables</th>
<th>Number of Instruments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference Equations, Endogenous</td>
<td>LSales</td>
<td>406</td>
</tr>
<tr>
<td>Difference Equations, Standard</td>
<td>LPrice LDisp LMin</td>
<td>3</td>
</tr>
<tr>
<td>Level Equations, Standard</td>
<td>Intercept</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>410</td>
</tr>
</tbody>
</table>

Sargan Test

<table>
<thead>
<tr>
<th>DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>45.20</td>
<td>0.3008</td>
</tr>
</tbody>
</table>

Parameter Estimates

| Variable          | DF   | Estimate | Standard Error | t Value | Pr > | | 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

<table>
<thead>
<tr>
<th>Lag</th>
<th>Z</th>
<th>Pr &gt;</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5.00</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.90</td>
<td>0.0575</td>
<td></td>
</tr>
</tbody>
</table>
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 \( LPrice \):

```plaintext
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

<table>
<thead>
<tr>
<th>Model Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation Method</td>
</tr>
<tr>
<td>Estimation Technique</td>
</tr>
<tr>
<td>Variance Estimation</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Cross Sections</td>
</tr>
<tr>
<td>Time Series Length</td>
</tr>
<tr>
<td>GMM Bandwidth</td>
</tr>
</tbody>
</table>

**Fit Statistics**

<table>
<thead>
<tr>
<th></th>
<th>SSE   3.3942</th>
<th>DFE  1283</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.0026</td>
<td>Root MSE 0.0514</td>
</tr>
</tbody>
</table>

**Instruments**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variables</th>
<th>Number of Instruments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference Equations, Endogenous</td>
<td>LSales LPrice</td>
<td>812</td>
</tr>
<tr>
<td>Difference Equations, Standard</td>
<td>LDisp LMin</td>
<td>2</td>
</tr>
<tr>
<td>Level Equations, Standard</td>
<td>Intercept</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>815</td>
</tr>
</tbody>
</table>

**Sargan Test**

<table>
<thead>
<tr>
<th>DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>44.98</td>
<td>0.3088</td>
</tr>
</tbody>
</table>
### Parameter Estimates

<table>
<thead>
<tr>
<th>Variable</th>
<th>DF Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1 0.531106</td>
<td>0.1271</td>
<td>4.18</td>
<td>&lt;.0001</td>
<td>Intercept</td>
</tr>
<tr>
<td>LSales (Lag 1)</td>
<td>1 0.799028</td>
<td>0.0412</td>
<td>19.38</td>
<td>&lt;.0001</td>
<td>Lag 1: Log cigarette sales in packs per capita</td>
</tr>
<tr>
<td>LPrice</td>
<td>1 -0.22491</td>
<td>0.0346</td>
<td>-6.50</td>
<td>&lt;.0001</td>
<td>Log price per pack of cigarettes</td>
</tr>
<tr>
<td>LDisp</td>
<td>1 0.139713</td>
<td>0.0202</td>
<td>6.92</td>
<td>&lt;.0001</td>
<td>Log per capita disposable income</td>
</tr>
<tr>
<td>LMin</td>
<td>1 0.029462</td>
<td>0.0383</td>
<td>0.77</td>
<td>0.4416</td>
<td>Log minimum price in adjoining states per pack of cigarettes</td>
</tr>
</tbody>
</table>

### AR(m) Tests

<table>
<thead>
<tr>
<th>Lag</th>
<th>Z</th>
<th>Pr &gt;</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5.02</td>
<td>&lt;.0001</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.94</td>
<td>0.0521</td>
<td></td>
</tr>
</tbody>
</table>

## References


Chapter 10
The CQLIM Procedure

Contents

Overview: CQLIM Procedure ................................................. 366
    PROC CQLIM Features .................................................. 366
    PROC CQLIM Compared with Other SAS Procedures .......... 367
    Using CAS Sessions and CAS Engine Librefs ............... 367
Getting Started: CQLIM Procedure ........................................ 368
Syntax: CQLIM Procedure ..................................................... 370
    Functional Summary .................................................. 370
    PROC CQLIM Statement ............................................... 372
    BOUNDS Statement ................................................... 373
    BY Statement ......................................................... 374
    CLASS Statement ..................................................... 374
    DISPLAY Statement .................................................. 374
    DISPLAYOUT Statement ............................................... 375
    ENDOGENOUS Statement ............................................. 376
    FREQ Statement ..................................................... 377
    HETERO Statement ................................................... 378
    INIT Statement ....................................................... 379
    MODEL Statement ..................................................... 379
    OUTPUT Statement ................................................... 381
    RESTRICT Statement ................................................ 382
    TEST Statement ....................................................... 383
    WEIGHT Statement .................................................... 384
Details: CQLIM Procedure .................................................... 384
    Ordinal Discrete Choice Modeling ................................ 384
    Limited Dependent Variable Models ......................... 385
    Stochastic Frontier Production and Cost Models .......... 386
    Heteroscedasticity .................................................... 388
    Tests on Parameters ................................................ 389
    Output to SAS Data Table ......................................... 389
    Naming ................................................................. 393
    ODS Table Names ..................................................... 394
Example: The CQLIM Procedure ............................................. 395
    Example 10.1: Model with Censoring ........................... 395
References ................................................................. 397
Overview: CQLIM Procedure

The CQLIM procedure is similar to the QLIM procedure in SAS/ETS software; PROC CQLIM requires SAS Cloud Analytic Services in order to run. Both procedures analyze univariate limited dependent variable models, in which dependent variables take discrete values or are observed within only a limited range of values. Unlike the QLIM procedure, which can be run only on a single workstation, the CQLIM procedure takes advantage of a distributed computing environment that 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 uses the maximum likelihood method to perform the inference of the available models. By default, it uses multiple threads to perform computations.

Another class of models that belongs to the limited dependent variable case is the stochastic frontier models. These models were first introduced by Aigner, Lovell, and Schmidt (1977); Meeusen and van den Broeck (1977); they are capable of analyzing technical inefficiencies within production or cost inefficiencies within costs.

The CQLIM procedure is also capable of modeling problems in which the dependent variable takes discrete values. For these cases, 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 considers 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.

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 (censored and truncated) with heteroscedasticity
- 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.

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.

The stochastic frontier production and cost models available in PROC CQLIM allow for random shocks of the production or cost. They include a systematic positive component in the error term that adjusts for technical or cost inefficiency.

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 support 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 present in the QLIM procedure. The QLIM procedure provides the following features, which are not present in the CQLIM procedure:

- multivariate modeling
- 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

---

**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,

\[
y^*_i = x_i' \beta + \epsilon_i
\]

\[
y_i = \begin{cases} 
y^*_i & \text{if } y^*_i > 0 \\
0 & \text{if } y^*_i \leq 0
\end{cases}
\]

where \( \epsilon_i \overset{iid}{\sim} N(0, \sigma^2) \) and \( x_i \) 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:

```plaintext
/--- 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**

<table>
<thead>
<tr>
<th>Model Fit Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependent Variable</td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Log Likelihood</td>
</tr>
<tr>
<td>Maximum Absolute Gradient</td>
</tr>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>AIC</td>
</tr>
<tr>
<td>SBC</td>
</tr>
<tr>
<td>Covariance Estimation</td>
</tr>
</tbody>
</table>

Convergence criterion (ABSGCONV=0.00001) satisfied.

| 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 \( \beta \). The last row is called _Sigma, which corresponds to the estimate of the error variance \( \sigma \).

### Syntax: CQLIM Procedure

The following statements are available in the CQLIM procedure:

```plaintext
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;
   ENDGENOUS 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 >;
```

One MODEL statement is required. If you specify a FREQ or WEIGHT statement more than once, only the variable that is specified in the first instance is used.

### Functional Summary

Table 10.1 summarizes the statements and options available in the CQLIM procedure.

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Table Options</strong></td>
<td>PROC CQLIM</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the input data table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writes predictions to an output data table</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td><strong>Declaring the Role of Variables</strong></td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies classification variables</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>Specifies a frequency variable</td>
<td>FREQ</td>
<td></td>
</tr>
<tr>
<td>Specifies a weight variable</td>
<td>WEIGHT</td>
<td>NONNORMALIZE</td>
</tr>
</tbody>
</table>
Table 10.1  continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Printing Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prints the correlation matrix of the estimates</td>
<td>PROC CQLIM</td>
<td>CORRB</td>
</tr>
<tr>
<td>Prints the covariance matrix of the estimates</td>
<td>PROC CQLIM</td>
<td>COVB</td>
</tr>
<tr>
<td>Prints a summary iteration listing</td>
<td>PROC CQLIM</td>
<td>ITPRINT</td>
</tr>
<tr>
<td><strong>Optimization Process Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sets boundary restrictions on parameters</td>
<td>BOUNDS</td>
<td></td>
</tr>
<tr>
<td>Sets initial values for parameters</td>
<td>INIT</td>
<td></td>
</tr>
<tr>
<td>Sets linear restrictions on parameters</td>
<td>RESTRICT</td>
<td></td>
</tr>
<tr>
<td><strong>Model Estimation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suppresses the intercept parameter</td>
<td>MODEL NOINT</td>
<td></td>
</tr>
<tr>
<td>Specifies the method to calculate parameter</td>
<td>PROC CQLIM COVEST=</td>
<td></td>
</tr>
<tr>
<td>covariance</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Endogenous Variable Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies a discrete variable</td>
<td>ENDOGENOUS DISCRETE()</td>
<td></td>
</tr>
<tr>
<td>Specifies a censored variable</td>
<td>ENDOGENOUS CENSORED()</td>
<td></td>
</tr>
<tr>
<td>Specifies a truncated variable</td>
<td>ENDOGENOUS TRUNCATED()</td>
<td></td>
</tr>
<tr>
<td>Specifies a stochastic frontier variable</td>
<td>ENDOGENOUS FRONTIER()</td>
<td></td>
</tr>
<tr>
<td><strong>Heteroscedasticity Model Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the function for heteroscedasticity models</td>
<td>HETERO LINK=</td>
<td></td>
</tr>
<tr>
<td>Squares the function for heteroscedasticity models</td>
<td>HETERO SQUARE</td>
<td></td>
</tr>
<tr>
<td>Specifies no constant for heteroscedasticity models</td>
<td>HETERO NOCONST</td>
<td></td>
</tr>
<tr>
<td><strong>Output Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td>Outputs the predicted values</td>
<td>OUTPUT PREDICTED</td>
<td></td>
</tr>
<tr>
<td>Outputs the structured part</td>
<td>OUTPUT XBETA</td>
<td></td>
</tr>
<tr>
<td>Outputs the residuals</td>
<td>OUTPUT RESIDUAL</td>
<td></td>
</tr>
<tr>
<td>Outputs the error standard deviation</td>
<td>OUTPUT ERRSTD</td>
<td></td>
</tr>
<tr>
<td>Outputs the marginal effects</td>
<td>OUTPUT MARGINAL</td>
<td></td>
</tr>
<tr>
<td>Outputs the probability for the current response</td>
<td>OUTPUT PROB</td>
<td></td>
</tr>
<tr>
<td>Outputs the probability for all responses</td>
<td>OUTPUT PROBALL</td>
<td></td>
</tr>
<tr>
<td>Outputs the expected value</td>
<td>OUTPUT EXPECTED</td>
<td></td>
</tr>
<tr>
<td>Outputs the conditional expected value</td>
<td>OUTPUT CONDITIONAL</td>
<td></td>
</tr>
<tr>
<td>Outputs the inverse Mills ratio</td>
<td>OUTPUT MILLS</td>
<td></td>
</tr>
<tr>
<td>Outputs the technical efficiency measures</td>
<td>OUTPUT TE1</td>
<td></td>
</tr>
<tr>
<td>Outputs the technical efficiency measures</td>
<td>OUTPUT TE2</td>
<td></td>
</tr>
</tbody>
</table>
Table 10.1  continued

<table>
<thead>
<tr>
<th>Description Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Request Options</td>
<td>ALL</td>
</tr>
<tr>
<td>Requests Wald, Lagrange multiplier, and likelihood ratio tests</td>
<td>TEST ALL</td>
</tr>
<tr>
<td>Requests the Wald test</td>
<td>TEST WALD</td>
</tr>
<tr>
<td>Requests the Lagrange multiplier test</td>
<td>TEST LM</td>
</tr>
<tr>
<td>Requests the likelihood ratio test</td>
<td>TEST LR</td>
</tr>
</tbody>
</table>

**PROC CQLIM Statement**

```
PROC CQLIM options;
```

The PROC CQLIM statement invokes the CQLIM procedure. You can specify the following options.

**Data Table Options**

```
DATA=\textit{CAS-libref.data-table}
```

names the input data table for PROC CQLIM to use. The default is the most recently created data table. \textit{CAS-libref.data-table} is a two-level name, where

- \textit{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 \textit{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 367.
- \textit{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.

**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 393. 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 382.

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;
```
Chapter 10: The CQLIM Procedure

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
</tbody>
</table>

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 394. 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 subsetted 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 394. 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. Currently, PROC CQLIM does not handle right-hand-side endogeneity. All variables that appear on the right-hand side of the equation are treated as exogenous. 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.

**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.
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:

\[
y_i = x_i' \beta + \epsilon_i \\
\epsilon_i \sim N(0, \sigma_i^2)
\]

For more information about the specification of functional forms, see the section “Heteroscedasticity” on page 388.

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:
    
    \[
    \sigma_i^2 = \sigma^2 (1 + \exp(z_i'y))
    \]

  - **LINEAR** specifies the linear link function:
    
    \[
    \sigma_i^2 = \sigma^2 (1 + z_i'y)
    \]

  By default, **LINK=EXP**.

- **NOCONST**

  specifies that no constant appear in the exponential heteroscedasticity model:

  \[
  \sigma_i^2 = \sigma^2 \exp(z_i'y)
  \]

- **SQUARE**

  estimates the model by using the square of the linear heteroscedasticity function. For example, you can specify the heteroscedasticity function

  \[
  \sigma_i^2 = \sigma^2 (1 + (z_i'y)^2)
  \]

  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 \( z_i'y \) is the same as the exponential of \( 2z_i'y \). Hence, the only difference is that all \( y \) estimates are divided by two.

  This option is ignored if you do not specify the **LINK=** option.
**INIT Statement**

```init
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
MODEL dependent-variables< (order-options ) > = regressors < / options > ;
```

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model. For binary LOGIT and binary PROBIT models you can specify the following `order-options`:

**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 SAS Visual Data Management and Utility Procedures Guide.

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.

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

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 367.

data-table  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 $\sigma_j$, 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.

XBETA
outputs estimates of $x\beta$.

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).

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:

```plaintext
proc cqlim data=mycas.dataset;
  model y = x1-x10 / censored(lb=0);
  restrict x1*2 <= x2 + x3;
run;
```

**TEST Statement**

```plaintext
'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):

```plaintext
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

\[ \beta_1 = 0 \]
Chapter 10: The CQLIM Procedure

and

\[ 0.5\beta_2 + 2\beta_3 = 0 \]

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

\[ y_i^* = x_i'\beta + \epsilon_i \]

where the value of the latent dependent variable, \( y_i^* \), is observed only as follows:

\[ y_i = 1 \quad \text{if} \quad y_i^* > 0 \]
\[ = 0 \quad \text{otherwise} \]
The disturbance, $\epsilon_i$, 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:

$$P(y_i = 1) = F(x'_i \beta) = \begin{cases} \Phi(x'_i \beta) & (\text{probit}) \\ \Lambda(x'_i \beta) & (\text{logit}) \end{cases}$$

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

$$y'_i = x'_i \beta + \epsilon_i$$

where error disturbances, $\epsilon_i$, have the distribution function $F$. The unobserved continuous random variable, $y'_i$, is identified as $M$ categories. Suppose there are $M + 1$ real numbers, $\mu_0, \ldots, \mu_M$, where $\mu_0 = -\infty$, $\mu_1 = 0$, $\mu_M = \infty$, and $\mu_0 \leq \mu_1 \leq \cdots \leq \mu_M$. Define

$$R_{i,j} = \mu_j - x'_i \beta$$

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
\[ y_i^* = x_i' \beta + \epsilon_i \]

\[ y_i = \begin{cases} 
  y_i^* & \text{if } y_i^* > 0 \\
  0 & \text{if } y_i^* \leq 0 
\end{cases} \]

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

\[
\ell = \sum_{i \in \{y_i > 0\}} \left\{ -\ln \phi(x_i' \beta / \sigma) + \ln \left[ \frac{\phi((y_i - x_i' \beta) / \sigma)}{\sigma} \right] \right\}
\]

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); Meeussen 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 \( \epsilon_i \) is negatively skewed and represents technical inefficiency. If \( u_i < 0 \), the error term \( \epsilon_i \) 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

\[
f(u, v) = \frac{1}{\sqrt{2\pi \sigma_u \sigma_v}} \exp \left\{ -\frac{u^2}{2\sigma_u^2} - \frac{v^2}{2\sigma_v^2} \right\}
\]

The marginal density function of \( \epsilon \) 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 $\epsilon$ 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 $\epsilon_i$ is heteroscedastic, the variance can be specified as a function of variables

$$E(\epsilon_i^2) = \sigma_i^2 = f(z_i' \gamma)$$

Table 10.3 shows various functional forms of heteroscedasticity and the corresponding options to request each model.

<table>
<thead>
<tr>
<th>Number</th>
<th>Model</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$f(z_i' \gamma) = \sigma^2 (1 + \exp(z_i' \gamma))$</td>
<td>LINK=EXP (default)</td>
</tr>
<tr>
<td>2</td>
<td>$f(z_i' \gamma) = \sigma^2 \exp(z_i' \gamma)$</td>
<td>LINK=EXP NOCONST</td>
</tr>
<tr>
<td>3</td>
<td>$f(z_i' \gamma) = \sigma^2 (1 + \sum_{l=1}^{L} \gamma_l z_{li})$</td>
<td>LINK=LINEAR</td>
</tr>
<tr>
<td>4</td>
<td>$f(z_i' \gamma) = \sigma^2 (1 + (\sum_{l=1}^{L} \gamma_l z_{li})^2)$</td>
<td>LINK=LINEAR SQUARE</td>
</tr>
</tbody>
</table>

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.
In general, the tested hypothesis can be written as

\[ H_0 : h(\theta) = 0 \]

where \( h(\theta) \) is an \( r \times 1 \) vector-valued function of the parameters \( \theta \) 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 \( \theta \), and let \( \tilde{\theta} \) be the constrained estimate of \( \theta \) such that \( h(\tilde{\theta}) = 0 \). Let

\[ A(\theta) = \frac{\partial h(\theta)}{\partial \theta} \bigg|_{\tilde{\theta}} \]

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
  \[
  LM = \lambda' A(\hat{\theta})\hat{V} A'(\hat{\theta}) \lambda
  \]
  where \( \lambda \) 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 \( \theta \) and \( L \) is the concentrated log-likelihood value.

The following statements use the TEST statement to perform a likelihood ratio test:

```sas
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 $\sigma_i$ 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 ($\mu_0, \ldots, \mu_M$), define

$$ R_{i,j} = \mu_j - x_i' \beta $$

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

$$ \frac{\partial \text{Prob}[y_i = j]}{\partial x} = [f(\mu_{j-1} - x_i' \beta) - f(\mu_j - x_i' \beta)] \beta $$

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

$$ \frac{\partial E[y_i | L_i < y_i^* < R_i]}{\partial x} = \beta \cdot \left[ 1 - \frac{(\phi(a_i) - \phi(b_i))^2}{(\Phi(b_i) - \Phi(a_i))^2} + \frac{a_i \phi(a_i) - b_i \phi(b_i)}{\Phi(b_i) - \Phi(a_i)} \right] $$

where $a_i = \frac{L_i - x_i' \beta}{\sigma_i}$ and $b_i = \frac{R_i - x_i' \beta}{\sigma_i}$.

The marginal effects in the censored regression model are

$$ \frac{\partial E[y_i | x_i]}{\partial x} = \beta \times \text{Prob}[L_i < y_i^* < R_i] $$

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 $a_i = \frac{L_i - x_i' \beta}{\sigma_i}$ and $b_i = \frac{R_i - x_i' \beta}{\sigma_i}$.

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

$$ \lambda = \frac{\phi(x_i' \beta)}{\Phi(x_i' \beta)} $$

for a binary discrete variable.
The expected value is the unconditional expectation of the dependent variable. For a censored variable, it is

\[ E[y_i] = \Phi(a_i) L_i + (x_i' \beta + \lambda \sigma_i) (\Phi(b_i) - \Phi(a_i)) + (1 - \Phi(b_i)) R_i \]

For a left-censored variable \( R_i = \infty \), this formula is

\[ E[y_i] = \Phi(a_i) L_i + (x_i' \beta + \lambda \sigma_i) (1 - \Phi(a_i)) \]

where \( \lambda = \frac{\Phi(a_i)}{1 - \Phi(a_i)} \).

For a right-censored variable \( L_i = -\infty \), this formula is

\[ E[y_i] = (x_i' \beta + \lambda \sigma_i) \Phi(b_i) + (1 - \Phi(b_i)) R_i \]

where \( \lambda = -\frac{\Phi(b_i)}{\Phi(b_i)} \).

For a noncensored variable, this formula is

\[ E[y_i] = x_i' \beta \]

The conditional expected value is the expectation given that the variable is inside the boundaries:

\[ E[y_i | L_i < y_i < R_i] = x_i' \beta + \lambda \sigma_i \]

**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 386.

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 386. 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 \( \mu_\star = -\epsilon \sigma_u^2 / \sigma^2 \) and \( \sigma_\star^2 = \sigma_u^2 \sigma_v^2 / \sigma^2 \). Then, as shown by Jondrow et al. (1982), conditional density is as follows:

\[
\begin{align*}
  f(u|\epsilon) &= \frac{f(u, \epsilon)}{f(\epsilon)} = \frac{1}{\sqrt{2\pi\sigma_\star}} \exp \left\{ -\frac{(u - \mu_{\star})^2}{2\sigma_\star^2} \right\} \sqrt{1 - \Phi \left( -\frac{\mu_{\star}}{\sigma_\star} \right)}
\end{align*}
\]

Hence, \( f(u|\epsilon) \) is the density for \( N^+(\mu_{\star}, \sigma_\star^2) \).

From this result, it follows that the estimate of technical efficiency (Battese and Coelli 1988) is

\[
TE1_i = E(\exp(-u_i)|\epsilon_i) = \left[ \frac{1 - \Phi(\sigma^2 - \mu_{\star i}/\sigma^2)}{1 - \Phi(-\mu_{\star i}/\sigma^2)} \right] \exp \left\{ -\mu_{\star i} + \frac{1}{2} \frac{\sigma^2}{\sigma^2} \right\}
\]

The second version of the estimate (Jondrow et al. 1982) is

\[
TE2_i = \exp\{-E(u_i|\epsilon_i)\}
\]

where

\[
E(u_i|\epsilon_i) = \mu_{\star i} + \sigma_\star \left[ \frac{\phi(-\mu_{\star i}/\sigma_\star)}{1 - \Phi(-\mu_{\star i}/\sigma_\star)} \right] = \sigma_\star \left[ \frac{\phi(\epsilon_i \lambda / \sigma)}{1 - \Phi(\epsilon_i \lambda / \sigma)} - \left( \frac{\epsilon_i \lambda}{\sigma} \right) \right]
\]

**Normal-Exponential Model**

Define \( A = -\bar{\mu}/\sigma_v \) and \( \bar{\mu} = -\epsilon - \sigma_v^2 / \sigma_u \). Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

\[
\begin{align*}
  f(u|\epsilon) &= \frac{1}{\sqrt{2\pi\sigma_v} \Phi(-\bar{\mu}/\sigma_v)} \exp \left\{ -\frac{(u - \bar{\mu})^2}{2\sigma_v^2} \right\}
\end{align*}
\]

Hence, \( f(u|\epsilon) \) is the density for \( N^+(\bar{\mu}, \sigma_v^2) \).

From this result, it follows that the estimate of technical efficiency is

\[
TE1_i = E(\exp(-u_i)|\epsilon_i) = \left[ \frac{1 - \Phi(\sigma_v - \bar{\mu}_i/\sigma_v)}{1 - \Phi(-\bar{\mu}_i/\sigma_v)} \right] \exp \left\{ -\bar{\mu}_i + \frac{1}{2} \frac{\sigma_v^2}{\sigma_v^2} \right\}
\]

The second version of the estimate is

\[
TE2_i = \exp\{-E(u_i|\epsilon_i)\}
\]

where

\[
E(u_i|\epsilon_i) = \bar{\mu}_i + \sigma_v \left[ \frac{\phi(-\bar{\mu}_i/\sigma_v)}{1 - \Phi(-\bar{\mu}_i/\sigma_v)} \right] = \sigma_v \left[ \frac{\phi(A)}{\Phi(-A)} - A \right]
\]
Normal–Truncated Normal Model

Define $\bar{\mu} = (-\sigma_u^2 \varepsilon_i + \mu \sigma_v^2) / \sigma^2$ and $\sigma_\epsilon^2 = \sigma_u^2 \sigma_v^2 / \sigma^2$. Then, as shown by Kumbhakar and Lovell (2000), conditional density is as follows:

$$f(u|\epsilon) = \frac{1}{\sqrt{2\pi} \sigma_\epsilon [1 - \Phi(-\bar{\mu}/\sigma_\epsilon)]} \exp \left\{ -\frac{(u - \bar{\mu})^2}{2\sigma_\epsilon^2} \right\}$$

Hence, $f(u|\epsilon)$ is the density for $N^+(\bar{\mu}, \sigma_\epsilon^2)$.

From this result, it follows that the estimate of technical efficiency is

$$TE1_i = E(\exp\{-u_i\}|\epsilon_i) = \frac{1 - \Phi(\sigma_\epsilon - \bar{\mu}_i/\sigma_\epsilon)}{1 - \Phi(-\bar{\mu}_i/\sigma_\epsilon)} \exp \left\{ -\bar{\mu}_i + \frac{1}{2} \sigma^2_\epsilon \right\}$$

The second version of the estimate is

$$TE2_i = \exp\{-E(u_i|\epsilon_i)\}$$

where

$$E(u_i|\epsilon_i) = \bar{\mu}_i + \sigma_\epsilon \left[ \frac{\phi(\bar{\mu}_i/\sigma_\epsilon)}{1 - \Phi(-\bar{\mu}_i/\sigma_\epsilon)} \right]$$

Naming

Naming of Parameters

In the CQLIM procedure, 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 called Intercept. The coefficients of independent variables are named by the independent variables. The standard deviation of the errors is called _Sigma. By default, _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.

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

<table>
<thead>
<tr>
<th>output-option</th>
<th>Variable Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDITIONAL</td>
<td>CEXPCT_y</td>
<td>Conditional expected value of y, conditioned on the truncation</td>
</tr>
<tr>
<td>ERRSTD</td>
<td>ERRSTD_y</td>
<td>Standard deviation of error term</td>
</tr>
<tr>
<td>EXPECTED</td>
<td>EXPCT_y</td>
<td>Unconditional expected value of y</td>
</tr>
<tr>
<td>MARGINAL</td>
<td>MEFF_x</td>
<td>Marginal effect of x on y ((\frac{dy}{dx})) with single equation</td>
</tr>
<tr>
<td>PREDICTED</td>
<td>P_y</td>
<td>Predicted value of y</td>
</tr>
<tr>
<td>RESIDUAL</td>
<td>RESID_y</td>
<td>Residual of y, (y – PredictedY)</td>
</tr>
<tr>
<td>PROB</td>
<td>PROB_y</td>
<td>Probability that y is taking the observed value in this observation (discrete y only)</td>
</tr>
<tr>
<td>PROBALL</td>
<td>PROBI_y</td>
<td>Probability that y is taking the ith value (discrete y only)</td>
</tr>
<tr>
<td>MILLS</td>
<td>MILLS_y</td>
<td>Inverse Mills ratio for y</td>
</tr>
<tr>
<td>TE1</td>
<td>TE1</td>
<td>Technical efficiency estimate for each producer proposed by Battese and Coelli (1988)</td>
</tr>
<tr>
<td>TE2</td>
<td>TE2</td>
<td>Technical efficiency estimate for each producer proposed by Jondrow et al. (1982)</td>
</tr>
<tr>
<td>XBETA</td>
<td>XBETA_y</td>
<td>Structure part ((x'\beta)) of y equation</td>
</tr>
</tbody>
</table>

### 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>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassInfo</td>
<td>Level information for classification variables</td>
<td>CLASS</td>
<td>Default</td>
</tr>
<tr>
<td>Correlation</td>
<td>Correlation of parameter estimates</td>
<td>PROC</td>
<td>CORRB</td>
</tr>
<tr>
<td>Covariance</td>
<td>Covariance of parameter estimates</td>
<td>PROC</td>
<td>COVB</td>
</tr>
<tr>
<td>DiscreteRespProfile</td>
<td>Response profile</td>
<td>ENDogenous</td>
<td>DISCRETE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or MODEL</td>
<td></td>
</tr>
<tr>
<td>FitModelSummary</td>
<td>Summary of nonlinear estimation</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>GoodnessOfFit</td>
<td>Pseudo-R-square measures</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>InitialParameterEstimates</td>
<td>Optimization start</td>
<td>PROC</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>IterationHistory</td>
<td>Iteration history of nonlinear estimation</td>
<td>PROC</td>
<td>ITPRINT</td>
</tr>
<tr>
<td>OptimizationResults</td>
<td>Optimization results</td>
<td>PROC</td>
<td>ITPRINT</td>
</tr>
</tbody>
</table>
### 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.

```plaintext
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;
```

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

#### Estimating a Tobit Model

The CQLIM Procedure

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Standard Error</th>
<th>Type</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
<th>N Obs Lower Bound</th>
<th>N Obs Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>41.11168</td>
<td>59.336269</td>
<td>Censored</td>
<td>0</td>
<td>400</td>
<td>246E4</td>
<td>189</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Fit Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependent Variable</td>
</tr>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Data Set</td>
</tr>
<tr>
<td>Log Likelihood</td>
</tr>
<tr>
<td>Maximum Absolute Gradient</td>
</tr>
<tr>
<td>Number of Iterations</td>
</tr>
<tr>
<td>Optimization Method</td>
</tr>
<tr>
<td>AIC</td>
</tr>
<tr>
<td>SBC</td>
</tr>
<tr>
<td>Covariance Estimation</td>
</tr>
</tbody>
</table>

Convergence criterion (FCONV=1E-11) satisfied.

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
<th>Parameter DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Approx Pr &gt;</th>
<th>t</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>2.045243</td>
<td>0.054689</td>
<td>37.40</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>1</td>
<td>3.021825</td>
<td>0.049348</td>
<td>61.23</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>1</td>
<td>4.006987</td>
<td>0.049345</td>
<td>81.20</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>1</td>
<td>1.955760</td>
<td>0.049327</td>
<td>39.65</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>1</td>
<td>4.045403</td>
<td>0.049338</td>
<td>81.19</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x5</td>
<td>1</td>
<td>-3.020339</td>
<td>0.049310</td>
<td>-61.25</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x6</td>
<td>1</td>
<td>-5.041650</td>
<td>0.049351</td>
<td>-102.16</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x7</td>
<td>1</td>
<td>-2.950510</td>
<td>0.049362</td>
<td>-59.77</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sigma</strong></td>
<td>1</td>
<td>100.015057</td>
<td>0.049335</td>
<td>2027.25</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
References


Chapter 11  
The CSPATIALREG Procedure

Contents
Overview: CSPATIALREG Procedure ................................................. 400
    PROC CSPATIALREG Features ........................................... 400
    PROC CSPATIALREG Compared with the SPATIALREG Procedure .... 401
    Using CAS Sessions and CAS Engine Librefs ............................ 402
Getting Started: CSPATIALREG Procedure ................................... 402
Syntax: CSPATIALREG Procedure .............................................. 406
    Functional Summary ....................................................... 407
    PROC CSPATIALREG Statement ......................................... 408
    BOUNDS Statement ....................................................... 412
    BY Statement .................................................................. 413
    CLASS Statement ......................................................... 413
    DISPLAY Statement ....................................................... 414
    DISPLAYOUT Statement ................................................... 415
    INIT Statement ............................................................. 415
    MODEL Statement ......................................................... 416
    OUTPUT Statement ........................................................ 417
    RESTRICT Statement ....................................................... 418
    SPATIALEFFECTS Statement ............................................. 418
    SPATIALID Statement ..................................................... 419
    TEST Statement ............................................................ 419
Details: CSPATIALREG Procedure .................................................. 420
    Missing Values .............................................................. 420
    Spatial Autoregressive Models .......................................... 421
    Spatial Durbin Models ................................................... 421
    Spatial Error Models ..................................................... 422
    Spatial Durbin Error Models ............................................ 423
    Spatial Moving Average Models ....................................... 424
    Spatial Durbin Moving Average Models ............................... 424
    Spatial Autoregressive Moving Average Models ..................... 425
    Spatial Durbin Autoregressive Moving Average Models .......... 426
    Spatial Autoregressive Confused Models ............................. 427
    Spatial Durbin Autoregressive Confused Models ................... 427
    Linear Regression Models ............................................... 428
    Spatial Lag of X Models ................................................. 429
    Specifying the Spatial Weights Matrix ................................. 430
    Compact Representation of the Spatial Weights Matrix ............ 431
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 63 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. The CSPATIALREG procedure provides the following additional features, which are not provided by the SPATIALREG procedure:

• support for Taylor and Chebyshev approximation techniques for spatial error models
• support for Taylor and Chebyshev approximation techniques for spatial Durbin error models
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:

```sas
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:

```sas
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:

```sas
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 430.
The following statements fit a SAR model:

```sas
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 table are shown in Figure 11.1.

<table>
<thead>
<tr>
<th>Obs</th>
<th>crime</th>
<th>income</th>
<th>hvalue</th>
<th>lat</th>
<th>lon</th>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.802</td>
<td>21.232</td>
<td>44.567</td>
<td>35.62</td>
<td>42.38</td>
<td>s1</td>
</tr>
<tr>
<td>2</td>
<td>0.178</td>
<td>8.438</td>
<td>75.000</td>
<td>33.36</td>
<td>38.41</td>
<td>s4</td>
</tr>
<tr>
<td>3</td>
<td>50.732</td>
<td>11.252</td>
<td>23.225</td>
<td>40.01</td>
<td>38.00</td>
<td>s7</td>
</tr>
<tr>
<td>4</td>
<td>34.001</td>
<td>13.598</td>
<td>96.400</td>
<td>47.61</td>
<td>36.42</td>
<td>s10</td>
</tr>
<tr>
<td>5</td>
<td>19.146</td>
<td>18.942</td>
<td>40.300</td>
<td>50.11</td>
<td>29.91</td>
<td>s13</td>
</tr>
<tr>
<td>6</td>
<td>16.241</td>
<td>29.833</td>
<td>61.950</td>
<td>48.44</td>
<td>27.93</td>
<td>s16</td>
</tr>
<tr>
<td>7</td>
<td>33.705</td>
<td>11.709</td>
<td>30.450</td>
<td>43.37</td>
<td>33.46</td>
<td>s19</td>
</tr>
<tr>
<td>8</td>
<td>41.968</td>
<td>9.918</td>
<td>23.600</td>
<td>44.10</td>
<td>30.40</td>
<td>s22</td>
</tr>
<tr>
<td>9</td>
<td>25.962</td>
<td>16.961</td>
<td>33.500</td>
<td>43.23</td>
<td>27.31</td>
<td>s25</td>
</tr>
<tr>
<td>10</td>
<td>29.028</td>
<td>14.135</td>
<td>27.733</td>
<td>39.32</td>
<td>25.85</td>
<td>s28</td>
</tr>
</tbody>
</table>

The following SAS statements fit a linear regression model to the CrimeOH data set:

```sas
/*--- 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.
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.

The following statements fit a SAR model to the CrimeOH data set:

```plaintext
/**** SAR Model ****/
  model crime = income hvalue / type=SAR;
  spatialid sid;
run;
```

The NONORMALIZE 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.
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 $\rho$ is estimated to be about 0.431, with a p-value of 0.0005.

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:

```sas
/*-- SDM --*/
  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.
The spatial autoregressive coefficient $\rho$ 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 $-2(L_{\text{linear}} - L_{\text{SAR}}) = -2(-187.38 + 182.39) = 9.98$, where $L_{\text{linear}}$ and $L_{\text{SAR}}$ 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.
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>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Table Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the CAS input data table</td>
<td>PROC CSPATIALREG</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the CAS spatial weights data input table</td>
<td>PROC CSPATIALREG</td>
<td>WMAT=</td>
</tr>
<tr>
<td>Suppresses normalization of the spatial weights</td>
<td>PROC CSPATIALREG</td>
<td>NONORMALIZE</td>
</tr>
<tr>
<td>Writes estimates of $x_i' \beta$, predicted values, and residuals to an output data table</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td><strong>Approximation Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the approximation-related options</td>
<td>PROC CSPATIALREG</td>
<td>APPROXIMATION=</td>
</tr>
<tr>
<td><strong>Declaring the Role of Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies classification variables</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>Specifies a spatial ID variable</td>
<td>SPATIALID</td>
<td></td>
</tr>
<tr>
<td><strong>Printing Control Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prints the correlation matrix of the estimates</td>
<td>MODEL</td>
<td>CORRB</td>
</tr>
<tr>
<td>Prints the covariance matrix of the estimates</td>
<td>MODEL</td>
<td>COVB</td>
</tr>
<tr>
<td>Prints a summary iteration listing</td>
<td>MODEL</td>
<td>ITPRINT</td>
</tr>
</tbody>
</table>
**Table 11.1  continued**

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppresses the normal printed output</td>
<td>PROC CSPATIALREG</td>
<td>NOPRINT</td>
</tr>
<tr>
<td>Prints all available output</td>
<td>MODEL</td>
<td>PRINTALL</td>
</tr>
<tr>
<td>Prints timing information</td>
<td>PROC CSPATIALREG</td>
<td>PRINTTIMING</td>
</tr>
<tr>
<td>Prints the names used internally for the parameters</td>
<td>PROC CSPATIALREG</td>
<td>PRINTINTERNALNAMES</td>
</tr>
</tbody>
</table>

**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 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=               |
| Specifies the type of covariance matrix                                  | PROC CSPATIALREG TYPE=    |
| Suppresses the intercept parameter                                       | MODEL COVEST=             |
|                                                                        | 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=             |
| Outputs the expected value of the response variable                      | OUTPUT PRED=              |
| Outputs estimates of $x_i'\beta$                                         | 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=\texttt{CAS-libref.data-table}

names the input data table for PROC CSPATIALREG to use. \texttt{CAS-libref.data-table} is a two-level name, where

\texttt{CAS-libref} refers to a collection of information that is defined in the LIBNAME statement and includes the \texttt{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 \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 402.

\texttt{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 \texttt{option}:

WMAT=\texttt{CAS-libref.data-table}

specifies the secondary spatial weights data table, which you can use to construct the spatial weights matrix $W$. \texttt{CAS-libref.data-table} is a two-level name, where \texttt{CAS-libref} refers to the \texttt{caslib} and session identifier, and \texttt{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 402.

Loosely speaking, the entries of $W$, $w(s_i, s_j)$, 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 $j = 1, 2, \ldots, n$ 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 $W$, see the \texttt{NONORMALIZE} option.

The $W$ 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 \texttt{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 \texttt{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 \texttt{SPATIALID} statement to match observations in the data tables that you specify in the \texttt{DATA=} and \texttt{WMAT=} options. For more information about the \texttt{SPATIALID} statement, see the section “\texttt{SPATIALID Statement}” on page 419. For more information about the compact representation of the spatial weights matrix, see the section “Compact Representation of the Spatial Weights Matrix” on page 431.

For all models except a purely linear model, you can also specify the following \texttt{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 \( w(s_i, s_j) \), 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:

\[ \textsc{approx-option} \]

specifies options that are related to approximating the Jacobian, as described in the section “Approximations to the Jacobian” on page 435. To invoke approximation, you must specify one or more of the following approx-options:

\[ \textsc{chebyshev} | \textsc{taylor} \]

specifies the approximation method. By default, Chebyshev approximation is used. The Taylor approximation is used only if you specify the TAYLOR option.

\[ \text{NMC}=\text{number} \]

specifies a positive integer as the number of standard random normal draws for Monte Carlo simulation. By default, NMC=100.

\[ \text{ORDER}=\text{number} \]

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.

\[ \text{SEED}=\text{number} \]

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:

\[ \text{CORRB} \]

prints the correlation matrix of the parameter estimates.

\[ \text{COVB} \]

prints the covariance matrix of the parameter estimates.

\[ \text{NOPRINT} \]

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:
**MAXFUNCI**
**MAXFUI**
specifies the maximum number of function calls in the optimization process. By default, MAXFUNCI=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.

**MAXITERI**
**MAXITI**
specifies the maximum number of iterations in the optimization process. By default, MAXITERI=200.

**MAXTIME=ri**
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 418.

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 < x_1-x_{10} < 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Allows design columns for a variable to enter or leave the model independently</td>
</tr>
</tbody>
</table>
DISPLAY Statement

\[
\text{DISPLAY} \ < \text{table-list}> \ < \ / \text{options}> \ ;
\]

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 442. 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 subsetted 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 442. 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 436.

**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=\texttt{CAS-libref.data-table} < options > < keyword \texttt{=}name > \ldots < keyword \texttt{=}name > ;

The OUTPUT statement creates a data table that contains observationwise statistics that PROC CSPATIAL-REG computes after fitting the model. These observationwise statistics include the estimates of $x_i'\beta$, 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 \texttt{COPYV AR=} option.

You can specify only one OUTPUT statement. You must specify the following option:

\texttt{OUT=} \texttt{CAS-libref.data-table}

names the output data table for PROC CSPATIALREG to use. You must specify this option before any other options. \texttt{CAS-libref.data-table} is a two-level name, where

- \texttt{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 \texttt{CAS-libref}, see the section “Using CAS Sessions and CAS Engine Librefs” on page 402.

- \texttt{data-table} specifies the name of the output data table.

You can also specify the following \texttt{options}:

\texttt{COPYV AR=} \texttt{variable}
\texttt{COPYV ARS=} \texttt{variables}

transfers one or more variables from the primary input data table to the output data table.

\texttt{PRED=} \texttt{name}
\texttt{MEAN=} \texttt{name}

assigns a name to the variable that contains the predicted value of the response variable.

\texttt{RESID=} \texttt{name}
\texttt{RESIDUAL=} \texttt{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).

\texttt{XBETA=} \texttt{name}

names the variable that contains estimates of $x_i'\beta$. 
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 436.

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 \( \rho \) 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 \( z_1, \ldots, z_q \) 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 \( z_1, \ldots, z_q \). 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 Q (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 436. 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:

```r
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 NONORMALIZE 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 \( i = 1, 2, \ldots, n \). For these spatial units, let an \( n \times n \) matrix \( W \) with nonnegative elements be a spatial weights matrix and let \( x_i \) 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

\[
y_i = \rho \sum_{j=1}^{n} W_{ij} y_j + x'_i \beta + \epsilon_i
\]

where \( i = 1, 2, \ldots, n \), \( \rho \) is the spatial autoregressive coefficient, \( \beta \) is a \( p \times 1 \) parameter vector, \( W_{ij} \) is the \((i,j)\)th element of the matrix \( W \) subject to \( W_{ii} = 0 \), and the error term \( \epsilon_i \) that is related to the spatial unit \( s_i \) is assumed to follow \( \epsilon_i \overset{iid}{\sim} \mathcal{N}(0,\sigma^2) \) for \( i = 1, 2, \ldots, n \).

The SAR model is often described in vector form as

\[
y = \rho W y + X \beta + \epsilon
\]

where \( y = (y_1, y_2, \ldots, y_n)' \), \( X \) is an \( n \times p \) matrix where each row consists of \( x'_i \), and \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \).

The standard estimator for the SAR model is the maximum likelihood estimator (MLE). For the SAR model, the log-likelihood function is (Anselin 2001)

\[
L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(Ay - X\beta)'(Ay - X\beta)}{2\sigma^2} + \ln|A|
\]

where \( A = I_n - \rho W \), \( I_n \) is an \( n \times n \) identity matrix, and \( |A| \) denotes the determinant of \( A \).

You can derive the gradients as follows:

\[
\frac{\partial L}{\partial \beta} = \frac{X'(Ay - X\beta)}{\sigma^2}
\]

\[
\frac{\partial L}{\partial \rho} = \frac{1}{\sigma^2} y'W'(Ay - X\beta) - \text{tr}(A^{-1}W)
\]

\[
\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^4} + \frac{(Ay - X\beta)'(Ay - X\beta)}{2\sigma^4}
\]

For the \( n \times n \) matrix \( A \), \( \text{tr}(A) = \sum_{i=1}^{n} a_{ii} \), where \( a_{ii} \) is the \( i \)th diagonal element of \( A \).

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 \( i = 1, 2, \ldots, n \). For these spatial units, let \( W \) be an \( n \times n \) spatial weights matrix of your choice. Further
assume that $x_i$ is a $p \times 1$ vector that denotes values of $p$ regressors that are recorded for the spatial unit $s_i$ and assume that $z_i$ 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)

$$ y = \rho Wy + X\beta + WZ\theta + \epsilon $$

where $y = (y_1, y_2, \ldots, y_n)'$, $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'$ with $\epsilon_i \sim \text{iid} N(0, \sigma^2)$, $X$ is an $n \times p$ matrix where each row consists of $x_i'$, $Z$ is an $n \times q$ matrix where each row consists of $z_i'$, and $\beta$ and $\theta$ are $p \times 1$ and $q \times 1$ parameter vectors, respectively.

By letting $\tilde{X} = [X \ WZ]$ and $\tilde{\theta} = (\beta' \ \theta')'$, you can rewrite the SDM as

$$ y = \rho Wy + \tilde{X}\tilde{\theta} + \epsilon $$

The log-likelihood function for the SDM is

$$ L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(Ay - \tilde{X}\tilde{\theta})'(Ay - \tilde{X}\tilde{\theta})}{2\sigma^2} + \ln |A| $$

where $A = I_n - \rho W$.

For the SDM, the gradients are

$$ \frac{\partial L}{\partial \beta} = \frac{\tilde{X}'(Ay - \tilde{X}\tilde{\theta})}{\sigma^2} $$

$$ \frac{\partial L}{\partial \rho} = \frac{1}{2\sigma^2} y'W'(Ay - \tilde{X}\tilde{\theta}) - \text{tr}(A^{-1}W) $$

$$ \frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^4} + \frac{(Ay - \tilde{X}\tilde{\theta})'(Ay - \tilde{X}\tilde{\theta})}{2\sigma^4} $$

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 $i = 1, 2, \ldots, n$. For these spatial units, let $W$ be an $n \times n$ spatial weights matrix. Further, let $x_i$ 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),

$$ y = X\beta + u $$

$$ u = \lambda Wu + \epsilon $$

where $y = (y_1, y_2, \ldots, y_n)'$, $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'$ with $\epsilon_i \sim \text{iid} N(0, \sigma^2)$, $X$ is an $n \times p$ matrix where each row consists of $x_i'$, and $\beta$ is a $p \times 1$ parameter vector.
The log-likelihood function for the SEM is
\[
\mathcal{L} = \frac{-n}{2} \ln(2\pi \sigma^2) - \frac{[B(y - X\beta)'] [B(y - X\beta)]}{2\sigma^2} + \ln |B|
\]
where \( B = I_n - \lambda W \).

For the SEM, the gradients are
\[
\frac{\partial \mathcal{L}}{\partial \beta} = \frac{(BX)' [B(y - X\beta)]}{\sigma^2}
\]
\[
\frac{\partial \mathcal{L}}{\partial \lambda} = \frac{1}{\sigma^2} [W(y - X\beta)'] [B(y - X\beta)] - \text{tr}(B^{-1}W)
\]
\[
\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[B(y - X\beta)'] [B(y - X\beta)]}{2\sigma^4}
\]

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 \( i = 1, 2, \ldots, n \). For these spatial units, let \( W \) be an \( n \times n \) spatial weights matrix. Further, let \( x_i \) be a \( p \times 1 \) vector that denotes values of \( p \) regressors that are recorded at unit \( s_i \), and let \( z_i \) 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),
\[
y = X\beta + WZ\theta + u
\]
\[
u = \lambda Wu + \epsilon
\]
where \( y = (y_1, y_2, \ldots, y_n)' \), \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \) with \( \epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2) \), \( X \) is an \( n \times p \) matrix where each row consists of \( x_i' \), \( Z \) is an \( n \times q \) matrix where each row consists of \( z_i' \), and \( \beta \) and \( \theta \) are \( p \times 1 \) and \( q \times 1 \) parameter vectors, respectively.

By letting \( \widetilde{X} = [X \ WZ] \) and \( \widetilde{\beta} = (\beta' \ \theta')' \), you can rewrite the SDEM as
\[
y = \widetilde{X}\widetilde{\beta} + B^{-1} \epsilon
\]
where \( B = I_n - \lambda W \).

The log-likelihood function for the SDEM is
\[
\mathcal{L} = \frac{-n}{2} \ln(2\pi \sigma^2) - \frac{[B(y - \widetilde{X}\widetilde{\beta})'] [B(y - \widetilde{X}\widetilde{\beta})]}{2\sigma^2} + \ln |B|
\]

For the SDEM, the gradients are
\[
\frac{\partial \mathcal{L}}{\partial \widetilde{\beta}} = \frac{(B\widetilde{X})' [B(y - \widetilde{X}\widetilde{\beta})]}{\sigma^2}
\]
Chapter 11: The CSPATIALREG Procedure

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 $i = 1, 2, \ldots, n$. For these spatial units, let $W$ be an $n \times n$ spatial weights matrix, and let $x_i$ 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,

$$
y = X\beta + u
$$
$$
 u = (I_n - \lambda W)e
$$

where $y = (y_1, y_2, \ldots, y_n)'$, $e = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'$ with $\epsilon_i \overset{iid}{\sim} N(0, \sigma^2)$, $X$ is an $n \times p$ matrix that has $x'_i$ in each row, $Z$ is an $n \times q$ matrix that has $z'_i$ in each row, and $\beta$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SMA model is

$$
\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[B^{-1}(y - X\beta)]' [B^{-1}(y - X\beta)]}{2\sigma^4} - \ln |B|
$$

where $B = I_n - \lambda W$.

For the SMA model, the gradients are

$$
\frac{\partial \mathcal{L}}{\partial \beta} = \frac{(B^{-1}X)' [B^{-1}(y - X\beta)]}{\sigma^2}
$$
$$
\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [B^{-1}(y - X\beta)]' [B^{-1}W] [B^{-1}(y - X\beta)] + \text{tr}(B^{-1}W)
$$
$$
\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[B^{-1}(y - X\beta)]' [B^{-1}(y - X\beta)]}{2\sigma^4}
$$

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 $i = 1, 2, \ldots, n$. For these spatial units, let $W$ be an $n \times n$ spatial weights matrix. Further, let $x_i$ be a $p \times 1$ vector that denotes values of $p$ regressors that are recorded at unit $s_i$, and let $z_i$ 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

$$
y = X\beta + WX\theta + (I_n - \lambda W)e
$$
Spatial Autoregressive Moving Average Models

where \( y = (y_1, y_2, \ldots, y_n)' \); \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \) with \( \epsilon_i \sim \text{N}(0, \sigma^2) \); \( X \) and \( Z \) are \( n \times p \) and \( n \times q \) matrices, where each row consists of \( x'_i \) and \( z'_i \), respectively; \( \beta \) is a \( p \times 1 \) parameter vector; and \( \theta \) is a \( q \times 1 \) parameter vector.

By letting \( \widetilde{X} = [X \ WZ] \) and \( \widetilde{B} = (\beta' \ \theta')' \), the SDMA model can be written as

\[
y = \widetilde{X} \widetilde{\beta} + B \epsilon
\]

The log-likelihood function for the SDMA model is

\[
\mathcal{L} = -\frac{n}{2} \ln(2\pi \sigma^2) - \frac{(B^{-1}(y - \widetilde{X} \widetilde{\beta}))'[B^{-1}(y - \widetilde{X} \widetilde{\beta})]}{2\sigma^2} + \ln |B|
\]

where \( B = I_n - \lambda W \) and \( |B| \) denotes the determinant of matrix \( B \).

For the SDMA model, the gradients are

\[
\frac{\partial \mathcal{L}}{\partial \beta} = \frac{(B^{-1} \widetilde{X})'[B^{-1}(y - \widetilde{X} \widetilde{\beta})]}{\sigma^2}
\]

\[
\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} [B^{-1}(y - \widetilde{X} \widetilde{\beta})]'[B^{-1}W][B^{-1}(y - \widetilde{X} \widetilde{\beta})] + \text{tr}(B^{-1}W)
\]

\[
\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(B^{-1}(y - \widetilde{X} \widetilde{\beta}))'[B^{-1}(y - \widetilde{X} \widetilde{\beta})]}{2\sigma^4}
\]

**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 \( i = 1, 2, \ldots, n \). For these spatial units, let the \( n \times n \) matrices \( W_1 \) and \( W_2 \) be two spatial weights matrices that contain nonnegative elements. In practice, \( W_1 \) and \( W_2 \) can be identical. Further, it is assumed that \( x_i \) 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),

\[
y = \rho W_1 y + X \beta + u
\]

\[
u = (I_n - \lambda W_2) \epsilon
\]

where \( y = (y_1, y_2, \ldots, y_n)' \), \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \) with \( \epsilon_i \sim \text{N}(0, \sigma^2) \), \( X \) is an \( n \times p \) matrix that consists of \( x'_i \) in each row, \( \beta \) is a \( p \times 1 \) parameter vector, and \( I_n \) is an \( n \times n \) identity matrix.

The log-likelihood function for the SARMA model is

\[
\mathcal{L} = -\frac{n}{2} \ln(2\pi \sigma^2) - \frac{(B^{-1}(Ay - X \beta))'[B^{-1}(Ay - X \beta)]}{2\sigma^2} + \ln |A| - \ln |B|
\]

where \( A = I_n - \rho W_1 \), \( B = I_n - \lambda W_2 \), and \( | \cdot | \) denotes the matrix determinant operator.
For the SARMA model, the gradients are
\[
\frac{\partial \mathcal{L}}{\partial \beta} = \frac{(B^{-1}X)' \left[ B^{-1} (Ay - X\beta) \right]}{\sigma^2} \\
\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(B^{-1}W_1 y)' B^{-1} (Ay - X\beta)}{\sigma^2} - \text{tr}(A^{-1}W_1) \\
\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} \left[ B^{-1} (Ay - X\beta) \right]' \left[ B^{-1} W_2 \right] \left[ B^{-1} (Ay - X\beta) \right] + \text{tr}(B^{-1}W_2) \\
\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} \left[ B^{-1} (Ay - X\beta) \right]' \left[ B^{-1} (Ay - X\beta) \right]
\]

**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 \( i = 1, 2, \ldots, n \). For these spatial units, let \( W_1 \) and \( W_2 \) be two spatial weights matrices. Further, let \( x_i \) be a \( p \times 1 \) vector that denotes values of \( p \) regressors that are recorded at unit \( s_i \), and let \( z_i \) 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,
\[
y = \rho W_1 y + X\beta + W_1 Z\theta + u \\
u = (I_n - \lambda W_2)\epsilon
\]
where \( y = (y_1, y_2, \ldots, y_n)' \), \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \) with \( \epsilon_i \overset{iid}{\sim} N(0, \sigma^2)' \), \( X \) is an \( n \times p \) matrix that has \( x_i' \) in each row, \( Z \) is an \( n \times q \) matrix that has \( z_i' \) in each row, and \( \beta \) is a \( p \times 1 \) parameter vector.

By letting \( \tilde{X} = [X \ W_1 Z] \) and \( \tilde{\beta} = (\beta' \ \theta')' \), the SDARMA model can be written as
\[
y = \rho W_1 y + \tilde{X}\tilde{\beta} + (I_n - \lambda W_2)\epsilon
\]

The log-likelihood function for the SDARMA model is
\[
\mathcal{L} = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[B^{-1} (Ay - \tilde{X}\tilde{\beta})]' [B^{-1} (Ay - \tilde{X}\tilde{\beta})]}{2\sigma^2} + \ln |A| - \ln |B|
\]
where \( A = I_n - \rho W_1 \) and \( B = I_n - \lambda W_2 \).

For the SDARMA model, the gradients are
\[
\frac{\partial \mathcal{L}}{\partial \tilde{\beta}} = \frac{(B^{-1}\tilde{X})' \left[ B^{-1} (Ay - \tilde{X}\tilde{\beta}) \right]}{\sigma^2} \\
\frac{\partial \mathcal{L}}{\partial \rho} = \frac{(B^{-1}W_1 y)' B^{-1} (Ay - \tilde{X}\tilde{\beta})}{\sigma^2} - \text{tr}(A^{-1}W_1) \\
\frac{\partial \mathcal{L}}{\partial \lambda} = -\frac{1}{\sigma^2} \left[ B^{-1} (Ay - \tilde{X}\tilde{\beta}) \right]' \left[ B^{-1} W_2 \right] \left[ B^{-1} (Ay - \tilde{X}\tilde{\beta}) \right] + \text{tr}(B^{-1}W_2) \\
\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{n}{2\sigma^2} \left[ B^{-1} (Ay - \tilde{X}\tilde{\beta}) \right]' \left[ B^{-1} (Ay - \tilde{X}\tilde{\beta}) \right]
\]
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 $i = 1, 2, \ldots, n$. For these spatial units, let $W_1$ and $W_2$ be two spatial weights matrices, and let $x_i$ 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),

$$
y = \rho W_1 y + X \beta + u
$$

$$
u = \lambda W_2 u + \epsilon
$$

where $y = (y_1, y_2, \ldots, y_n)'$, $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'$ with $\epsilon_i \sim N(0, \sigma^2)'$, $X$ is an $n \times p$ matrix that has $x_i'$ in each row, and $\beta$ is a $p \times 1$ parameter vector.

The log-likelihood function for the SAC model is

$$
L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{[B(Ay - X\beta)]' [B(Ay - X\beta)]}{2\sigma^2} + \ln|A| + \ln|B|
$$

where $A = I_n - \rho W_1$ and $B = I_n - \lambda W_2$.

For the SAC model, the gradients are

$$
\frac{\partial L}{\partial \beta} = \frac{(BX)' [B(Ay - X\beta)]}{\sigma^2}
$$

$$
\frac{\partial L}{\partial \rho} = \frac{(BW_1 y)' B(Ay - X\beta)}{\sigma^2} - \text{tr}(A^{-1}W_1)
$$

$$
\frac{\partial L}{\partial \lambda} = \frac{1}{\sigma^2} [W_2(Ay - X\beta)]' [B(Ay - X\beta)] - \text{tr}(B^{-1}W_2)
$$

$$
\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{[B(Ay - X\beta)]' [B(Ay - X\beta)]}{2\sigma^2}
$$

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 $i = 1, 2, \ldots, n$. For these spatial units, let $W_1$ and $W_2$ be two spatial weights matrices. Further, let $x_i$ be a $p \times 1$ vector that denotes values of $p$ regressors that are recorded at unit $s_i$, and assume that $z_i$ 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,

$$
y = \rho W_1 y + X \beta + W_1 Z \theta + u
$$
where \(y = (y_1, y_2, \ldots, y_n)'\), \(\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'\) with \(\epsilon_i \overset{iid}{\sim} N(0, \sigma^2)\), \(X\) is an \(n \times p\) matrix that has \(x_{i}'\) in each row, \(Z\) is an \(n \times q\) matrix that has \(z_{i}'\) in each row, and \(\beta\) is a \(p \times 1\) parameter vector.

By letting \(\tilde{X} = [X \ W_1 Z]\) and \(\tilde{\beta} = (\beta' \ \theta')'\), the SDAC model can be rewritten as

\[
y = \rho W_1 y + \tilde{X}\tilde{\beta} + (I_n - \lambda W_2)^{-1} \epsilon
\]

The log-likelihood function for the SDAC model is

\[
L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \left[ B(Ay - \tilde{X}\tilde{\beta}) \right]' \left[ B(Ay - \tilde{X}\tilde{\beta}) \right] + \ln |A| + \ln |B|
\]

For the SDAC model, the gradients are

\[
\frac{\partial L}{\partial \tilde{\beta}} = \frac{(\tilde{X}\tilde{\beta})' [B(Ay - \tilde{X}\tilde{\beta})]}{\sigma^2}
\]

\[
\frac{\partial L}{\partial \rho} = \frac{2 \rho (W_1 y)' B (Ay - \tilde{X}\tilde{\beta})}{\sigma^2} - \text{tr}(A^{-1}W_1)
\]

\[
\frac{\partial L}{\partial \lambda} = \frac{1}{\sigma^2} \left[ W_2 (Ay - \tilde{X}\tilde{\beta}) \right]' \left[ B(Ay - \tilde{X}\tilde{\beta}) \right] - \text{tr}(B^{-1}W_2)
\]

\[
\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \left[ B(Ay - \tilde{X}\tilde{\beta}) \right]' \left[ B(Ay - \tilde{X}\tilde{\beta}) \right]
\]

**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 \(i = 1, 2, \ldots, n\), and let \(x_i\) 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

\[
y = X\beta + \epsilon
\]

where \(y = (y_1, y_2, \ldots, y_n)'\), \(\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)'\) with \(\epsilon_i \overset{iid}{\sim} N(0, \sigma^2)\), and \(X\) is an \(n \times p\) matrix where each row consists of \(x_{i}'\).

The log-likelihood function for the linear regression model is

\[
L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(y - X\beta)'(y - X\beta)}{2\sigma^2}
\]

For the linear regression model, the gradients are

\[
\frac{\partial L}{\partial \beta} = \frac{X' (y - X\beta)}{\sigma^2}
\]
\[
\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(y - X\beta)'(y - X\beta)}{2\sigma^4}
\]

The Hessians take the following forms:

\[
\frac{\partial^2 L}{\partial \beta \partial \beta'} = -\frac{X'X}{\sigma^2}
\]
\[
\frac{\partial^2 L}{\partial \beta \partial \sigma^2} = -\frac{X'(y - X\beta)}{\sigma^4}
\]
\[
\frac{\partial^2 L}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(y - X\beta)'(y - X\beta)}{\sigma^6}
\]

**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 \( i = 1, 2, \ldots, n \). For these spatial units, let \( W \) be a spatial weights matrix. Further, let \( x_i \) be a \( p \times 1 \) vector that denotes values of \( p \) regressors recorded at unit \( s_i \), and let \( z_i \) 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

\[
y = X\beta + WZ\theta + \epsilon
\]

where \( y = (y_1, y_2, \ldots, y_n)' \), \( \epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)' \) with \( \epsilon_i \overset{\text{iid}}{\sim} \text{N}(0, \sigma^2) \), \( X \) is an \( n \times p \) matrix where each row consists of \( x_i' \), \( Z \) is an \( n \times q \) matrix where each row consists of \( z_i' \), and \( \beta \) is a \( p \times 1 \) parameter vector.

By letting \( \tilde{X} = [X \ WZ] \) and \( \tilde{\beta} = (\beta' \ \theta')' \), you can rewrite the SLX model as

\[
y = \tilde{X}\tilde{\beta} + \epsilon
\]

The log-likelihood function for the SLX model is

\[
L = -\frac{n}{2} \ln(2\pi\sigma^2) - \frac{(y - \tilde{X}\tilde{\beta})'(y - \tilde{X}\tilde{\beta})}{2\sigma^2}
\]

For the SLX model, the gradients are

\[
\frac{\partial L}{\partial \tilde{\beta}} = \frac{(\tilde{X})'(y - \tilde{X}\tilde{\beta})}{\sigma^2}
\]
\[
\frac{\partial L}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{(y - \tilde{X}\tilde{\beta})'(y - \tilde{X}\tilde{\beta})}{2\sigma^4}
\]

The Hessians take the following forms:

\[
\frac{\partial^2 L}{\partial \tilde{\beta} \partial \tilde{\beta}'} = -\frac{\tilde{X}'\tilde{X}}{\sigma^2}
\]
\[
\frac{\partial^2 L}{\partial \tilde{\beta} \partial \sigma^2} = -\frac{\tilde{X}'(y - \tilde{X}\tilde{\beta})}{\sigma^4}
\]
\[
\frac{\partial^2 L}{\partial \sigma^4} = \frac{n}{2\sigma^4} - \frac{(y - \tilde{X}\tilde{\beta})'(y - \tilde{X}\tilde{\beta})}{\sigma^6}
\]
Specifying the Spatial Weights Matrix

The spatial weights matrix \(W\) plays a vital role in spatial econometric modeling. If you fit a purely linear model without SLX effects, you do not need a \(W\) 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 \(W\) matrix is often problem-specific, there are some general guidelines to consider. Two common ways to create the \(W\) matrix are \(k\)-order binary contiguity matrices and \(k\)-nearest-neighbor matrices (Elhorst 2013).

\(k\)-Order Binary Contiguity Matrices

To create the \(W\) matrix by using \(k\)-order binary contiguity matrices, you start with the spatial contiguity matrix \(C\). In the case of the first-order neighbors \((k = 1)\), a value of 1 for the \((i, j)\)th entry in \(C\) 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:

\[
C = \begin{pmatrix}
\text{SID} & L1 & L2 & L3 & L4 \\
L1 & 0 & 1 & 0 & 1 \\
L2 & 1 & 0 & 0 & 0 \\
L3 & 0 & 0 & 0 & 1 \\
L4 & 1 & 0 & 1 & 0 \\
\end{pmatrix}
\]

The diagonal elements of \(C\) 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 \(W\) by row-standardizing the contiguity matrix \(C\). To do so, you divide entries in each row of \(C\) by the sum of that row. The spatial weights matrix \(W\), which is the row-standardized version of \(C\), is as follows:

\[
W = \begin{pmatrix}
\text{SID} & L1 & L2 & L3 & L4 \\
L1 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\
L2 & 1 & 0 & 0 & 0 \\
L3 & 0 & 0 & 0 & 1 \\
L4 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
\end{pmatrix}
\]

\(k\)-Nearest-Neighbor Matrices

To create the \(W\) 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

\[
d_{ij} = \sqrt{(\text{lat}_i - \text{lat}_j)^2 + (\text{lon}_i - \text{lon}_j)^2}
\]
Compact Representation of the Spatial Weights Matrix

After computing the distance between the unit \(i\) and other units under a certain metric, you sort \(d_{ij}\) in ascending order; for example, \(d_{ij_1} \leq d_{ij_2} \leq \cdots \leq d_{ij_k} \leq \cdots \leq d_{ijn-1}\). For a given \(k\), let \(N_k(i) = \{j_1, j_2, \ldots, j_k\}\) be the set that contains the indices of \(k\)-nearest neighbors of unit \(i\); then the \((i, j)\)th entries of the contiguity matrix \(C\) 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 \(W\) 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 \(C\) 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 \(C^*\) 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 \(d_{\text{cutoff}}\) 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 NONORMALIZE 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 \(C\) or the spatial weights matrix \(W\) increases dramatically. To circumvent the storage issue, PROC CSPATIALREG enables you to provide a compact representation of \(W\) (or \(C\)) 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 \(W\) (or \(C\)), 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 \(W\) (or \(C\)) 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 \(W\) (or \(C\)).

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 \( W \),

\[
W = \begin{pmatrix}
0 & 0.5 & 0 & 0.5 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0.5 & 0 & 0.5 & 0
\end{pmatrix}
\]

would look like the following:

```r
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 \( C \),

\[
C = \begin{pmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0
\end{pmatrix}
\]

the compact representation would look like the following:

```r
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 430, its compact representation would be as follows:

```r
data Ws2;
  input SID $ cSID $ Weight;
datalines;
  L1 L2 0.5
  L1 L4 0.5
  L2 L1 1.0
  L3 L4 1.0
```
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
;run;
```
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 \( \rho \) and \( \lambda \) 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 \( I - \rho W_1 \), and the parameter space of \( \rho \) is specified such that \( I - \rho W_1 \) is nonsingular. For the SEM, SDEM, SMA, and SDMA models, the Jacobian term involves the log-determinant of \( I - \lambda W_1 \), and the parameter space of \( \lambda \) is specified such that \( I - \lambda W_1 \) is nonsingular. For the SAC, SDAC, SARMA, and SDARMA models, both \( I - \rho W_1 \) and \( I - \lambda W_1 \) must be nonsingular. As a result, the parameter space of \( \rho \) and \( \lambda \) is often specified such that both \( I - \rho W_1 \) and/or \( I - \lambda W_1 \) are nonsingular.

In the CSPATIALREG procedure, the parameter space of the parameters \( \rho \) and \( \lambda \) depends on the spatial weights matrix \( W \) that you choose. For \( W \), the parameter space of the parameters \( \rho \) and \( \lambda \) in PROC CSPATIALREG is determined as follows:

1. For a symmetric \( W \), the nonsingularity condition requires \( \rho \in \left( \omega_{\min}^{-1}, \omega_{\max}^{-1} \right) \) and \( \lambda \in \left( \omega_{\min}^{-1}, \omega_{\max}^{-1} \right) \). Here \( \omega_{\min} \) and \( \omega_{\max} \) denote the smallest (that is, most negative) and largest real eigenvalues of \( W \), respectively.

2. If \( W \) is symmetric and subsequently row-standardized, the nonsingularity condition requires \( \rho \in \left( \omega_{\min}^*, -1, 1 \right) \) and \( \lambda \in \left( \omega_{\min}^*, -1, 1 \right) \). Here \( \omega_{\min}^* \) denotes the smallest purely real eigenvalue of the row-standardized \( W \).

3. If \( W \) is asymmetric and subsequently row-standardized, the nonsingularity condition requires \( \rho \in \left( r_{\min}^*, -1, 1 \right) \) and \( \lambda \in \left( r_{\min}^*, -1, 1 \right) \). Here \( r_{\min}^* \) denotes the smallest purely real eigenvalue of the row-standardized \( W \).

4. When Taylor or Chebyshev approximation is used for SDMs and SAR models, \( W \) is required to be row-standardized. In these cases, the restriction on the spatial autoregressive coefficient \( \rho \) is \( \rho \in (-1, 1) \).
5. When Taylor or Chebyshev approximation is used for SEMs and SDEMs, \( W \) is required to be row-standardized. In these cases, the restriction on the spatial autocorrelation coefficient \( \lambda \) 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 \( \ln |I_n - \rho W| \) for SDMs and SAR models and \( \ln |I_n - \lambda W| \) for SEMs and SDEMs, where \( n \) is the number of observations and \( W \) is the spatial weights matrix. When \( n \) is not large, the Jacobian is computed as follows,

\[
\ln |I_n - \rho W| = \sum_{i=1}^{n} \ln |1 - \rho \omega_i|
\]

where \( \omega_i \) are the eigenvalues of \( W \). Such a method requires you to precompute all eigenvalues of \( W \). 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

  \[
  \ln |I_n - \rho W| \approx -\sum_{k=1}^{q} \rho^k \frac{\text{tr}(W^k)}{k}
  \]

- **Chebyshev approximation** uses finite, lower-order Chebyshev polynomials of the first kind to approximate the log-determinant as

  \[
  \ln |I_n - \rho W| \approx \sum_{k=0}^{q} c_k(\rho) \text{tr}(T_k(W))
  \]

where \( T_0(W) = I_n \), \( T_1(W) = W \), and \( T_{k+1}(W) = 2WT_k(W) - T_{k-1}(W) \) for \( k = 1, 2, \ldots, q \). 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 \( j = 0, 1, \ldots, q \).

The traces of powers of \( W \) can be computed exactly or approximated using Monte Carlo simulation. The Monte Carlo simulation is done as follows,

\[
\text{tr}(W^k) \approx \frac{1}{M} \sum_{l=1}^{M} n \frac{n}{u_i' u_l} u_i' W^k u_l
\]
where \( \mathbf{u}_t \overset{iid}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_n) \) 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 \texttt{APPROXIMATION=} option in the PROC CSPATIALREG statement. For the approximation method, you can use the keyword \texttt{TAYLOR} in the \texttt{APPROXIMATION=} option to request Taylor approximation. Otherwise, the approximation method defaults to Chebyshev approximation. You specify \texttt{ORDER=} \( q \) in the \texttt{APPROXIMATION=} option to request a series of order \( q \) when approximating the log-determinant. In addition, you specify \texttt{NMC=} \( M \) in the \texttt{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 \texttt{SEED=} suboption of the \texttt{APPROXIMATION=} option to specify an integer seed for a random number generator to replicate your analysis.

---

### 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

\[
\text{model } y = x_1 - x_3 \text{ / type=SAR;}
\]

where \( x_1 - x_3 \) are continuous variables. Suppose you want to restrict the parameter that is associated with the regressor \( x_3 \) to be greater than 1.7. You should provide the following statement:

\[
\text{RESTRICT } x_3 > 1.7;
\]

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 \( \mathbf{W}_\) 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 \textit{W}_x3 and provide the following statement:

\textbf{RESTRICT} \textit{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 \textit{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.

\section*{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 \textit{C} and has three levels: 0, 1, and 2. Suppose that your model is the following:

\begin{verbatim}
  class C;
  model y = x1 x2 C;
\end{verbatim}

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 \textit{C} to always be greater than 0.7, you refer to the parameter as \textit{C}_0 and provide the following statement:

\textbf{RESTRICT} \textit{C}_0 > 0.7;

\section*{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:

\begin{verbatim}
  model y = x1 x2 x3*x4;
\end{verbatim}

You can form the name of the parameter that is associated with the interaction regressor \textit{x3}*\textit{x4} by replacing the multiplication sign with an underscore. Thus, \textit{x3}_x4 refers to the parameter that is associated with the interaction regressor \textit{x3}*\textit{x4}.

You refer to interactions between regressors and CLASS variables in exactly the same way. Suppose you have a CLASS variable named \textit{C} that has three levels (0, 1, and 2), and your model is the following:
The interaction between the continuous variable \(x_3\) and the CLASS variable \(C\) introduces three additional parameters, which are named \(x_3\_C\_0\), \(x_3\_C\_1\), and \(x_3\_C\_2\). Note that, although the order of the terms in the interaction is \(C\) followed by \(x_3\), the name of the parameter that is associated with the interaction is formed by placing the name of the continuous variable \(x_3\) 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:

```plaintext
class C;
model y = x1 x2 x3(C);
```

The nested interaction between the continuous variable \(x_3\) and the CLASS variable \(C\) introduces three additional parameters, which are named \(x_3\_C\_0\), \(x_3\_C\_1\), and \(x_3\_C\_2\). 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:

```plaintext
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 $8 \times (p + p \times (p + 1)/2)$ 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 411.
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 $\rho$ and the variance of the error terms $\sigma^2$. For the SEMs and SDEMs, estimates are displayed for the spatial autocorrelation coefficient $\lambda$ and the variance of the error terms $\sigma^2$. For the linear and SLX models, estimates are displayed for the variance of the error terms $\sigma^2$.

The internal name of the spatial autoregressive coefficient $\rho$ 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 $\lambda$ in the SEMs and SDEMs, and “_sigma2” is the internal name of the variance parameter $\sigma^2$.

**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 $x^T\beta$, 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>
<thead>
<tr>
<th>Table Name</th>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassInfo</td>
<td>Level information from the CLASS statement</td>
<td>CLASS</td>
<td>Default</td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>CovB</td>
<td>Correlation of parameter estimates</td>
<td>MODEL</td>
<td>CORRB</td>
</tr>
<tr>
<td>CovB</td>
<td>Covariance of parameter estimates</td>
<td>MODEL</td>
<td>COVB</td>
</tr>
<tr>
<td>FitModelSummary</td>
<td>Summary of nonlinear estimation</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Parameter estimates</td>
<td>MODEL</td>
<td>Default</td>
</tr>
<tr>
<td>TestResults</td>
<td>Hypothesis test statistics</td>
<td>TEST</td>
<td>Default</td>
</tr>
<tr>
<td>TimingDetails</td>
<td>Detailed summary of time used for all phases of execution</td>
<td>PROC</td>
<td>PRINNTIMING=(DETAILS)</td>
</tr>
<tr>
<td>TimingSummary</td>
<td>Summary of time used for main phases of execution</td>
<td>PROC</td>
<td>PRINNTIMING</td>
</tr>
</tbody>
</table>

**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:

```plaintext
proc means data=mycas.CrimeOH;
   var crime income hvalue;
run;
```
Example 11.1: Columbus Crime Data

Output 11.1.1 Summary Statistics

The MEANS Procedure

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>crime</td>
<td>49</td>
<td>35.1288367</td>
<td>16.7320385</td>
<td>0.1780000</td>
<td>68.8920000</td>
</tr>
<tr>
<td>income</td>
<td>49</td>
<td>14.3749388</td>
<td>5.7033781</td>
<td>4.4770000</td>
<td>31.0700000</td>
</tr>
<tr>
<td>hvalue</td>
<td>49</td>
<td>38.4362245</td>
<td>18.4660693</td>
<td>17.9000000</td>
<td>96.4000000</td>
</tr>
</tbody>
</table>

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:

```plaintext
/*-- SAR --*/
  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 \( \rho \) 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 | 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 | 0.086218       | -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 --*/
  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 $\rho$ 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**

| Parameter     | DF   | Estimate  | Standard Error | t Value | Approx Pr > |t| |
|---------------|------|-----------|----------------|---------|--------------|---|
| Intercept     | 1    | 42.803457 | 13.92447       | 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 --*/
  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 $\lambda$ 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 | 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:

```plaintext
/*-- SDEM --*/
  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 | 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:
Chapter 11: The CSPATIALREG Procedure

The parameter estimates are shown in Output 11.1.6.

Output 11.1.6 Parameter Estimates of SMA Model

The CSPATIALREG Procedure

| Parameter   | DF | Estimate  | Standard Error | t Value | 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:

```
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    | DF  | Estimate | Standard Error | t Value | 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:

```plaintext
   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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Approx Pr &gt;</th>
<th>t</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>47.778869</td>
<td>9.278442</td>
<td>5.15</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>income</td>
<td>1</td>
<td>-1.025839</td>
<td>0.334006</td>
<td>-3.07</td>
<td>0.0021</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hvalue</td>
<td>1</td>
<td>-0.281636</td>
<td>0.093366</td>
<td>-3.02</td>
<td>0.0026</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_rho</td>
<td>1</td>
<td>0.368144</td>
<td>0.181118</td>
<td>2.03</td>
<td>0.0421</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_lambda</td>
<td>1</td>
<td>0.166525</td>
<td>0.298114</td>
<td>0.56</td>
<td>0.5764</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_sigma2</td>
<td>1</td>
<td>95.597489</td>
<td>19.474422</td>
<td>4.91</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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:

```plaintext
   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
Chapter 11: The CSPATIALREG Procedure

Output 11.1.9  Parameter Estimates of SDAC Model

The CSPATIALREG Procedure

| 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:

```plaintext
   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 | 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:
  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

| 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 CSPATIALREG statement.

The following statements fit a linear model to the CrimeOH data:

```plaintext
/***  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

| 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:

```bash
/*-- SLX --*/
  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

| 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:

```bash
proc means data=mycas.SimData;
  var x1 x2 x3 y;
run;
```
Example 11.2: Simulated Data Example

Output 11.2.1 Summary Statistics

The MEANS Procedure

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>50</td>
<td>-0.0076329</td>
<td>1.0989504</td>
<td>-2.4523193</td>
<td>1.6539456</td>
</tr>
<tr>
<td>x2</td>
<td>50</td>
<td>-0.0829941</td>
<td>0.9671181</td>
<td>-2.5725767</td>
<td>2.4034547</td>
</tr>
<tr>
<td>x3</td>
<td>50</td>
<td>-0.0894387</td>
<td>0.9975304</td>
<td>-2.4470049</td>
<td>2.6720533</td>
</tr>
<tr>
<td>y</td>
<td>50</td>
<td>1.1569199</td>
<td>1.5687060</td>
<td>-1.9399423</td>
<td>4.7136835</td>
</tr>
</tbody>
</table>

The following statements fit a spatial autoregressive (SAR) model to the data by using three regressors, x1, x2, and x3:

```sql
/*-- 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 | 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 x1, x2, and x3, submit the following statements:

```sql
/*-- 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 | DF | Estimate  | Standard Error | t Value | 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 | 0.035316       | -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 | DF | Estimate  | Standard Error | t Value | 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;
```
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 450 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.
Chapter 11: The CSPATIALREG Procedure

Output 11.3.1  SimW_Compact Data Set

<table>
<thead>
<tr>
<th>Obs</th>
<th>SID</th>
<th>csID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L50</td>
<td>L45</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>L32</td>
<td>L35</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>L50</td>
<td>L25</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>L45</td>
<td>L36</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>L10</td>
<td>L8</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>L44</td>
<td>L48</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>L7</td>
<td>L6</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>L36</td>
<td>L15</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>L5</td>
<td>L19</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>L37</td>
<td>L46</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Approx Pr &gt;</th>
<th>t</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>1</td>
<td>1</td>
<td>1.780650</td>
<td>0.098703</td>
<td>18.04</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>1</td>
<td>1</td>
<td>0.573329</td>
<td>0.047395</td>
<td>12.10</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>1</td>
<td>1</td>
<td>0.707048</td>
<td>0.057181</td>
<td>12.37</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>1</td>
<td>1</td>
<td>-0.902843</td>
<td>0.053314</td>
<td>-16.93</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_rho</td>
<td>1</td>
<td>1</td>
<td>-0.473713</td>
<td>0.063008</td>
<td>-7.52</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_sigma2</td>
<td>1</td>
<td>1</td>
<td>0.131509</td>
<td>0.026350</td>
<td>4.99</td>
<td>&lt;.0001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
Example 11.3: Compact Representation of a Spatial Weights Matrix

Output 11.3.3 Parameter Estimates of SEM with Compact Representation

The CSPATIALREG Procedure

| Parameter   | DF | Estimate   | Standard Error | t Value | 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:

```plaintext
/*--- 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   | DF | Estimate   | Standard Error | t Value | 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 |
Chapter 11: The CSPATIALREG Procedure

Output 11.3.5 Parameter Estimates of an SAC Model with Compact Representation

The CSPATIALREG Procedure

| Parameter  | DF  | Estimate  | Standard Error | t Value | Pr > |t| |
|------------|-----|-----------|----------------|---------|-------|---|
| Intercept  | 1   | 1.74514  | 0.108969       | 16.01   | <.0001|
| x1         | 1   | 0.57463  | 0.047481       | 12.10   | <.0001|
| x2         | 1   | 0.69984  | 0.057547       | 12.16   | <.0001|
| x3         | 1   | -0.9094  | 0.053365       | -17.04  | <.0001|
| _rho       | 1   | -0.44974 | 0.075425       | -5.96   | <.0001|
| _lambda    | 1   | -0.23597 | 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 \( |I - \rho W| \) 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 450, 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:

```sas
/*** 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.
Example 11.4: Taylor and Chebyshev Approximations

Output 11.4.1 Parameter Estimates of SAR Model with Chebyshev Approximation

The CSPATIALREG Procedure

| Parameter | DF | Estimate  | Standard Error | t Value | Pr > |t| |
|-----------|----|-----------|----------------|---------|------|---|
| Intercept | 1  | 1.780638  | 0.098699       | 18.04   | <.0001 |
| x1        | 1  | 0.573329  | 0.047395       | 12.10   | <.0001 |
| x2        | 1  | 0.707050  | 0.057181       | 12.37   | <.0001 |
| x3        | 1  | -0.902843 | 0.053314       | -16.93  | <.0001 |
| _rho      | 1  | 0.473704  | 0.063004       | -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:

```bash
/*-- 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 | DF | Estimate  | Standard Error | t Value | Pr > |t| |
|-----------|----|-----------|----------------|---------|------|---|
| Intercept | 1  | 1.780638  | 0.098699       | 18.04   | <.0001 |
| x1        | 1  | 0.573329  | 0.047395       | 12.10   | <.0001 |
| x2        | 1  | 0.707050  | 0.057181       | 12.37   | <.0001 |
| x3        | 1  | -0.902843 | 0.053314       | -16.93  | <.0001 |
| _rho      | 1  | 0.473704  | 0.063004       | -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.

```bash
/*-- 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 | DF  | Estimate | Standard Error | t Value | 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:

```plaintext
/*-- 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 | DF  | Estimate | Standard Error | t Value | 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:

```plaintext
/*-- 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.
To use Chebyshev approximation for this model with compact representation, submit the following statements:

```plaintext
/*-- 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 following statements fit a spatial error model (SEM) by using Chebyshev approximation:

```plaintext
/*-- 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 very similar parameter estimates.

**Output 11.4.7** Parameter Estimates of SEM with Chebyshev Approximation

### The CSPATIALREG Procedure

| Parameter | DF | Estimate   | Standard Error | t Value | Pr > |l| |
|-----------|----|------------|----------------|--------|------|---|
| Intercept | 1  | 1.166102   | 0.029941       | 38.95  | <.0001 |
| x1        | 1  | 0.488665   | 0.049412       | 9.89   | <.0001 |
| x2        | 1  | 0.637142   | 0.061543       | 10.35  | <.0001 |
| x3        | 1  | -0.831674  | 0.055165       | -15.08 | <.0001 |
| _lambda   | 1  | -0.944395  | 0.110388       | -8.56  | <.0001 |
| _sigma2   | 1  | 0.149103   | 0.031221       | 4.78   | <.0001 |

Using the compact representation of the spatial weights matrix, you can submit the following statements to fit an SEM by using Chebyshev approximation:

```plaintext
/*-- 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 | DF | Estimate   | Standard Error | t Value | Pr > |l| |
|-----------|----|------------|----------------|--------|------|---|
| Intercept | 1  | 1.166102   | 0.029941       | 38.95  | <.0001 |
| x1        | 1  | 0.488665   | 0.049412       | 9.89   | <.0001 |
| x2        | 1  | 0.637142   | 0.061543       | 10.35  | <.0001 |
| x3        | 1  | -0.831674  | 0.055165       | -15.08 | <.0001 |
| _lambda   | 1  | -0.944395  | 0.110388       | -8.56  | <.0001 |
| _sigma2   | 1  | 0.149103   | 0.031221       | 4.78   | <.0001 |

The following statements fit a spatial Durbin error model (SDEM) by using Taylor approximation:

```plaintext
/*-- 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 | DF | Estimate | Standard Error | t Value | 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:

```plaintext
/*--- 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 | DF | Estimate | Standard Error | t Value | 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:
Chapter 11: The CSPATIALREG Procedure

/*-- 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 | DF | Estimate | Standard Error | t Value | Approx Pr > |t| |
|-----------|----|----------|---------------|--------|-------------|---|
| Intercept | 1  | 1.193199 | 0.047156      | 25.30  | <.0001      |
| x1        | 1  | 0.566631 | 0.056730      | 9.99   | <.0001      |
| x2        | 1  | 0.722163 | 0.059688      | 12.10  | <.0001      |
| x3        | 1  | -0.909288| 0.061252      | -14.84 | <.0001      |
| W_x1      | 1  | -0.135591| 0.128237      | -1.06  | 0.2904      |
| W_x2      | 1  | -0.418437| 0.152818      | -2.74  | 0.0062      |
| W_x3      | 1  | 0.290294 | 0.192253      | 1.51   | 0.1311      |
| _lambda   | 1  | -0.722074| 0.208152      | -3.47  | 0.0005      |
| _sigma2   | 1  | 0.126797 | 0.026816      | 4.73   | <.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=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 | DF | Estimate | Standard Error | t Value | Approx Pr > |t| |
|-----------|----|----------|----------------|---------|-------------|---|
| Intercept | 1  | 1.193199 | 0.047156       | 25.30   | <.0001      |
| x1        | 1  | 0.566631 | 0.056730       | 9.99    | <.0001      |
| x2        | 1  | 0.722163 | 0.059688       | 12.10   | <.0001      |
| x3        | 1  | -0.909288| 0.061252       | -14.84  | <.0001      |
| W_x1      | 1  | -0.135591| 0.128237       | -1.06   | 0.2904      |
| W_x2      | 1  | -0.418437| 0.152818       | -2.74   | 0.0062      |
| W_x3      | 1  | 0.290294 | 0.192253       | 1.51    | 0.1311      |
| _lambda   | 1  | -0.722074| 0.208152       | -3.47   | 0.0005      |
| _sigma2   | 1  | 0.126797 | 0.026816       | 4.73    | <.0001      |

References


Chapter 12
The ECM Procedure

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-wise 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:
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:

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

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:

```sas
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 CCDM 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 467, but you can substitute any appropriately named CAS engine libref.

```sas
/* 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):

```sas
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 reproducible. 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 $\alpha$ is defined as the smallest number $l$ such that the probability that the loss exceeds $l$ is no larger than $1 - \alpha$.

Formally,

$$\text{VaR}_\alpha(L) = \inf\{l \in \mathbb{R} : \Pr[L > l] \leq 1 - \alpha\}$$

$$= \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\}$$

$$= F^{-1}_L(\alpha)$$

where $F_L$ and $F^{-1}_L$ 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**

<table>
<thead>
<tr>
<th>Summary Statistics</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>500000</td>
</tr>
<tr>
<td>Minimum</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>161698.7</td>
</tr>
<tr>
<td>Mean</td>
<td>14653.4</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>11330.6</td>
</tr>
<tr>
<td>Variance</td>
<td>128383524</td>
</tr>
<tr>
<td>Skewness</td>
<td>1.39642</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>3.33577</td>
</tr>
<tr>
<td>Median</td>
<td>12307.6</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>14061.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>972.3825</td>
</tr>
<tr>
<td>10</td>
<td>2549.5</td>
</tr>
<tr>
<td>25</td>
<td>6310.9</td>
</tr>
<tr>
<td>50</td>
<td>12307.6</td>
</tr>
<tr>
<td>75</td>
<td>20372.2</td>
</tr>
<tr>
<td>90</td>
<td>29598.5</td>
</tr>
<tr>
<td>95</td>
<td>36244.7</td>
</tr>
<tr>
<td>97.5</td>
<td>42721.0</td>
</tr>
<tr>
<td>98.5</td>
<td>47460.3</td>
</tr>
<tr>
<td>99</td>
<td>51283.2</td>
</tr>
<tr>
<td>99.5</td>
<td>57915.0</td>
</tr>
</tbody>
</table>

**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 > VaR column indicates the probability that a loss exceeds the VaR value. For a level $\alpha$, the TVaR $\alpha$ is defined as the expected value of the loss given that the loss exceeds VaR $\alpha$. Formally, 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

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Pr &gt; VaR</th>
<th>Value at Risk (VaR)</th>
<th>Tail Value at Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>0.1</td>
<td>29598.5</td>
<td>39108.7</td>
</tr>
<tr>
<td>95</td>
<td>0.05</td>
<td>36244.7</td>
<td>45682.5</td>
</tr>
<tr>
<td>97.5</td>
<td>0.025</td>
<td>42721.0</td>
<td>52247.9</td>
</tr>
<tr>
<td>98.5</td>
<td>0.015</td>
<td>47460.3</td>
<td>57151.1</td>
</tr>
<tr>
<td>99.5</td>
<td>0.005</td>
<td>57915.0</td>
<td>67992.2</td>
</tr>
</tbody>
</table>

### Syntax: ECM Procedure

The following statements are available in the ECM procedure:

```plaintext
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

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the information about a marginal variable</td>
<td>MARGINAL</td>
<td></td>
</tr>
<tr>
<td>Specifies the subset of marginal variables to analyze</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>Specifies where and how to write the full total loss sample</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>Specifies where and how to write the summary statistics of the total loss sample</td>
<td>OUTSUM</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display (an alternative to the PRINT= option)</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td><strong>Data Table Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data table that contains the copula simulation</td>
<td>PROC ECM</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the input data table that contains a marginal sample</td>
<td>MARGINAL</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the output data table for the total loss sample</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies the output data table for the summary statistics</td>
<td>OUTSUM</td>
<td>OUT=</td>
</tr>
<tr>
<td><strong>Marginal Input Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the draw (sample) identifier for all marginals</td>
<td>PROC ECM</td>
<td>DRAWID=</td>
</tr>
<tr>
<td>Specifies the draw (sample) identifier for a specific marginal variable</td>
<td>MARGINAL</td>
<td>DRAWID=</td>
</tr>
<tr>
<td>Specifies the variable that contains the marginal variable sample</td>
<td>MARGINAL</td>
<td>SAMPLEVAR=</td>
</tr>
<tr>
<td><strong>Empirical Distribution Function (EDF)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Estimation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the EDF accuracy for the body region of the distribution</td>
<td>PROC ECM</td>
<td>EDFACCURACY=</td>
</tr>
<tr>
<td>Computes exact counts for the final set of bins</td>
<td>PROC ECM</td>
<td>EXACTFINALCOUNT=</td>
</tr>
<tr>
<td>Specifies the maximum number of iterations for the EDF estimation algorithm</td>
<td>PROC ECM</td>
<td>MAXITER=</td>
</tr>
<tr>
<td>Suppresses the shuffling of marginal data</td>
<td>PROC ECM</td>
<td>NOSHUFFLE=</td>
</tr>
<tr>
<td>Specifies a sampling fraction for the body region of the distribution</td>
<td>PROC ECM</td>
<td>SAMPLEFRACTION=</td>
</tr>
<tr>
<td>Specifies a seed for the internal pseudorandom number generator</td>
<td>PROC ECM</td>
<td>SEED=</td>
</tr>
</tbody>
</table>
Table 12.1  continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the EDF accuracy for the tail region of the distribution</td>
<td>PROC ECM</td>
<td>TAILEDFACCURACY=</td>
</tr>
<tr>
<td>Specifies the EDF value where the tail region of the distribution begins</td>
<td>PROC ECM</td>
<td>TAILSTARTEDF=</td>
</tr>
<tr>
<td>Specifies the tolerance for the EDF estimation algorithm</td>
<td>PROC ECM</td>
<td>TOLERANCE=</td>
</tr>
<tr>
<td><strong>Output Preparation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writes the EDF and percentile estimates of marginals to the output data table</td>
<td>OUTPUT</td>
<td>MARGINALSOUT</td>
</tr>
<tr>
<td>Specifies the decimal precision to use in forming percentile variable names</td>
<td>OUTSUM</td>
<td>PCTLNDEC=</td>
</tr>
<tr>
<td>Specifies the prefix for the percentile variable names</td>
<td>OUTSUM</td>
<td>PCTLPRE=</td>
</tr>
<tr>
<td>Specifies the percentiles to compute and report</td>
<td>OUTSUM</td>
<td>PCTLPPTS=</td>
</tr>
<tr>
<td>Specifies the method for computing the percentiles</td>
<td>PROC ECM</td>
<td>PCTLLEDEF=</td>
</tr>
<tr>
<td>Specifies the method for computing the percentiles of a specific marginal variable</td>
<td>MARGINAL</td>
<td>PCTLLEDEF=</td>
</tr>
<tr>
<td>Specifies the variable name for the total loss</td>
<td>OUTPUT</td>
<td>TOTALLOSSVAR=</td>
</tr>
<tr>
<td>Specifies the prefix for the TVaR variable names</td>
<td>OUTSUM</td>
<td>TVARPRE=</td>
</tr>
<tr>
<td>Specifies percentile values for which to compute and report the TVaR</td>
<td>OUTSUM</td>
<td>TVARPTS=</td>
</tr>
<tr>
<td>Specifies the denominator for computing second- and higher-order moments</td>
<td>PROC ECM</td>
<td>VARDEF=</td>
</tr>
<tr>
<td><strong>Displayed Output Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suppresses all displayed output</td>
<td>PROC ECM</td>
<td>NOPRINT</td>
</tr>
<tr>
<td>Specifies which displayed output to produce</td>
<td>PROC ECM</td>
<td>PRINT=</td>
</tr>
</tbody>
</table>

**PROC ECM Statement**

```plaintext
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 467.

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 TAILEDFACCURACY= 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 488. 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 490.

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.

**TAILEDACCURACY=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, TAIREDACCURACY=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 $\epsilon$ 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 $\delta$ 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 490. 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 subsetted 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 490. 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 467. 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 488.

By default, PCTLDEF=5.

**OUTPUT Statement**

```plaintext
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 467. 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 489.
OUTSUM Statement

\[ \text{OUTSUM} \ < \ \text{outsum-options} \ > \ < \ \text{statistic-keyword} \ \leq \text{variable-name} \ > \ldots \ \text{statistic-keyword} \ \leq \text{variable-name} \ > \ ; \]

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 \textit{outsum-options}:

\textbf{OUT} = \textit{CAS-libref.data-table}

\textbf{OUTSUM} = \textit{CAS-libref.data-table}

specifies the output data table that contains the summary statistics of the total loss sample. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section “Using CAS Sessions and CAS Engine Librefs” on page 467. The \textit{CAS-libref} must be identical to the \textit{CAS-libref} that you specify in the \texttt{DATA=} option.

You can control the statistics that appear in this data table by specifying the \texttt{PCTLPTS=} option, the \texttt{TVARPTS=} option, and different \textit{statistic-keyword} values.

\textbf{PCTLNDEC} = \textit{integer-value}

specifies the maximum number of decimal places to use when creating the names of the variables for the percentile values in the \texttt{PCTLPTS=} or \texttt{TVARPTS=} options. For example, for a percentile value of 99.9995, if you specify \texttt{PCTLNDEC=4}, then the suffix of the variable is \texttt{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 \texttt{PCTLPTS=} option or the \texttt{TVARPTS=} option.

\textbf{PCTLPRE} = \textit{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 \textit{prefix} \textit{t}, where \( t \) is the text representation of \( p \) that is formed by replacing the decimal point with an underscore (_). For example, if \texttt{PCTLPRE=P}, then the estimate of the 97.5th percentile is written to a variable whose name is \texttt{P97_5}. By default, \texttt{PCTLPRE=P}.

\textbf{PCTLPTS} = \textit{percentile-option}

specifies the percentiles of the total loss that you want PROC ECM to compute. The \textit{percentile-option} can have one of the following two forms:

\textbf{DEFAULT}

\textbf{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:

\[
\text{pctlpts}=10, 20, 99.6, 99.7, 99.8, 99.9 \\
\text{pctlpts}=10, 20, 99.6 \text{ to } 99.9 \text{ by } 0.1
\]

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 prefix\(t\), 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:

\[
\text{tvarpts}=90, 95, 96.25, 97.5, 98.75, 99.5 \\
\text{tvarpts}=90, 95 \text{ to } 99 \text{ by } 1.25, 99.5
\]

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:

\[
\text{statistic-keyword}=\text{variable-name}
\]
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:

\[
\text{KURTOSIS | KURT} \\
\text{MEAN} \\
\text{P01}
\]
writes the kurtosis of the total loss sample.
writes the mean of the total loss sample.
writes the first percentile of the total loss sample.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P05</td>
<td>writes the fifth percentile of the total loss sample.</td>
</tr>
<tr>
<td>P10</td>
<td>writes the 10th percentile of the total loss sample.</td>
</tr>
<tr>
<td>P25</td>
<td>Q1</td>
</tr>
<tr>
<td>P50</td>
<td>MEDIAN</td>
</tr>
<tr>
<td>P75</td>
<td>Q3</td>
</tr>
<tr>
<td>P90</td>
<td>writes the 90th percentile of the total loss sample.</td>
</tr>
<tr>
<td>P95</td>
<td>writes the 95th percentile of the total loss sample.</td>
</tr>
<tr>
<td>P99</td>
<td>writes the 99th percentile of the total loss sample.</td>
</tr>
<tr>
<td>P99_5</td>
<td>P995</td>
</tr>
<tr>
<td>Q RANGE</td>
<td>writes the interquartile range (Q3–Q1) of the total loss sample.</td>
</tr>
<tr>
<td>SKEWNESS</td>
<td>SKEW</td>
</tr>
<tr>
<td>STDDEV</td>
<td>STD</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>writes the variance of the total loss sample.</td>
</tr>
</tbody>
</table>

**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 488 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 100\( p_t \)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 accuracies for the body and tail regions of the distribution by using the EDFACCURACY= and TAILEDFACCURACY= 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, \( \epsilon (0 < \epsilon < 1) \), 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 \( \delta \) is the EDF accuracy of the body or tail region that is being analyzed. You can specify \( \epsilon \) 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 reproducible results across multiple runs of PROC ECM for identical inputs and identical grid configuration.

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 \( b_i = x_{\text{max}} \) for \( i > B \), where \( x_{\text{max}} \) 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>
<thead>
<tr>
<th>PCTLDEF=</th>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Weighted average</td>
<td>( y = (1 - g)b_i + gb_{i+1} )</td>
</tr>
</tbody>
</table>
| 2        | Value closest to \( Np \) | \[
y = b_i \quad \text{if } g < \frac{1}{2} \\
y = b_i \quad \text{if } g = \frac{1}{2} \text{ and } j \text{ is even} \\
y = b_{i+1} \quad \text{if } g = \frac{1}{2} \text{ and } j \text{ is odd} \\
y = b_{i+1} \quad \text{if } g > \frac{1}{2}
\]|
| 3        | Empirical distribution function | \[
y = b_i \quad \text{if } g = 0 \\
y = b_{i+1} \quad \text{if } g > 0
\]|
| 4        | Weighted average (identical to PCTLDEF=1) | \( y = (1 - g)b_i + gb_{i+1} \) |
| 5        | Empirical distribution function with averaging | \[
y = \frac{1}{2}(b_i + b_{i+1}) \quad \text{if } g = 0 \\
y = b_{i+1} \quad \text{if } g > 0
\]|

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

<table>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSummary</td>
<td>Summary of the marginal or total loss sample</td>
<td>PRINT=DATASUMMARY</td>
</tr>
<tr>
<td>EDFInit</td>
<td>Initialization summary of the EDF estimation process</td>
<td>PRINT=EDFINIT</td>
</tr>
<tr>
<td>EDFInitOptDetails</td>
<td>Optimization details of the initialization of the EDF estimation process</td>
<td>PRINT=EDFINIT</td>
</tr>
<tr>
<td>EDFOptDetails</td>
<td>Optimization details of the EDF estimation process</td>
<td>PRINT=EDFOPTDETAILS</td>
</tr>
<tr>
<td>EDFS Summary</td>
<td>Summary of the EDF estimation process</td>
<td>PRINT=EDFSUMMARY</td>
</tr>
<tr>
<td>Percentiles</td>
<td>Percentiles of the total loss sample</td>
<td>PRINT=PERCENTILES in the PROC ECM statement and PCTLPTS= in the OUTSUM statement</td>
</tr>
<tr>
<td>SummaryStatistics</td>
<td>Summary statistics of the total loss sample</td>
<td>PRINT=SUMMARYSTATISTICS</td>
</tr>
<tr>
<td>TVaR</td>
<td>Tail value-at-risk (TVaR) estimates of the total loss sample</td>
<td>PRINT=TVAR in the PROC ECM statement and TVARPTS= in the OUTSUM statement</td>
</tr>
<tr>
<td>Timing</td>
<td>Timing information for various computational stages of the procedure</td>
<td>PRINT=ALL</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Detailed Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSummary</td>
<td>The “Data Summary” table contains the summary of each sample for which PROC ECM estimates the EDF.</td>
</tr>
<tr>
<td>EDFInit</td>
<td>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.</td>
</tr>
<tr>
<td>EDFInitOptDetails</td>
<td>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.</td>
</tr>
<tr>
<td>ODS Table Name</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EDFOptDetails</td>
<td>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.</td>
</tr>
<tr>
<td>EDFSummary</td>
<td>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.</td>
</tr>
<tr>
<td>Percentiles</td>
<td>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.</td>
</tr>
<tr>
<td>SummaryStatistics</td>
<td>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.</td>
</tr>
<tr>
<td>TVaR</td>
<td>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 ).</td>
</tr>
</tbody>
</table>
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 465 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:

```sas
/*** 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);
umloss = 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 corp1rb3;
array sbetaC(7) _TEMPORARY_ (5 1 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 */
corporb3 = 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 467, 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:

```sas
/*** 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:

```sas
/*** 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 ***/
```
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:

```sas
/*** 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, both the Poisson and negative binomial models have identical SBC values. This usually implies that the negative binomial model is equivalent to the Poisson model, because the overdispersion parameter $\alpha$ is close to 0. Formally, the conditional variance of the negative binomial distribution is $V(y|x) = \mu + \alpha \mu^2$, where $\mu$ denotes the mean. As $\alpha$ 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

<table>
<thead>
<tr>
<th>line</th>
<th>Model</th>
<th>Property</th>
<th>PropertyValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommercialBanking</td>
<td>Poisson</td>
<td>SBC</td>
<td>4010.42</td>
</tr>
<tr>
<td>CommercialBanking</td>
<td>NegBin(p=2)</td>
<td>SBC</td>
<td>3608.513</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>Poisson</td>
<td>SBC</td>
<td>2940.835</td>
</tr>
<tr>
<td>RetailBanking</td>
<td>NegBin(p=2)</td>
<td>SBC</td>
<td>2940.835</td>
</tr>
</tbody>
</table>
Example 12.1: Developing an Economic Capital Model

Output 12.1.2 Parameter Estimates of Negative Binomial Count Model (Retail Banking)

The CNTSELECT Procedure

Selected Model

| Parameter | DF | Estimate | Std Error | t Value | Approx Pr > |t| |
|-----------|----|----------|-----------|---------|-------------|---|
| Intercept | 1  | 0.124745 | 0.032401  | 3.85    | 0.0001      |   |
| corpKRI1  | 1  | 0.989717 | 0.019687  | 50.27   | <.0001      |   |
| corpKRI2  | 1  | -0.504276| 0.019626  | -25.69  | <.0001      |   |
| rbKRI1    | 1  | 0.522673 | 0.019662  | 26.32   | <.0001      |   |
| rbKRI2    | 1  | 0.712892 | 0.020307  | 35.11   | <.0001      |   |
| _Alpha    | 0  | 1.053671E-8 |         | .       | .           |   |
| Restrict 1 | -1 | 1356.711901 |         | .       | .           |   |

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 14, “The SEVSELECT Procedure.”

The following PROC SEVSELECT step fits severity models for three candidate distributions—gamma, lognormal, and Weibull—to the example data:

```sas
/*** Fit severity models ***/
proc sevselect data=mycas.OpRiskLosses store=mycas.opriskSevstore(promote=yes) covout print=(allfitstats);
   by line;
   loss lossValue;
   dist 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

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>46964</td>
<td>46974</td>
<td>46974</td>
<td>47004</td>
<td>3.17537</td>
<td>64.32374</td>
<td>3.95517</td>
</tr>
<tr>
<td>Logn</td>
<td>46471*</td>
<td>46481*</td>
<td>46481*</td>
<td>46512*</td>
<td>3.18322</td>
<td>52.02956</td>
<td>4.16386</td>
</tr>
<tr>
<td>Weibull</td>
<td>46986</td>
<td>46996</td>
<td>46996</td>
<td>47027</td>
<td>2.94192*</td>
<td>47.65430*</td>
<td>2.36234*</td>
</tr>
</tbody>
</table>

* Asterisk (*) denotes the best model in the column.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma</td>
<td>30851*</td>
<td>30865*</td>
<td>30865*</td>
<td>30908*</td>
<td>7.27617</td>
<td>453.85603</td>
<td>21.00515</td>
</tr>
<tr>
<td>Logn</td>
<td>30985</td>
<td>30999</td>
<td>30999</td>
<td>31041</td>
<td>6.66834*</td>
<td>346.69875*</td>
<td>20.11890</td>
</tr>
<tr>
<td>Weibull</td>
<td>30979</td>
<td>30993</td>
<td>30993</td>
<td>31036</td>
<td>7.38807</td>
<td>564.92046</td>
<td>18.82058*</td>
</tr>
</tbody>
</table>

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

```sas
/*** 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 51,320 ± 3,315 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

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Estimate</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5321.5</td>
<td>418.68166</td>
</tr>
<tr>
<td>5</td>
<td>8028.0</td>
<td>552.36943</td>
</tr>
<tr>
<td>25</td>
<td>13402.7</td>
<td>829.85931</td>
</tr>
<tr>
<td>50</td>
<td>18447.3</td>
<td>1116.7</td>
</tr>
<tr>
<td>75</td>
<td>24774.2</td>
<td>1487.9</td>
</tr>
<tr>
<td>95</td>
<td>36409.2</td>
<td>2279.8</td>
</tr>
<tr>
<td>99</td>
<td>46843.7</td>
<td>2932.4</td>
</tr>
<tr>
<td>99.5</td>
<td>51320.2</td>
<td>3314.7</td>
</tr>
</tbody>
</table>

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 CCDM, 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:

```sas
/*** 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

<table>
<thead>
<tr>
<th>Copula</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gumbel</td>
<td>-69.6587</td>
</tr>
<tr>
<td>T</td>
<td>-61.4626</td>
</tr>
<tr>
<td>Frank</td>
<td>-48.3310</td>
</tr>
<tr>
<td>Clayton</td>
<td>-25.3843</td>
</tr>
</tbody>
</table>

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:

```plaintext
/*** 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.
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 485.

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:

```sas
/* 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 EDFACCURACY= 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 reproducibility of the results when you rerun 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

<table>
<thead>
<tr>
<th>Data Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Table</td>
</tr>
<tr>
<td>Severity Model</td>
</tr>
<tr>
<td>Draw Identifier</td>
</tr>
<tr>
<td>Observations</td>
</tr>
<tr>
<td>Valid Observations</td>
</tr>
<tr>
<td>Zero-valued Observations</td>
</tr>
<tr>
<td>Minimum</td>
</tr>
<tr>
<td>Maximum</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>EDF Binning Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Bins</td>
</tr>
<tr>
<td>Convergence Status</td>
</tr>
<tr>
<td>Tail Start EDF</td>
</tr>
<tr>
<td>Tail Start</td>
</tr>
<tr>
<td>Bins Distributed Among Workers</td>
</tr>
</tbody>
</table>

**EDF Binning Initialization Details**

<table>
<thead>
<tr>
<th>Target Proportion</th>
<th>Iterations</th>
<th>Criterion DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.142857</td>
<td>4</td>
<td>0.0016000</td>
<td>6</td>
<td>0.1034</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>EDF Binning Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

**EDF Binning Optimization Details**

<table>
<thead>
<tr>
<th>Target Proportion</th>
<th>Iterations</th>
<th>Criterion DF</th>
<th>Chi-Square</th>
<th>Pr &gt; ChiSq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.023256</td>
<td>3</td>
<td>0.010067</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>0.023256</td>
<td>1</td>
<td>0.0071066</td>
<td>42</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Summary Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Size</td>
<td>10000</td>
</tr>
<tr>
<td>Minimum</td>
<td>5985.2</td>
</tr>
<tr>
<td>Maximum</td>
<td>82785.3</td>
</tr>
<tr>
<td>Mean</td>
<td>23330.9</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>9697.7</td>
</tr>
<tr>
<td>Variance</td>
<td>94045774.9</td>
</tr>
<tr>
<td>Skewness</td>
<td>1.03636</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>1.72531</td>
</tr>
<tr>
<td>Median</td>
<td>22042.0</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>12260.3</td>
</tr>
</tbody>
</table>
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:

```sas
/*** 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;
%if &draw1=0 and &draw2=0 %then %do;
    data allecmstats;
    set mycas.tlSumm;
  %end;
%mend EstimateECMvariability;
```
Example 12.1: Developing an Economic Capital Model

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 $46,294 \pm 1,815$ units if you use the mean and standard deviation as measures of location and dispersion. The corresponding TVaR estimate is about $53,053 \pm 1,995$ units. If you use the median and interquartile range as measures of location and dispersion, then the VaR estimate is about $46,497 \pm 2,152$ units and the TVaR estimate is about $53,371 \pm 2,699$ units.

Output 12.1.11 Location and Dispersion Estimates of Various ECM Summary Statistics and Risk Measures

<table>
<thead>
<tr>
<th>Variable</th>
<th>Label</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Median</th>
<th>Quartile Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEAN</td>
<td>Mean</td>
<td>100</td>
<td>22952.73</td>
<td>868.5844723</td>
<td>23031.14</td>
<td>1031.02</td>
</tr>
<tr>
<td>Median</td>
<td>50th Percentile</td>
<td>100</td>
<td>21551.33</td>
<td>824.3308288</td>
<td>21612.51</td>
<td>1036.47</td>
</tr>
<tr>
<td>Var95</td>
<td>95th Percentile</td>
<td>100</td>
<td>41004.97</td>
<td>1616.80</td>
<td>41168.26</td>
<td>2010.28</td>
</tr>
<tr>
<td>TVaR95</td>
<td>Tail Value-at-Risk (TVaR): 95th Percentile</td>
<td>100</td>
<td>48259.52</td>
<td>1842.27</td>
<td>48511.46</td>
<td>2374.48</td>
</tr>
<tr>
<td>Var97_5</td>
<td>97.5th Percentile</td>
<td>100</td>
<td>46294.49</td>
<td>1815.43</td>
<td>46497.09</td>
<td>2151.79</td>
</tr>
<tr>
<td>TVaR97_5</td>
<td>Tail Value-at-Risk (TVaR): 97.5th Percentile</td>
<td>100</td>
<td>53053.46</td>
<td>1994.83</td>
<td>53370.87</td>
<td>2698.52</td>
</tr>
<tr>
<td>Var99_5</td>
<td>99.5th Percentile</td>
<td>100</td>
<td>56854.96</td>
<td>2096.26</td>
<td>57415.00</td>
<td>3129.65</td>
</tr>
<tr>
<td>TVaR99_5</td>
<td>Tail Value-at-Risk (TVaR): 99.5th Percentile</td>
<td>100</td>
<td>62962.23</td>
<td>2284.90</td>
<td>63514.55</td>
<td>3727.21</td>
</tr>
</tbody>
</table>

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


Chapter 13
The HMM Procedure

Contents

Overview: HMM Procedure .................................................. 512
   Using CAS Sessions and CAS Engine Librefs .................. 513
Getting Started: HMM Procedure ...................................... 514
   Gaussian Hidden Markov Model ................................. 514
   Gaussian Hidden Markov Model for Cross-Sectional Time Series Data .................. 526
   Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional
      Time Series Data .................................................. 529
   Regime-Switching Regression Model ............................ 549
   Regime-Switching Autoregression Model ....................... 555
Syntax: HMM Procedure .................................................. 560
   Functional Summary .............................................. 560
   PROC HMM Statement ............................................. 562
   DECODE Statement ............................................... 563
   DISPLAY Statement .............................................. 563
   DISPLAYOUT Statement ......................................... 564
   ESTIMATE Statement ............................................. 565
   EVALUATE Statement ............................................. 565
   FILTER Statement ................................................ 566
   FORECAST Statement ........................................... 566
   ID Statement .................................................... 567
   INITIAL Statement ............................................... 567
   MODEL Statement ................................................. 568
   OPTIMIZE Statement ............................................. 570
   PRIOR Statement ................................................ 572
   SCORE Statement ................................................ 574
   SMOOTH Statement .............................................. 574
Details: HMM Procedure ................................................ 575
   Hidden Markov Model .......................................... 575
   Gaussian Hidden Markov Model ................................. 579
   Gaussian Mixture Hidden Markov Model ....................... 579
   Regime-Switching Regression Model ............................ 580
   Regime-Switching Autoregression Model ....................... 581
   Parameter Estimation Methods ................................ 583
   Optimization Algorithms ....................................... 588
   Matrix Expression ............................................... 588
   Information Criteria ............................................ 598
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).

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
- two types of optimization algorithms: the active set algorithm and the interior point algorithm
- multistart mode for optimization; 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.
- initial values for optimization
- the prior distribution for parameters when you use the MAP method
- 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
- 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, median, and confidence interval of 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

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:

```sas
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:

```sas
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:

```sas
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.”
Gaussian Hidden Markov Model

Let \( Y_t = (y_{1t}, \ldots, y_{pt})' \), \( t = 1, \ldots, T \), denote a \( p \)-dimensional time series vector of random variables. Conditional on a latent variable \( S_t \), \( Y_t \) follows a normal distribution,

\[
Y_t | S_t \sim N(\mu_{S_t}, \Sigma_{S_t})
\]

where \( \mu_{S_t} \) and \( \Sigma_{S_t} \) are mean and covariance parameters whose values depend on the variable \( S_t \). The variable \( S_t \), the so-called state, follows the first-order Markov chain; that is,

\[
p(S_t | S_{t-1}, S_{t-2}, \ldots, S_1) = p(S_t | S_{t-1})
\]

where \( p(\cdot | \cdot) \) denotes the conditional probability. The range of \( S_t \) is a finite set, \( \{1, \ldots, K\} \). The transition probability from state \( i \) to state \( j \) is expressed as

\[
a_{ij} = p(S_t = j | S_{t-1} = i)
\]

The \( K \times K \) matrix \( A = \{a_{ij}\} \) is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), \( \pi \), of the first state \( S_1 \):

\[
\pi = \{\pi_i = p(S_1 = i), i = 1, \ldots, K\}
\]

Because the variable \( S_t \) is unobservable and follows a Markov chain and because \( Y_t \) follows a Gaussian distribution conditional on \( S_t \), the model is called the Gaussian hidden Markov model (Gaussian HMM). The Gaussian HMM is sometimes described as \( \{\pi, A, B\} \), where \( B = \{\mu_i, \Sigma_i, i = 1, \ldots, K\} \), consisting of parameters that define the state-dependent distribution of observable variables.

Consider a bivariate Gaussian HMM that has the following parameter values:

\[
\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \ A = \begin{pmatrix} 0.001 & 0.999 \\ 0.999 & 0.001 \end{pmatrix}
\]

\[
\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \ \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mu_2 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \ \Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\]
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;
```

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 $x$:

```sas
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 \( y \):

```plaintext
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.

Ignoring the sequential information, the following statements plot the observed points \((x, y)\):

```plaintext
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:

```sas
/* 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;
```

\(^1\)In 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.
data clusterCheck;
  merge DGPone(in=a) Cluster(in=b);
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

<table>
<thead>
<tr>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9202</td>
</tr>
</tbody>
</table>

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.

proc hmm data=mycas.One;
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

<table>
<thead>
<tr>
<th>Observation Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Missing Observations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Model</td>
</tr>
<tr>
<td>Stationary</td>
</tr>
<tr>
<td>Number of States</td>
</tr>
<tr>
<td>Number of Dependent Variables</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM1_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM1_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM2_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM2_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>MU1_1</td>
<td>0.642059</td>
</tr>
<tr>
<td>MU1_2</td>
<td>0.513308</td>
</tr>
<tr>
<td>MU2_1</td>
<td>-1.609545</td>
</tr>
<tr>
<td>MU2_2</td>
<td>-1.329369</td>
</tr>
<tr>
<td>SIGMA1_1_1</td>
<td>1.152038</td>
</tr>
<tr>
<td>SIGMA1_2_1</td>
<td>0.320856</td>
</tr>
<tr>
<td>SIGMA1_2_2</td>
<td>1.478814</td>
</tr>
<tr>
<td>SIGMA2_1_1</td>
<td>0.525065</td>
</tr>
<tr>
<td>SIGMA2_2_1</td>
<td>-0.243910</td>
</tr>
<tr>
<td>SIGMA2_2_2</td>
<td>0.892018</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Algorithm Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Number of Iterations</td>
<td>128</td>
</tr>
<tr>
<td>Tolerance for Optimality Error</td>
<td>1E-6</td>
</tr>
<tr>
<td>Tolerance for Infeasibility</td>
<td>1E-6</td>
</tr>
<tr>
<td>Maximum Allowed Time (seconds)</td>
<td>1.797693E308</td>
</tr>
<tr>
<td>Maximum Magnitude of Objective Function Value</td>
<td>1E20</td>
</tr>
<tr>
<td>Random Seed</td>
<td>1</td>
</tr>
<tr>
<td>Solution Type</td>
<td>1</td>
</tr>
<tr>
<td>Multi-Start Globalization Scheme</td>
<td>0</td>
</tr>
<tr>
<td>Multi-Start Globalization Bound Range</td>
<td>2</td>
</tr>
</tbody>
</table>

The iterations of the optimization process are shown in Figure 13.8.

**Figure 13.8** Iterations of the Optimization Process

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective Function</th>
<th>Optimality Error</th>
<th>Infeasibility</th>
<th>Function Calls</th>
<th>Gradient Calls</th>
<th>Hessian Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-34510.54106</td>
<td>610.24392</td>
<td>0.00000</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-29343.35393</td>
<td>350.54614</td>
<td>0.00000</td>
<td>20</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-28606.45004</td>
<td>78.05203</td>
<td>0.00000</td>
<td>36</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>-28467.87680</td>
<td>1.42097</td>
<td>0.00000</td>
<td>54</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>-28460.41282</td>
<td>0.34998</td>
<td>0.00000</td>
<td>72</td>
<td>36</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>-28459.08728</td>
<td>0.10030</td>
<td>0.00000</td>
<td>90</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>-28458.80039</td>
<td>0.02968</td>
<td>0.00000</td>
<td>108</td>
<td>54</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>-28458.73576</td>
<td>0.00812</td>
<td>0.00000</td>
<td>126</td>
<td>63</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>-28458.72552</td>
<td>0.00167</td>
<td>0.00000</td>
<td>144</td>
<td>72</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>-28458.72484</td>
<td>0.00018</td>
<td>0.00000</td>
<td>162</td>
<td>81</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>-28458.72483</td>
<td>0.00000</td>
<td>0.00000</td>
<td>180</td>
<td>90</td>
<td>10</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Optimization Results</th>
<th>Parameter</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM1_1</td>
<td>-0.002071</td>
<td></td>
</tr>
<tr>
<td>TPM1_2</td>
<td>0.997929</td>
<td></td>
</tr>
<tr>
<td>TPM2_1</td>
<td>0.999863</td>
<td></td>
</tr>
<tr>
<td>TPM2_2</td>
<td>0.000137</td>
<td></td>
</tr>
<tr>
<td>MU1_1</td>
<td>1.006466</td>
<td></td>
</tr>
<tr>
<td>MU1_2</td>
<td>0.994184</td>
<td></td>
</tr>
<tr>
<td>MU2_1</td>
<td>-0.993036</td>
<td></td>
</tr>
<tr>
<td>MU2_2</td>
<td>-1.007869</td>
<td></td>
</tr>
<tr>
<td>SIGMA1_1_1</td>
<td>1.011152</td>
<td></td>
</tr>
<tr>
<td>SIGMA1_2_1</td>
<td>0.006387</td>
<td></td>
</tr>
<tr>
<td>SIGMA1_2_2</td>
<td>0.988329</td>
<td></td>
</tr>
<tr>
<td>SIGMA2_1_1</td>
<td>0.992333</td>
<td></td>
</tr>
<tr>
<td>SIGMA2_2_1</td>
<td>-0.005378</td>
<td></td>
</tr>
<tr>
<td>SIGMA2_2_2</td>
<td>1.008722</td>
<td></td>
</tr>
</tbody>
</table>

---

**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

<table>
<thead>
<tr>
<th>Initial State Probabilities</th>
<th>State Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

The estimates of the transition probability matrix (TPM) are shown in Figure 13.11.

**Figure 13.11** Estimates of Transition Probability Matrix

<table>
<thead>
<tr>
<th>Transition Probabilities</th>
<th>State</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0.00207</td>
<td>0.99793</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.99986</td>
<td>0.00014</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>State</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00647</td>
<td>0.99418</td>
</tr>
<tr>
<td>2</td>
<td>-0.99304</td>
<td>-1.00787</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>State</th>
<th>Variable</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>1.01115</td>
<td>0.00639</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>0.00639</td>
<td>0.98833</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>0.99233</td>
<td>-0.00538</td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>-0.00538</td>
<td>1.00872</td>
</tr>
</tbody>
</table>

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                     | 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](image)

<table>
<thead>
<tr>
<th>Fit Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Log Likelihood</strong></td>
<td>-28458.72483</td>
</tr>
<tr>
<td><strong>AIC</strong></td>
<td>56941.449663</td>
</tr>
<tr>
<td><strong>AICC</strong></td>
<td>56941.480904</td>
</tr>
<tr>
<td><strong>BIC</strong></td>
<td>57027.973748</td>
</tr>
<tr>
<td><strong>HQC</strong></td>
<td>56970.737507</td>
</tr>
</tbody>
</table>

The accuracy of classification through the Gaussian HMM is calculated by the following statements:

```sas
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.
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.

```sas
proc sgplot data=gHmmCheck;
  scatter y=y x=x / group=ds;
run;
```

**Figure 13.16** Accuracy of Decoding by the Gaussian HMM

<table>
<thead>
<tr>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9999</td>
</tr>
</tbody>
</table>

**Figure 13.17** Classified Data Points by Gaussian HMM
Chapter 13: The HMM Procedure

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 $Y_{n,t} = (y_{n,1}, \ldots, y_{n,T_n})'$, where $n = 1, \ldots, N$ and $t = 1, \ldots, T_n$, 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 $Y_{n,t}$ follows the Gaussian HMM; that is,

$$Y_{n,t} | S_{n,t} \sim N(\mu_{S_{n,t}}, \Sigma_{S_{n,t}})$$

where $\mu_{S_{n,t}}$ and $\Sigma_{S_{n,t}}$ are mean and covariance parameters, respectively, whose values depend on the state variable $S_{n,t}$. $S_{n,t}$ follows the first-order Markov chain; that is,

$$p(S_{n,t} | S_{n,t-1}, S_{n,t-2}, \ldots, S_{n,1}) = p(S_{n,t} | S_{n,t-1})$$

where $p(\cdot | \cdot)$ denotes the conditional probability. $S_{n,t}$ is independent of any $S_{m,s}, m = 1, \ldots, N, s = 1, \ldots, T_m$ and $m \neq n$. The range of $S_{n,t}$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state $i$ to state $j$ is expressed as

$$a_{ij} = p(S_{n,t} = j | S_{n,t-1} = i)$$

The $K \times K$ matrix $A = \{a_{ij}\}$ is called the transition probability matrix (TPM). The initial state probability vector (ISPV), $\pi$, of the first state $S_{n,1}$ is

$$\pi = \{\pi_i = p(S_{n,1} = i), i = 1, \ldots, K\}$$

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 $\{\pi, A, B\}$, where $B \equiv \{\mu_i, \Sigma_i, i = 1, \ldots, K\}$, 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:

$$\pi = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}, A = \begin{pmatrix} 0.95 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.95 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.95 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.95 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.95 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0.01 & 0.01 & 0.95 \end{pmatrix}, B = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \mu_4 \\ \mu_5 \\ \mu_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}, \Sigma = \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \\ \Sigma_3 \\ \Sigma_4 \\ \Sigma_5 \\ \Sigma_6 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 1 \\ 4 \\ 16 \\ 0.16 \\ 0.16 \end{pmatrix}.$$
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 = 12345;
%let nStates = 6;
%let mu1 = 0;
%let sigma1 = 0.25;
%let mu2 = 0;
%let sigma2 = 1;
%let mu3 = 0;
%let sigma3 = 4;
%let mu4 = 0;
%let sigma4 = 16;
%let mu5 = 1;
%let sigma5 = 0.16;
%let mu6 = -1;
%let sigma6 = 0.16;
%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.cstsForcast;
score outmodel=mycas.cstsModel;
run;

The estimates of the ISPV, TPM, mean parameters, and covariance parameters are shown in Output 13.18, Output 13.19, Output 13.20, and Output 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)

<table>
<thead>
<tr>
<th>Initial State Probabilities</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Estimation</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.16616</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.16655</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.16457</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.16579</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.16759</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.16935</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 13.19** Estimates of the Transition Probability Matrix (TPM)

<table>
<thead>
<tr>
<th>Transition Probabilities</th>
<th>State</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0.95011</td>
<td>0.00945</td>
<td>0.00983</td>
<td>0.00967</td>
<td>0.01018</td>
<td>0.01076</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01005</td>
<td>0.95010</td>
<td>0.00988</td>
<td>0.00997</td>
<td>0.00975</td>
<td>0.01025</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.01008</td>
<td>0.00990</td>
<td>0.94955</td>
<td>0.01014</td>
<td>0.01006</td>
<td>0.01027</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.01008</td>
<td>0.00982</td>
<td>0.01012</td>
<td>0.95009</td>
<td>0.01004</td>
<td>0.00986</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.00940</td>
<td>0.00993</td>
<td>0.01020</td>
<td>0.01052</td>
<td>0.95023</td>
<td>0.00972</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.01010</td>
<td>0.01075</td>
<td>0.00966</td>
<td>0.00930</td>
<td>0.01008</td>
<td>0.95011</td>
</tr>
</tbody>
</table>

**Figure 13.20** Estimates of the Mean Parameters

<table>
<thead>
<tr>
<th>Mu</th>
<th>State</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>-1.00060</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.00787</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.00032</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-0.00184</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.00084</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1.00043</td>
</tr>
</tbody>
</table>
Figure 13.21 Estimates of the Covariance Parameters

<table>
<thead>
<tr>
<th>Sigma</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 y</td>
<td>0.16046</td>
</tr>
<tr>
<td>2 y</td>
<td>16.04870</td>
</tr>
<tr>
<td>3 y</td>
<td>4.01224</td>
</tr>
<tr>
<td>4 y</td>
<td>1.00364</td>
</tr>
<tr>
<td>5 y</td>
<td>0.24878</td>
</tr>
<tr>
<td>6 y</td>
<td>0.16030</td>
</tr>
</tbody>
</table>

Gaussian Mixture Hidden Markov Model for Time Series Data and Cross-Sectional Time Series Data

Let \( Y_t = (y_{1t}, \ldots, y_{pt})' \), \( t = 1, \ldots, T \), denote a \( p \)-dimensional time series vector of random variables. Conditional on a latent variable \( S_t \), \( Y_t \) follows a Gaussian mixture distribution,

\[
Y_t|S_t \sim GM(\{w_{S_t,j}, \ldots, w_{S_t,M}\}, \{\mu_{S_t,1}, \ldots, \mu_{S_t,M}\}, \{\Sigma_{S_t,1}, \ldots, \Sigma_{S_t,M}\})
\]

where the variable \( S_t \) is the (hidden) state; \( GM(\ldots) \) represents the Gaussian mixture distribution; \( M \) is the number of components at each state; and the \( w_{S_t,j}, j = 1, \ldots, M \), are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights \( w_{S_t,j} \geq 0, j = 1, \ldots, M \), and \( \sum_{j=1}^{M} w_{S_t,j} = 1 \). The \( \mu_{S_t,j} \) and \( \Sigma_{S_t,j}, j = 1, \ldots, M \), are the mean and covariance parameters for the \( j \)th Gaussian component at state \( S_t \), and their values depend on the variable \( S_t \). The state variable \( S_t \) follows the first-order Markov chain; that is,

\[
p(S_t|S_{t-1}, S_{t-2}, \ldots, S_1) = p(S_t|S_{t-1})
\]

where \( p(\cdot|\cdot) \) denotes the conditional probability. The range of \( S_t \) is a finite set, \( \{1, \ldots, K\} \). The transition probability from state \( i \) to state \( j \) is expressed as

\[
a_{ij} = p(S_t = j|S_{t-1} = i)
\]

The \( K \times K \) matrix \( A = \{a_{ij}\} \) is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), \( \pi \), of the first state \( S_1 \):

\[
\pi = \{\pi_i = p(S_1 = i), i = 1, \ldots, K\}
\]

Because the variable \( S_t \) is unobservable and follows a Markov chain and because \( Y_t \) follows a Gaussian mixture distribution conditional on \( S_t \), the model is called the Gaussian mixture hidden Markov model (GM HMM). The GM HMM is sometimes described as \( \{\pi, A, B\} \), where \( B = \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\} \), 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 526.
Consider a bi-state bi-component GM HMM that has the following parameter values. The ISPV and TPM are

\[
\pi = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{pmatrix}
\]

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

\[
\mu_{11} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \Sigma_{11} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mu_{12} = \begin{pmatrix} -3 \\ -3 \end{pmatrix}, \quad \Sigma_{12} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}
\]

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

\[
\mu_{21} = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \Sigma_{22} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mu_{22} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \quad \Sigma_{22} = \begin{pmatrix} 1.69 & 1.00 \\ 1.00 & 1.69 \end{pmatrix}
\]

The following statements simulate the bivariate vector time series from the previous model to provide test data for the HMM procedure:

```bash
%let a11 = 0.95;
%let a22 = 0.85;
%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 = &mul1_1_1. + cd1_1_11*e1;
        y = &mul1_1_2. + cd1_1_21*e1+cd1_1_22*e2;
      end;
      else do;
        * (x,y) ~ N(mu, Sigma) at state 1, component 2;
        x = &mul1_2_1. + cd1_2_11*e1;
        y = &mul1_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=(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:

```sas
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.
The following statements plot the simulated series $y$ that is contained in the data set gmTrain:

```sas
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`:

```sas
proc sgplot data=gmTrain;
    scatter y=y x=x;
run;
```
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:

```plaintext
proc sgplot data=gmDGP(where=(t<=&T./2 and s=1));
  scatter y=y x=x;
run;
```
Other points are from state 2 in which there are two clusters, as shown in Figure 13.26:

```sas
proc sgplot data=gmDGP(where=(t<=&T./2 and s=2));
  scatter y=y x=x;
run;
```
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:

``` SAS
data mycas.gmTrain; set gmTrain; run;
data mycas.gmTest; set gmTest; run;
```

The following statements estimate the bi-state 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.

``` SAS
proc hmm data=mycas.gmTrain;	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

<table>
<thead>
<tr>
<th>Observation Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Observations</td>
</tr>
<tr>
<td>Number of Missing Observations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Model</td>
</tr>
<tr>
<td>Stationary</td>
</tr>
<tr>
<td>Number of States</td>
</tr>
<tr>
<td>Number of Components for a State</td>
</tr>
<tr>
<td>Number of Dependent Variables</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM1_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM1_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM2_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>TPM2_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>MCP1_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>MCP1_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>MCP2_1</td>
<td>0.500000</td>
</tr>
<tr>
<td>MCP2_2</td>
<td>0.500000</td>
</tr>
<tr>
<td>MU1_1</td>
<td>0.636287</td>
</tr>
<tr>
<td>MU1_1_2</td>
<td>0.797987</td>
</tr>
<tr>
<td>MU1_2</td>
<td>-4.214553</td>
</tr>
<tr>
<td>MU1_2_2</td>
<td>-4.327550</td>
</tr>
<tr>
<td>MU2_1</td>
<td>3.056600</td>
</tr>
<tr>
<td>MU2_1_2</td>
<td>2.438731</td>
</tr>
<tr>
<td>MU2_2</td>
<td>-2.086517</td>
</tr>
<tr>
<td>MU2_2_2</td>
<td>-2.083903</td>
</tr>
<tr>
<td>SIGMA1_1_1_1</td>
<td>0.857343</td>
</tr>
<tr>
<td>SIGMA1_1_2_1</td>
<td>0.061675</td>
</tr>
<tr>
<td>SIGMA1_1_2_2</td>
<td>1.102373</td>
</tr>
<tr>
<td>SIGMA1_2_1_1</td>
<td>0.808210</td>
</tr>
<tr>
<td>SIGMA1_2_2_1</td>
<td>0.044037</td>
</tr>
<tr>
<td>SIGMA1_2_2_2</td>
<td>0.645859</td>
</tr>
<tr>
<td>SIGMA2_1_1_1</td>
<td>0.805066</td>
</tr>
<tr>
<td>SIGMA2_1_2_1</td>
<td>0.534776</td>
</tr>
<tr>
<td>SIGMA2_1_2_2</td>
<td>2.007700</td>
</tr>
<tr>
<td>SIGMA2_2_1_1</td>
<td>1.132478</td>
</tr>
<tr>
<td>SIGMA2_2_1_2</td>
<td>0.235316</td>
</tr>
<tr>
<td>SIGMA2_2_2_2</td>
<td>0.940597</td>
</tr>
</tbody>
</table>

Initial Value of Objective Function -401123.4322
The details of the optimization algorithm are shown in Figure 13.29.

**Figure 13.29** Optimization Algorithm Details

<table>
<thead>
<tr>
<th>Algorithm Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Number of Iterations</td>
<td>128</td>
</tr>
<tr>
<td>Tolerance for Optimality Error</td>
<td>1E-6</td>
</tr>
<tr>
<td>Tolerance for Infeasibility</td>
<td>1E-6</td>
</tr>
<tr>
<td>Maximum Allowed Time (seconds)</td>
<td>1.797693E308</td>
</tr>
<tr>
<td>Maximum Magnitude of Objective Function Value</td>
<td>1E20</td>
</tr>
<tr>
<td>Random Seed</td>
<td>1</td>
</tr>
<tr>
<td>Solution Type</td>
<td>1</td>
</tr>
<tr>
<td>Multi-Start Globalization Scheme</td>
<td>0</td>
</tr>
<tr>
<td>Multi-Start Globalization Bound Range</td>
<td>2</td>
</tr>
</tbody>
</table>

The iterations of the optimization process are shown in Figure 13.30.

**Figure 13.30** Iterations of the Optimization Process

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective Function</th>
<th>Optimality Error</th>
<th>Infeasibility</th>
<th>Function Calls</th>
<th>Gradient Calls</th>
<th>Hessian Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-401123.4322</td>
<td>2782.53245</td>
<td>0.00000</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-382295.0361</td>
<td>687.86414</td>
<td>0.00000</td>
<td>20</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-377230.9712</td>
<td>616.99173</td>
<td>0.00000</td>
<td>42</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>-374911.5887</td>
<td>364.29603</td>
<td>0.00000</td>
<td>60</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>-372637.2842</td>
<td>186.93449</td>
<td>0.00000</td>
<td>78</td>
<td>39</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>-372189.0530</td>
<td>116.35569</td>
<td>0.00000</td>
<td>100</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>-372096.0243</td>
<td>11.48132</td>
<td>0.00000</td>
<td>120</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>-372095.8549</td>
<td>0.02276</td>
<td>0.00000</td>
<td>140</td>
<td>70</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>-372095.8549</td>
<td>0.00000</td>
<td>0.00000</td>
<td>164</td>
<td>82</td>
<td>8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM1_1</td>
<td>0.950464</td>
</tr>
<tr>
<td>TPM1_2</td>
<td>0.049536</td>
</tr>
<tr>
<td>TPM2_1</td>
<td>0.152670</td>
</tr>
<tr>
<td>TPM2_2</td>
<td>0.847330</td>
</tr>
<tr>
<td>MCP1_1</td>
<td>0.598572</td>
</tr>
<tr>
<td>MCP1_2</td>
<td>0.401428</td>
</tr>
<tr>
<td>MCP2_1</td>
<td>0.301149</td>
</tr>
<tr>
<td>MCP2_2</td>
<td>0.698851</td>
</tr>
<tr>
<td>MU1_1_1</td>
<td>0.992382</td>
</tr>
<tr>
<td>MU1_1_2</td>
<td>0.996008</td>
</tr>
<tr>
<td>MU1_2_1</td>
<td>-2.989957</td>
</tr>
<tr>
<td>MU1_2_2</td>
<td>-2.995100</td>
</tr>
<tr>
<td>MU2_1_1</td>
<td>2.968407</td>
</tr>
<tr>
<td>MU2_1_2</td>
<td>2.967274</td>
</tr>
<tr>
<td>MU2_2_1</td>
<td>-1.002345</td>
</tr>
<tr>
<td>MU2_2_2</td>
<td>-1.000781</td>
</tr>
<tr>
<td>SIGMA1_1_1_1</td>
<td>1.002741</td>
</tr>
<tr>
<td>SIGMA1_1_2_1</td>
<td>0.005801</td>
</tr>
<tr>
<td>SIGMA1_1_2_2</td>
<td>1.002483</td>
</tr>
<tr>
<td>SIGMA1_2_1_1</td>
<td>1.715696</td>
</tr>
<tr>
<td>SIGMA1_2_2_1</td>
<td>1.014397</td>
</tr>
<tr>
<td>SIGMA1_2_2_2</td>
<td>1.700003</td>
</tr>
<tr>
<td>SIGMA2_1_1_1</td>
<td>1.700481</td>
</tr>
<tr>
<td>SIGMA2_1_2_1</td>
<td>1.003522</td>
</tr>
<tr>
<td>SIGMA2_1_2_2</td>
<td>1.710452</td>
</tr>
<tr>
<td>SIGMA2_2_1_1</td>
<td>0.995587</td>
</tr>
<tr>
<td>SIGMA2_2_2_1</td>
<td>0.009321</td>
</tr>
<tr>
<td>SIGMA2_2_2_2</td>
<td>1.003323</td>
</tr>
</tbody>
</table>

*Final Value of Objective Function*  -372095.8549

The estimates of the initial state probability vector (ISPV) are shown in **Figure 13.32**.

**Figure 13.32** Estimates of Initial State Probability Vector

<table>
<thead>
<tr>
<th>State</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.75502</td>
</tr>
<tr>
<td>2</td>
<td>0.24498</td>
</tr>
</tbody>
</table>
The estimates of the transition probability matrix (TPM) are shown in Figure 13.33.

**Figure 13.33** Estimates of Transition Probability Matrix

<table>
<thead>
<tr>
<th>Transition Probabilities</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>2</td>
</tr>
<tr>
<td>1 0.95046 0.04954</td>
<td></td>
</tr>
<tr>
<td>2 0.15267 0.84733</td>
<td></td>
</tr>
</tbody>
</table>

The estimates of the mixture component probabilities (MCP) are shown in Figure 13.34.

**Figure 13.34** Estimates of Mixture Component Probabilities

<table>
<thead>
<tr>
<th>Mixture Component Probabilities</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Component 1</td>
<td>2</td>
</tr>
<tr>
<td>State 1</td>
<td>2</td>
</tr>
<tr>
<td>1 0.59857 0.40143</td>
<td></td>
</tr>
<tr>
<td>2 0.30115 0.69885</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Mu</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Component 1</td>
<td>0.99238</td>
<td>0.99601</td>
</tr>
<tr>
<td>1</td>
<td>-2.98996</td>
<td>-2.99510</td>
</tr>
<tr>
<td>2</td>
<td>2.96841</td>
<td>2.96727</td>
</tr>
<tr>
<td></td>
<td>-1.00235</td>
<td>-1.00078</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Sigma Variate</th>
<th>State Component 1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td>1.00274 0.00580</td>
<td></td>
</tr>
<tr>
<td>1 y</td>
<td>0.00580 1.00248</td>
<td></td>
</tr>
<tr>
<td>2 x</td>
<td>1.71570 1.01440</td>
<td></td>
</tr>
<tr>
<td>2 y</td>
<td>1.01440 1.70000</td>
<td></td>
</tr>
<tr>
<td>1 x</td>
<td>1.70048 1.00352</td>
<td></td>
</tr>
<tr>
<td>1 y</td>
<td>1.00352 1.71045</td>
<td></td>
</tr>
<tr>
<td>2 x</td>
<td>0.99559 0.00932</td>
<td></td>
</tr>
<tr>
<td>2 y</td>
<td>0.00932 1.00332</td>
<td></td>
</tr>
</tbody>
</table>
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   | Estimate | Standard Error | t Value | Pr > |t| |
|-------------|----------|----------------|---------|------|--|--|---|
| TPM1_1      | 0.95046  | 0.001099       | 864.85  | <.0001 |
| TPM1_2      | 0.049536 | 0.001099       | 45.07   | <.0001 |
| TPM2_1      | 0.152670 | 0.003013       | 50.68   | <.0001 |
| TPM2_2      | 0.847330 | 0.003013       | 281.26  | <.0001 |
| MCP1_1      | 0.598572 | 0.002027       | 295.33  | <.0001 |
| MCP1_2      | 0.401428 | 0.002027       | 198.06  | <.0001 |
| MCP2_1      | 0.301149 | 0.003682       | 81.79   | <.0001 |
| MCP2_2      | 0.698851 | 0.003682       | 189.81  | <.0001 |
| MU1_1_1     | 0.992382 | 0.005272       | 188.22  | <.0001 |
| MU1_1_2     | 0.996008 | 0.005281       | 188.62  | <.0001 |
| MU1_2_1     | -2.989957| 0.010033       | -298.00 | <.0001 |
| MU1_2_2     | -2.995100| 0.009976       | -300.23 | <.0001 |
| MU2_1_1     | 2.968407 | 0.023308       | 127.35  | <.0001 |
| MU2_1_2     | 2.967274 | 0.023266       | 127.54  | <.0001 |
| MU2_2_1     | -1.002345| 0.009253       | 108.32  | <.0001 |
| MU2_2_2     | -1.000781| 0.009298       | 107.64  | <.0001 |
| SIGMA1_1_1_1_1 | 1.002741 | 0.007833 | 128.01 | <.0001 |
| SIGMA1_1_2_1 | 0.005801 | 0.005866 | 0.99  | 0.3227 |
| SIGMA1_1_2_2 | 1.002483 | 0.007852 | 127.68 | <.0001 |
| SIGMA1_2_1_1 | 1.715696 | 0.018717 | 91.67 | <.0001 |
| SIGMA1_2_1_2 | 1.014397 | 0.015989 | 63.44 | <.0001 |
| SIGMA1_2_2_2 | 1.700003 | 0.018530 | 91.74 | <.0001 |
| SIGMA2_1_1_1 | 1.700481 | 0.040949 | 41.53 | <.0001 |
| SIGMA2_1_2_1 | 1.003522 | 0.034984 | 28.68 | <.0001 |
| SIGMA2_1_2_2 | 1.710452 | 0.040805 | 41.92 | <.0001 |
| SIGMA2_2_1_1 | 0.995587 | 0.014146 | 70.38 | <.0001 |
| SIGMA2_2_2_1 | 0.009321 | 0.010652 | 0.87 | 0.3816 |
| SIGMA2_2_2_2 | 1.003323 | 0.014082 | 71.25 | <.0001 |
The log likelihood and information criteria are shown in Figure 13.38.

![Figure 13.38 Log Likelihood and Information Criteria](image)

<table>
<thead>
<tr>
<th>Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Log Likelihood</strong></td>
</tr>
<tr>
<td>AIC</td>
</tr>
<tr>
<td>AICC</td>
</tr>
<tr>
<td>BIC</td>
</tr>
<tr>
<td>HQC</td>
</tr>
</tbody>
</table>

The estimates from the data table `gmTrain` are used to score the testing data table `gmTest` to check the predictability in the following statements:

```sql
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:

```sql
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 94%. This means that the GM HMM successfully takes advantage of the sequential information.

**Figure 13.39** Accuracy of Decoding by the GM HMM

<table>
<thead>
<tr>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.94451</td>
</tr>
</tbody>
</table>

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.

```sas
%let a11 = 0.95;
%let a22 = 0.85;
%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;
   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.

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Model</td>
</tr>
<tr>
<td>Stationary</td>
</tr>
<tr>
<td>Gaussian Mixture HMM</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>Number of States</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Number of Components for a State</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Number of Dependent Variables</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Number of Sections</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

The estimates of all parameters are shown in Figure 13.41; the estimates are very close to the parameter values in the DGP.
You can apply the estimates from the data table gmTrain to the testing data table gmTest to check the predictability in the following statements:

```sas
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;
```

You can apply the estimates from the data table gmTrain to the testing data table gmTest to check the predictability in the following statements:

```sas
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;
```
Regime-Switching Regression Model

Let $Y_t = (y_{1t}, \ldots, y_{pt})', t = 1, \ldots, T$, denote a $p$-dimensional time series vector of random variables. The $Y_t$ can be modeled in the regression

$$Y_t = \beta S_t z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_{S_t})$, the regressor $z_t$ is observable, the latent variable $S_t$ is the so-called state, and $\beta S_t$ and $\Sigma_{S_t}$ are mean and covariance parameters whose values depend on the state $S_t$. The variable $S_t$ follows the first-order Markov chain; that is,

$$p(S_t|S_{t-1}, S_{t-2}, \ldots, S_1) = p(S_t|S_{t-1})$$

where $p(\cdot|\cdot)$ denotes the conditional probability. The range of $S_t$ is a finite set, $\{1, \ldots, K\}$. The transition probability from state $i$ to state $j$ is expressed as

$$a_{ij} = p(S_t = j|S_{t-1} = i)$$
The $K \times K$ matrix $A = \{a_{ij}\}$ is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), $\pi$, of the first state $S_1$:

$$\pi = \{\pi_i = p(S_1 = i), i = 1, \ldots, K\}$$

Because the regression parameters depend on the state, the $Y_t$ 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):

$$y_t = \begin{cases} x_t + \varepsilon_t, \varepsilon_t \sim N(0, 2.56) & \text{if } S_t = 1 \\ 4 + 1.5x_t + \varepsilon_t, \varepsilon_t \sim N(0, 4) & \text{if } S_t = 2 \end{cases}$$

The initial state probability vector and transition probability matrix are as follows:

$$\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, A = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$$

The following statements simulate the time series from the previous model to provide test data for the HMM procedure:

```sas
%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. + &xl1_0_1_1 * x + cd1_11*e;
   end;
   else do;
      /* y ~ N(beta2_0+beta2_1*x, Sigma2) at state 2 */
      y = &const2_1. + &xl2_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;
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:

```sas
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 | Estimate | Standard Error | \( t \) Value | \( p > |t| \) |
|-----------|----------|----------------|--------------|--------------|
| TPM1_1    | 0.942958 | 0.007335       | 128.56       | <.0001       |
| TPM1_2    | 0.057042 | 0.007335       | 7.78         | <.0001       |
| TPM2_1    | 0.049221 | 0.006259       | 7.86         | <.0001       |
| TPM2_2    | 0.950779 | 0.006259       | 151.91       | <.0001       |
| CONST1_1  | 4.105147 | 0.113280       | 36.24        | <.0001       |
| CONST2_1  | -0.058860| 0.079933       | -0.74        | 0.4616       |
| XL1_0_1_1 | 1.513899 | 0.022254       | 68.03        | <.0001       |
| XL2_0_1_1 | 0.986513 | 0.016304       | 60.51        | <.0001       |
| COV1_1_1  | 3.839914 | 0.143172       | 26.82        | <.0001       |
| COV2_1_1  | 2.563899 | 0.094248       | 27.20        | <.0001       |

The accuracy of classification through the regime-switching regression model is calculated by the following statements:

```sas
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;
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

<table>
<thead>
<tr>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.89925</td>
</tr>
</tbody>
</table>

The following statements create the scatter plot in Figure 13.46 to show how well the data points are classified:

```sas
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 \( Y_t = (y_{1t}, \ldots, y_{pt})' \), \( t = 1, \ldots, T \), denote a \( p \)-dimensional time series vector of random variables. The \( Y_t \) can be modeled in the regression

\[
Y_t = \beta_s z_t + \epsilon_t
\]

where \( \epsilon_t \sim N(0, \Sigma_s) \), the regressor \( z_t \) is observable and contains the lagged dependent variables, the latent variable \( S_t \) is the so-called state, and \( \beta_s \) and \( \Sigma_s \) are mean and covariance parameters whose values depend on the state \( S_t \). The variable \( S_t \) follows the first-order Markov chain; that is,

\[
p(S_t|S_{t-1}, S_{t-2}, \ldots, S_1) = p(S_t|S_{t-1})
\]

where \( p(\cdot, \cdot) \) denotes the conditional probability. The range of \( S_t \) is a finite set, \( \{1, \ldots, K\} \). The transition probability from state \( i \) to state \( j \) is expressed as

\[
a_{ij} = p(S_t = j|S_{t-1} = i)
\]

The \( K \times K \) matrix \( A = \{a_{ij}\} \) is called the transition probability matrix (TPM). The last element in the model is the initial state probability vector (ISPV), \( \pi \), of the first state \( S_1 \):

\[
\pi = \{\pi_i = p(S_1 = i), i = 1, \ldots, K\}
\]

Because the lagged dependent variables are included in the regressors and the autoregression parameters depend on the state, the \( Y_t \) 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:

\[
y_t = \begin{cases} 
0.8 * y_{t-1} + \epsilon_t, \epsilon_t \sim N(0, 2.56) & \text{if } S_t = 1 \\
-0.7 * y_{t-1} + \epsilon_t, \epsilon_t \sim N(0, 4) & \text{if } S_t = 2
\end{cases}
\]

The initial state probability vector and transition probability matrix are as follows:

\[
\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, A = \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix}
\]

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 = &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;
    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;
The following code uploads the data table and estimates the regime-switching autoregression model:

```plaintext
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.
The accuracy of classification through the regime-switching autoregression model is calculated by the following statements:

```sas
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;
```

### Table 13.48 Parameter Estimates

| Parameter  | Estimate | Standard Error | t Value | Pr > |t| |
|------------|----------|----------------|---------|------|---|
| 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 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

<table>
<thead>
<tr>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.927</td>
</tr>
</tbody>
</table>

The following statements create the scatter plot in Figure 13.50 to show how well the data points are classified:

```r
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 options;
  DECODE options;
  DISPLAY < table-list > </ options >;
  DISPLAYOUT table-spec-list </ options >;
  ESTIMATE options;
  EVALUATE options;
  FILTER options;
  FORECAST options;
  ID TIME= variable < SECTION= variable >;
  INITIAL equation, . . . , equation;
  MODEL dependents < = regressors > < / options >;
  OPTIMIZE options;
  PRIOR equation, . . . , equation;
  SCORE options;
  SMOOTH options;

Functional Summary

The statements and options available in the HMM procedure are summarized in Table 13.1.

Table 13.1 Functional Summary

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Set Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data table</td>
<td>PROC HMM</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the input data table for parameter estimation</td>
<td>ESTIMATE</td>
<td>IN=</td>
</tr>
<tr>
<td>Specifies the input data table for model information</td>
<td>SCORE</td>
<td>INMODEL=</td>
</tr>
<tr>
<td>Writes the parameter estimates to the specified output data table</td>
<td>ESTIMATE</td>
<td>OUT=</td>
</tr>
<tr>
<td>Writes the evaluation results to the specified output data table</td>
<td>EVALUATE</td>
<td>OUT=</td>
</tr>
<tr>
<td>Writes the decoding results to the specified output data table</td>
<td>DECODE</td>
<td>OUT=</td>
</tr>
<tr>
<td>Writes the filtering results to the specified output data table</td>
<td>FILTER</td>
<td>OUT=</td>
</tr>
<tr>
<td>Writes the forecasting results to the specified output data table</td>
<td>FORECAST</td>
<td>OUT=</td>
</tr>
<tr>
<td>Writes the log likelihood and information criteria to the specified output data table</td>
<td>PROC HMM</td>
<td>OUTSTAT=</td>
</tr>
<tr>
<td>Writes the model information to the specified output data table</td>
<td>SCORE</td>
<td>OUTMODEL=</td>
</tr>
<tr>
<td>Writes the smoothing results to the specified output data table</td>
<td>SMOOTH</td>
<td>OUT=</td>
</tr>
</tbody>
</table>
Table 13.1  continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ID Variable</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the time or sequence identifying variable</td>
<td>ID</td>
<td>TIME=</td>
</tr>
<tr>
<td>Specifies the section identifying variable</td>
<td>ID</td>
<td>SECTION=</td>
</tr>
<tr>
<td><strong>Model Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimates the initial state probability vector (ISPV)</td>
<td>MODEL</td>
<td>ESTISPV</td>
</tr>
<tr>
<td>Specifies the estimation method</td>
<td>MODEL</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies the number of components in a state in a Gaussian mixture HMM (GM HMM)</td>
<td>MODEL</td>
<td>NCOMPONENT</td>
</tr>
<tr>
<td>Suppresses the present values of the exogenous variables</td>
<td>MODEL</td>
<td>NOCURRENTX</td>
</tr>
<tr>
<td>Suppresses the constant (intercept) parameters</td>
<td>MODEL</td>
<td>NOINT</td>
</tr>
<tr>
<td>Specifies the number of seasonal periods</td>
<td>MODEL</td>
<td>NSEASON=</td>
</tr>
<tr>
<td>Specifies the number or the range of numbers of hidden states</td>
<td>MODEL</td>
<td>NSTATE=</td>
</tr>
<tr>
<td>Centers seasonal dummies</td>
<td>MODEL</td>
<td>SCENTER</td>
</tr>
<tr>
<td>Specifies the degree of deterministic time trend</td>
<td>MODEL</td>
<td>TREND=</td>
</tr>
<tr>
<td>Specifies the type of model of interest</td>
<td>MODEL</td>
<td>TYPE=</td>
</tr>
<tr>
<td>Specifies the lags of exogenous (independent) variables</td>
<td>MODEL</td>
<td>XLAG=</td>
</tr>
<tr>
<td>Specifies the order or the range of orders of the autoregressive process</td>
<td>MODEL</td>
<td>YLAG=</td>
</tr>
<tr>
<td><strong>Optimization Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the initial parameter values for optimization</td>
<td>INITIAL</td>
<td></td>
</tr>
<tr>
<td>Specifies the optimization algorithm</td>
<td>OPTIMIZE</td>
<td>ALGORITHM=</td>
</tr>
<tr>
<td>Specifies the tolerance for the infeasibility</td>
<td>OPTIMIZE</td>
<td>FEASTOL=</td>
</tr>
<tr>
<td>Specifies the maximum number of iterations in the optimization process</td>
<td>OPTIMIZE</td>
<td>MAXITER=</td>
</tr>
<tr>
<td>Specifies the maximum time (in seconds) allowed for optimization</td>
<td>OPTIMIZE</td>
<td>MAXTIME=</td>
</tr>
<tr>
<td>Specifies the range from which each variable can take values during the sampling process in multistart mode</td>
<td>OPTIMIZE</td>
<td>MSRANGE=</td>
</tr>
<tr>
<td>Specifies whether to enable multistart mode</td>
<td>OPTIMIZE</td>
<td>MULTISTART=</td>
</tr>
<tr>
<td>Specifies the upper limit on the magnitude of the objective value</td>
<td>OPTIMIZE</td>
<td>OBJLIMIT=</td>
</tr>
<tr>
<td>Specifies the tolerance for the optimality error</td>
<td>OPTIMIZE</td>
<td>OPTTOL=</td>
</tr>
<tr>
<td>Specifies the print iteration frequency</td>
<td>OPTIMIZE</td>
<td>PRINTITERFREQ=</td>
</tr>
<tr>
<td>Specifies the print level</td>
<td>OPTIMIZE</td>
<td>PRINTLEVEL=</td>
</tr>
<tr>
<td>Specifies whether the optimizer should return only a solution that is locally optimal</td>
<td>OPTIMIZE</td>
<td>SOLTYPE=</td>
</tr>
</tbody>
</table>
Table 13.1  continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the priors for parameters</td>
<td>PRIOR</td>
<td></td>
</tr>
<tr>
<td>Specifies a nonnegative integer to be used as the seed for generating</td>
<td>PROC HMM</td>
<td>SEED</td>
</tr>
<tr>
<td>random number sequences</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Forecasting Options**

| 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    |
| Specifies the number of paths to be simulated in the forecasting of a        | FORECAST  | NSIM    |
| regime-switching autoregression model                                        |           |         |
| Specifies whether to do forecasts after each observation                     | FORECAST  | ONLINE  |
| Specifies a nonnegative integer to be used as the seed for generating      | PROC HMM  | SEED    |
| random number sequences                                                       |           |         |

**Output Control Options**

| Specifies the ODS tables to display                                         | DISPLAY   |         |
| Specifies the ODS tables to save as CAS output tables                       | DISPLAYOUT|         |

---

**PROC HMM Statement**

PROC HMM options;

The PROC HMM statement invokes the HMM procedure. You can specify the following options:

**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

*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 513.

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

**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 513.

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 specifies options that are related to the decoding problem, which is to find 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 513.

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 602. 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 subsetted 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 602. 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**

```syntax
ESTIMATE options ;
LEARN options ;
```

The ESTIMATE statement specifies options that are related to parameter estimation. 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 513.

**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 513.

---

**EVALUATE Statement**

```syntax
EVALUATE option ;
```

The EVALUATE statement specifies options that are related to the evaluation, which is the log-likelihood calculation. You can specify the following option:
OUT=\textit{CAS-libref.data-table}

writes the evaluation results to the specified output data table. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 513.

**FILTER Statement**

\begin{verbatim}
FILTER \textit{option} ;
\end{verbatim}

The FILTER statement specifies options that are related to 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 \textit{option}:

\begin{verbatim}
OUT=\textit{CAS-libref.data-table}
\end{verbatim}

writes the filtering results to the specified output data table. \textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the caslib and session identifier, and \textit{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 513.

**FORECAST Statement**

\begin{verbatim}
FORECAST \textit{options} ;
\end{verbatim}

The FORECAST statement specifies options that are related to the forecasting problem, which is to find the probability distribution of the future states and the dependent variables. You can specify the following \textit{options}:

\begin{verbatim}
ALPHA=\textit{number}
\end{verbatim}

specifies the forecast confidence limit size, where \textit{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.

\begin{verbatim}
BACK=\textit{number}
\end{verbatim}

specifies the number of observations before the end of the data at which the multistep forecasts begin, where \textit{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.

\begin{verbatim}
LEAD=\textit{number}
\end{verbatim}

specifies the number of multistep forecast values to compute, where \textit{number} must be a nonnegative integer. By default, LEAD=1, which means that a one-step-ahead forecast is performed.

\begin{verbatim}
NSIM=\textit{number}
\end{verbatim}

specifies the number of paths to be simulated in the forecasting of a regime-switching autoregression model, where \textit{number} must be a positive integer. By default, NSIM=100.
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 the distribution operator ~ are not supported in the INITIAL statement.

If you specify the NSTATE=n1 : n2 option or the YLAG=m1 : m2 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=n1 or YLAG=m1 (or that corresponds to both NSTATE=n1 and YLAG=m1 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 588.
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 $\text{SIGMA}$ or $\text{COV}$ parameter matrix are set to ones if the $\text{SIGMA}$ or $\text{COV}$ parameter matrix is to be estimated.
- If a row of the $\text{TPM}$ 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 $\text{TPM}$ 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 $\text{ISPV}$ parameter vector are zeros, each element of the $\text{ISPV}$ parameter vector is set to $1/K$, where $K$ is the number of states; if the sum of the $\text{ISPV}$ parameter vector is not equal to one, it is normalized by dividing each element of the $\text{ISPV}$ parameter vector by the sum of the $\text{ISPV}$ 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:

```plaintext
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 (/):

- **ESTISPV** estimates the initial state probability vector (ISPV). If you do not specify this option, 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 $\pi$ such that $\pi' A = \pi'$, where $A$ is the transition probability matrix (TPM); that is, $\pi$ is the leading left eigenvector of $A$. If you do not specify ESTISPV in the MODEL statement but specify ISPV in the INITIAL or PRIOR statement, the HMM procedure issues errors and then stops.

- **METHOD=value** requests the type of estimates to be computed. You can specify the following values:
MODEL Statement ♦ 569

**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.

**TREND=value**
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=value**
specifies the type of the model of interest. You can specify the following values:

**AR** specifies the regime-switching autoregression model (RS-AR, also known as the autoregressive hidden Markov model, AR HMM).

**GAUSSIAN** specifies the Gaussian hidden Markov model (Gaussian HMM).
**Chapter 13: The HMM Procedure**

**GAUSSIANMIXTURE** specifies the Gaussian mixture hidden Markov model (GM HMM).

**REG** specifies 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. By default, XLAG=0. 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 specifies options that are related to optimization.

You can specify the following **options**:

**ALGORITHM=** *value*

specifies the optimization algorithm. You can specify the following **values**:

**ACTIVESET** specifies the active set algorithm.

**INTERIORPOINT** specifies the interior point algorithm.

By default, ALGORITHM=ACTIVESET.

**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= option. By default, FEASTOL=10^-6.

**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. If you do not specify this option, the optimizer does not stop on the basis of the amount of time elapsed. The default value is the positive number that has the largest absolute value that your operating environment can represent.
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=10^{20}. If number is less than $10^8$, it is reset to the default value 10^{20}.

OPTTOL=\epsilon
specifies the measure by which the current iterate is determined to be an acceptable approximation of a local optimum, where \epsilon 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 \epsilon and the true constraint violation is less than the value of the FEASTOL= option. By default, OPTTOL=10^{-6}.

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 PRINTITERFREQ=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 and objective value, the algorithm information, and the final parameter values and objective value.
1      outputs the initial parameter values and objective value.
2      outputs the initial parameter values and objective value, and the algorithm information.
3      outputs the initial parameter values and objective value, the algorithm information, and the final parameter values and objective value.

By default, PRINTLEVEL=0.
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 found, provided that its objective value is better than that of the locally optimal solution found.

By default, SOLTYPE=1.

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 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 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:

\texttt{parameter-name \sim distribution-function(arguments)} ;

You can specify the following parameter-names:

\begin{itemize}
  \item **ISPV** indicates the initial state probability vector. The prior distribution must be the Dirichlet distribution.
  \item **MCP** indicates the mixture component probabilities in the Gaussian mixture HMM (GM HMM). The prior distribution must be the Dirichlet distribution.
  \item **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.
  \item **TPM** indicates the transition probability matrix. The prior distribution must be the Dirichlet distribution.
\end{itemize}

You can specify the following distribution-functions:
DIR specifies the Dirichlet distribution. There is one argument 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, or TPM, the size of the argument must be the same as the size of the ISPV, MCP, or TPM. All elements of the argument should be positive. If you specify any nonpositive element in the argument, 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 588. For more information about the Dirichlet distribution, see the section “Maximum a Posteriori (MAP) Method” on page 583.

NIW specifies the normal-inverse-Wishart (NIW) distribution. There are four arguments for the NIW distribution (, , , and ), 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:

- is a matrix, where is the number of states, and is the number of dependent variables.
- is a matrix.
- is a vector.
- is a vector.

If the NIW distribution is the prior distribution of the MUSIGMA for GM HMM, then the following statements are true:

- is a matrix, where is the number of states, is the number of components at a state, and is the number of dependent variables.
- is a matrix.
- is a vector.
- is a 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:

- is a matrix, where is the number of states, is the number of dependent variables, and is the number of regressors in each equation. (For more information about how to calculate , see the sections “Regime-Switching Regression Model” on page 580 and “Regime-Switching Autoregression Model” on page 581.)
- is a matrix.
- is a matrix.
- is a vector.

For information about how to define the matrix expressions, see the section “Matrix Expression” on page 588. For more information about the normal-inverse-Wishart (NIW) distribution, see the section “Maximum a Posteriori (MAP) Method” on page 583.

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 513.

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 513.

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 513.

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 \( \{S_t, Y_t\} \), where \( \{S_t\} \) is a (hidden) Markov chain; conditional on \( \{S_t\} \), the observable process \( \{Y_t\} \) is a sequence of independent random variables such that the conditional distribution of \( Y_t \) depends only on \( S_t \). The \( S_t \) is often called the state. In general, the HMM can be expressed in three equations:

1. Initialization equation: \( S_1 \sim h_\theta(\cdot) \)
2. Transition equation: \( S_t \sim f_\theta(\cdot | S_{t-1}) \)
3. Observation equation: \( Y_t \sim g_\theta(\cdot | S_t) \)

Here the initial probability distribution function \( h_\theta(\cdot) \) is the probability distribution function for initial state \( S_1 \); the transition probability distribution function \( f_\theta(\cdot | S_{t-1}) \) is the probability distribution function of current state \( S_t \) conditional on the past state \( S_{t-1} \); the observation probability distribution function \( g_\theta(\cdot | S_t) \) is the probability distribution function of current observation \( Y_t \) conditional on the current state \( S_t \); and all probability distribution functions might depend on the parameter \( \theta \).

There is no restriction on the dimensionality of \( S_t \) and \( Y_t \); 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 \( S_t \) could take discrete or continuous values or even a mix of discrete and continuous values.\(^2\) In the simplest and most popular case, \( S_t \) is a scalar and takes finite discrete values; that is, \( S_t \in \{1, \ldots, K\} \). 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 \( \pi \), which is called the initial state probability vector (ISPV); that is,

\[
\pi = \{\pi_i = p(S_1 = i), i = 1, \ldots, K\}
\]

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 \( A \), which is called the transition probability matrix (TPM), where an element

\(^2\)In 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 \( A \) denotes the probability transitioning from past state \( i \) to current state \( j \); that is,

\[
a_{ij} = p(S_t = j | S_{t-1} = i)
\]

There are six common problems to solve for an HMM:

- Filtering problem: What is \( p(S_t | Y_1, \ldots, Y_t) \), the probability distribution of state \( S_t \) given observations \( Y_1, \ldots, Y_t \)?

- Smoothing problem: What is \( p(S_t | Y_1, \ldots, Y_T) \), the probability distribution of state \( S_t \) given observations \( Y_1, \ldots, Y_T \), where \( T \) is the sample size?

- Forecasting problem: What is \( p(S_{t+h} | Y_1, \ldots, Y_t) \), \( h > 0 \), the probability distribution of state \( S_{t+h} \) given observations \( Y_1, \ldots, Y_t \)?

- Evaluating problem: What is \( p(Y_1, \ldots, Y_T) \), the probability (or the likelihood) of observations \( Y_1, \ldots, Y_T \)?

- Decoding problem: What is \( \arg \max_{S_1, \ldots, S_T} p(S_1, \ldots, S_T, Y_1, \ldots, Y_T) \), the most likely sequence of hidden states \( S_1, \ldots, S_T \) given observations \( Y_1, \ldots, Y_T \), where \( T \) is the sample size?

- Learning problem: How do you estimate the unknown parameter \( \theta \) given observations \( Y_1, \ldots, Y_T \), 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 \( \theta \). For more information, see the section “Parameter Estimation Methods” on page 583.

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 \( t \), \( S_t \), and all observations up to time \( t \), \( Y_1, \ldots, Y_t \). First, define \( \alpha_t(i), i = 1, \ldots, K \), as the joint probability of \( (S_t = i) \) and \( Y_1, \ldots, Y_t \):

\[
\alpha_t(i) = p(S_t = i, Y_1, \ldots, Y_t)
\]

Then, \( \alpha_t(i) \) can be recursively calculated by the following steps:

1. Calculate \( \alpha_1(i), i = 1, \ldots, K \), by

\[
\alpha_1(i) = \pi_i g_\theta(Y_1 | S_1 = i)
\]
2. Calculate $\alpha_t(i), i = 1, \ldots, K, t = 2, \ldots, T,$ by

$$\alpha_t(i) = \left( \sum_{j=1}^{K} \alpha_{t-1}(j) a_{ji} \right) g_\theta(Y_t|S_t = i)$$

You can solve the evaluating problem by

$$L_t = p(Y_1, \ldots, Y_t) = \sum_{i=1}^{K} \alpha_{t}(i)$$

Because the likelihood of all observations, $L_T$, can be calculated, you can apply the maximum likelihood method to estimate the unknown parameter $\theta$; if the prior distribution of the parameter $\theta$ 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

$$p_t^{(f)}(i) \equiv p(S_t = i|Y_1, \ldots, Y_t) = \alpha_t(i)/L_t$$

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

$$p_{t+h|t}^{(p)} = (A')^h p_t^{(f)}$$

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 $p_{t+h|t}^{(p)}(i)$:

$$p_{t+h|t}^{(p)}(i) \equiv p(S_{t+h} = i|Y_1, \ldots, Y_t)$$

The distribution of $Y_{t+h}$ conditional on observations $Y_1, \ldots, Y_t$ is the following mixture distribution:

$$Y_{t+h}|Y_1, \ldots, Y_t \sim \sum_{i=1}^{K} p_{t+h|t}^{(p)}(i) g_\theta(\cdot|S_{t+h} = i)$$

**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 $Y_{t+1}, \ldots, Y_T$ conditional on the current state $S_t = i$:

$$\beta_t(i) \equiv p(Y_{t+1}, \ldots, Y_T|S_t = i)$$

The backward algorithm recursively calculates $\beta_t(i)$ as follows:
1. Set $\beta_T(i) = 1, i = 1, \ldots, K$.
2. Calculate $\beta_t(i), i = 1, \ldots, K, t = T - 1, \ldots, 1$, by

$$\beta_t(i) = \sum_{j=1}^{K} a_{ij} g_\theta(Y_{t+1}|S_{t+1} = j) \beta_{t+1}(j)$$

When you calculate both $\alpha_t(i)$ and $\beta_t(i), i = 1, \ldots, K, t = 1, \ldots, T$, by using the forward algorithm and backward algorithm, the smoothing problem is solved, because you can calculate the probability of state $S_t$ taking value $i$ given all observations $Y_1, \ldots, Y_T$ as follows:

$$\gamma_t(i) \equiv p(S_t = i|Y_1, \ldots, Y_T) = \frac{\alpha_t(i) \beta_t(i)}{\sum_{j=1}^{K} \alpha_t(j) \beta_t(j)}$$

**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 $S_t = i$ and observations $Y_1, \ldots, Y_t$ as follows:

$$V_t(i) = \max_{S_1, \ldots, S_{t-1}} p(S_1, \ldots, S_{t-1}, S_t = i, Y_1, \ldots, Y_t)$$

To keep track of the best path, for $t = 2, \ldots, T$, 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 $V_1(i), i = 1, \ldots, K$, by

$$V_1(i) = \pi_i g_\theta(Y_1|S_1 = i)$$

2. Calculate $V_t(i)$ and $v_t(i), i = 1, \ldots, K, t = 2, \ldots, T$, by

$$V_t(i) = \left( \max_j V_{t-1}(j) a_{ji} \right) g_\theta(Y_t|S_t = i)$$

$$v_t(i) = \arg \max_j V_{t-1}(j) a_{ji}$$

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

$$v_t^* = v_{t+1}(v_{t+1}^*), t = T - 1, \ldots, 1$$

That is, $\{v_t^*\}_{t=1,T}$ 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,

\[ Y_t | S_t \sim N(\mu_{S_t}, \Sigma_{S_t}) \]

where \( \mu_{S_t} \) and \( \Sigma_{S_t} \) are mean and covariance parameters at state \( S_t \), \( S_t = 1, \ldots, K \). Hence, the initial state probability vector (ISPV) \( \pi \), the transition probability matrix (TPM) \( A \), and the observation parameter \( B \) (\( \equiv \{\mu_i, \Sigma_i\}_{i=1,\ldots,K} \), which consists of mean and covariance parameters) together specify the Gaussian HMM; that is, the parameter \( \theta \) of the Gaussian HMM is \( \{\pi, A, B\} \).

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 575. 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,

\[ Y_t | S_t \sim GM\{w_{S_t,1}, \ldots, w_{S_t,M}; \mu_{S_t,1}, \ldots, \mu_{S_t,M}; \Sigma_{S_t,1}, \ldots, \Sigma_{S_t,M}\} \]

where the variable \( S_t \) is the (hidden) state; \( GM\{\ldots\} \) represents the Gaussian mixture distribution; \( M \) is the number of components at each state; and the \( w_{S_t,j}, j = 1, \ldots, M \) are mixture component probabilities (MCPs, also called mixture weights), which satisfy the basic requirement for weights (\( w_{S_t,j} \geq 0, j = 1, \ldots, M \), and \( \sum_{j=1}^{M} w_{S_t,j} = 1 \)). The \( \mu_{S_t,j} \) and \( \Sigma_{S_t,j}, j = 1, \ldots, M \), are the mean and covariance parameters for the \( j \)th Gaussian component at state \( S_t \), and their values depend on the variable \( S_t \). Hence, the initial state probability vector (ISPV) \( \pi \), the transition probability matrix (TPM) \( A \), and the observation parameter \( B \) (\( \equiv \{w_{ij}, \mu_{ij}, \Sigma_{ij}, i = 1, \ldots, K, j = 1, \ldots, M\} \), which consists of the mixture component probabilities, the mean and covariance parameters for all components) together specify the GM HMM; that is, the parameter \( \theta \) of the GM HMM is \( \{\pi, A, B\} \).

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 575. 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,

\[ Y_t | z_t, S_t \sim N(B \Sigma S, \Sigma S) \]

where \( B \Sigma S \) and \( \Sigma S \) are the mean and covariance parameters, respectively, at state \( S_t, S_t = 1, \ldots, K \). Hence, the initial state probability vector (ISPV) \( \pi \), the transition probability matrix (TPM) \( A \), and the observation parameter \( B (\| B_i, \Sigma_i \|_{i=1,..,K} \) which consists of mean and covariance parameters) together specify the RS REG; that is, the parameter \( \theta \) of the RS REG is \( \{ \pi, A, B \} \).

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,

\[ z_t = \left( z_t^{(\text{CONST})}, z_t^{(\text{SD})}, z_t^{(\text{LTREND})}, z_t^{(\text{QTREND})}, z_t^{(\text{XL})} \right)' \]

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, \( z_t^{(\text{SD})} = (sd_1, \ldots, sd_{n_s-1})' \), where \( sd_i, i = 1, \ldots, n_s - 1 \), 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,
    \[ z_t^{(\text{XL})} = (x_1, x_2, \ldots, x_{k_x,t-1}, x_{1,t-s}, \ldots, x_{k_x,t-s})' \]
  - If the NOCURRENTX option is not specified,
    \[ z_t^{(\text{XL})} = (x_1, x_2, \ldots, x_{k_x,t}, x_{1,t-1}, x_{2,t-1}, \ldots, x_{k_x,t-1}, x_{1,t-s}, \ldots, x_{k_x,t-s})' \]

The statement

```
model y = x1 x2 x3 / type=reg nstate=2;
```
defines the regression for state $i, i = 1, \ldots, K$, as

$$y_t = b_{i,1} + b_{i,2} x_{1t} + b_{i,3} x_{2t} + b_{i,4} x_{3t} + \epsilon_t$$

$$= (b_{i,1} b_{i,2} b_{i,3} b_{i,4}) z_t + \epsilon_t$$

where $\epsilon_t \sim N(0, \Sigma_i)$, $z_t = (1 \times 1_t \times 2_t \times 3_t)'$, and $k_z = 4$.

In the following statement, $z_t = (1 \times d_t^{(1)} \times d_t^{(2)} \times d_t^{(3)} t \times t^2 \times 1_t \times 2_t \times 3_t \times x_{1t-1} \times x_{2t-1} \times x_{3t-1})'$, where $sd_t^{(i)}, i = 1, 2, 3$, denote the values of seasonal dummies at time $t$ and where $k_z = 12$:

```plaintext
model y = x1 x2 x3 / trend=quad nseason=4 xlag=1 type=reg nstate=2;
```

In the following statement, $z_t = (x_{1t-1} \times 2_{t-1} \times 3_{t-1})'$ and $k_z = 3$:

```plaintext
model y = x1 x2 x3 / noint xlag=1 nocurrentx type=reg nstate=2;
```

The number of dependent variables, $k_y$, might be greater than 1; in that case, $B_i, i = 1, \ldots, K$, is the $k_y \times k_z$ matrix, and $\Sigma_i, i = 1, \ldots, K$, 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 $z_t$ (`CONST`), $d_t$ (`SD`), $z_t^{(LTREND)}$, $z_t^{(QTREND)}$, and $z_t^{(XL)}$, respectively; the `COV` function refers to the covariance parameters $\Sigma = (\Sigma_1 \cdots \Sigma_K)'$, 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 575. 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,

$$Y_t | \pi_t, S_t \sim N(B_{S_t}, \Sigma_{S_t})$$

where $B_{S_t}$ and $\Sigma_{S_t}$ are mean and covariance parameters at state $S_t, S_t = 1, \ldots, K$. Hence, the initial state probability vector (ISPV) $\pi_t$, the transition probability matrix (TPM) $A$, and the observation parameter $B$ ($\equiv \{B_i, \Sigma_i\}_{i=1, \ldots, K}$, which consists of mean and covariance parameters) together specify the RS AR; that is, the parameter $\theta$ of the RS AR is $\{\pi, A, B\}$. 
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,

$$z_t = \left( (z_t^{(\text{CONST})})' (z_{t-1}^{(\text{SD})})' (z_{t-1}^{(\text{LTREND})})' (z_{t-1}^{(\text{QTREND})})' (z_t^{(\text{XL})})' (z_t^{(\text{AR})})' \right)'$$

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, $z_t^{(\text{SD})} = (s_{d_1}^{(1)} \ldots s_{d_{n_s-1}}^{(n_s-1)})'$, where $s_{d_i}^{(j)}$, $i = 1, \ldots, n_s$, 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,
    $$z_t^{(\text{XL})} = (x_{1,t-1} \ldots x_{k_x,t-1} \ldots x_{1,t-s} \ldots x_{k_x,t-s})'$$
  - If the NOCURRENTX option is not specified,
    $$z_t^{(\text{XL})} = (x_{1,t} \ldots x_{k_x,t} x_{1,t-1} \ldots x_{k_x,t-1} \ldots x_{1,t-s} \ldots x_{k_x,t-s})'$$
- $z_t^{(\text{AR})}$ 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:
  $$z_t^{(\text{AR})} = (y_{1,t-1} \ldots y_{k_y,t-1} \ldots y_{1,t-p} \ldots y_{k_y,t-p})'$$

The statement

```plaintext
model y = x1 x2 x3 / ylag=2 type=ar nstate=2;
```

defines the autoregression for state $i, i = 1, \ldots, K$, as

$$y_t = b_{i,1} + b_{i,2} x_{1t} + b_{i,3} x_{2t} + b_{i,4} x_{3t} + b_{i,5} y_{t-1} + b_{i,6} y_{t-2} + \varepsilon_t$$

$$= (b_{i,1} b_{i,2} b_{i,3} b_{i,4} b_{i,5} b_{i,6}) z_t + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \Sigma_t), z_t = (1 x_{1t} x_{2t} x_{3t} y_{t-1} y_{t-2})'$, and $k_z = 6$.

In the following statement, $z_t = (1 s_{d_1}^{(1)} s_{d_2}^{(2)} s_{d_3}^{(3)} t^2 x_{1t} x_{2t} x_{3t} x_{1t-1} x_{2t-1} x_{3t-1} y_{t-1} y_{t-2})'$, where $s_{d_i}^{(j)}, i = 1, 2, 3$, denote the values of seasonal dummies at time $t$ and where $k_z = 14$:
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

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(Y_1, \ldots, Y_T; \theta))$$

where $L_T(\cdot; \theta)$ is the likelihood of all observations for a particular parameter $\theta$. 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

$$\hat{\theta} = \arg \max_{\theta} \log (L_T(Y_1, \ldots, Y_T; \theta)) + \log (p(\theta))$$

where $p(\theta)$ is the prior distribution function of parameter $\theta$. 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), $\pi$, is the Dirichlet distribution, 

$$\pi \sim \text{Dir}(\alpha)$$

where the hyperparameter $\alpha$ 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 $x = (x_1 \cdots x_K)'$, $x_i > 0$, $i = 1, \ldots, K$, and $\sum_{i=1}^{K} x_i = 1$; $\alpha = (\alpha_1 \cdots \alpha_K)'$ and $\alpha_i > 0$, $i = 1, \ldots, K$; $B(\alpha)$ 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), $A$, is the Dirichlet distribution,

$$a_i' \sim \text{Dir}(\alpha^{(i)}), \ i = 1, \ldots, K$$

where $a_i$ is the $i$th row of the TPM $A$, $\alpha^{(i)}$ 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,

$$A \sim \text{Dir}(\tilde{\alpha})$$

where the hyperparameter $\tilde{\alpha} = (\alpha^{(1)} \cdots \alpha^{(K)})'$, which is the $K \times K$ matrix that results from stacking the transpose of $\alpha^{(i)}$, $i = 1, \ldots, K$.

**Prior Distribution of the Mixture Component Probabilities**

The prior distribution of each row of the mixture component probabilities (MCP), $W$, is the Dirichlet distribution,

$$w_i' \sim \text{Dir}(\alpha^{(i)}), \ i = 1, \ldots, K$$

where $w_i$ is the $i$th row of the MCP $W$, $\alpha^{(i)}$ 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,

$$W \sim \text{Dir}(\tilde{\alpha})$$
where the hyperparameter $\tilde{A} = (\alpha^{(1)} \cdots \alpha^{(K)})'$, which is the $K \times M$ matrix that results from stacking the transpose of $\alpha^{(i)}$, $i = 1, \ldots, K$.

**Prior Distribution of the Observation Parameters**

**Gaussian HMM** In the Gaussian HMM, the observation parameters are $\{\mu_i, \Sigma_i\}, i = 1, \ldots, K$, where $\mu_i$ and $\Sigma_i$ are the mean and covariance parameters for state $i$. The prior distribution of $\mu_i$ and $\Sigma_i$ is the normal-inverse-Wishart (NIW) distribution,

$$\mu_i, \Sigma_i \sim \text{NIW}(\tilde{\mu}_i, \tilde{\Sigma}_i, \tilde{\kappa}_i, \tilde{\upsilon}_i), i = 1, \ldots, K$$

which is equivalent to

$$\mu_i | \Sigma_i \sim N(\tilde{\mu}_i, \tilde{\kappa}_i^{-1} \Sigma_i) \quad \Sigma_i \sim \text{IW}(\tilde{\Sigma}_i, \tilde{\upsilon}_i)$$

where the hyperparameter $\tilde{\mu}_i$ is a $k \times 1$ vector; the hyperparameter $\tilde{\Sigma}_i$ is a $k \times k$ symmetric positive definite matrix; the hyperparameter $\tilde{\kappa}_i$ is a positive scalar; the hyperparameter $\tilde{\upsilon}_i$ is a positive scalar, $\tilde{\upsilon}_i > k_y + 3$; $K$ is the number of states; and $k_y$ is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

$$f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{\upsilon}) = f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\upsilon})$$

$$f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1} \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1} \Sigma|}} \exp\left(-\frac{1}{2} (\mu - \tilde{\mu})' (\tilde{\kappa}^{-1} \Sigma)^{-1} (\mu - \tilde{\mu})\right)$$

$$f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{\upsilon}) = \frac{|\tilde{\Sigma}|^{\frac{k}{2}} |\Sigma|^{-\frac{k+k+1}{2}}}{2^{\frac{k+1}{2}} \Gamma_k(\frac{k}{2})} \exp\left(-\frac{1}{2} \text{tr}(\tilde{\Sigma}^{-1} \Sigma)\right)$$

where the parameter $\mu$ is a $k \times 1$ vector; the parameter $\Sigma$ 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 $\tilde{\upsilon}$ is a positive scalar, $\tilde{\upsilon} > k + 3$. The $\Gamma_k(.)$ is the multivariate gamma function, which is defined as

$$\Gamma_k(x) = \pi^{k(k-1)/4} \prod_{j=1}^{k} \Gamma\left(x + \frac{1-j}{2}\right)$$

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

$$\tilde{\mu} = (\tilde{\mu}_1 \cdots \tilde{\mu}_K)'$$

$$\tilde{\Sigma} = (\tilde{\Sigma}_1 \cdots \tilde{\Sigma}_K)'$$

$$\tilde{\kappa} = (\tilde{\kappa}_1 \cdots \tilde{\kappa}_K)'$$

$$\tilde{\upsilon} = (\tilde{\upsilon}_1 \cdots \tilde{\upsilon}_K)'$$

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, $\tilde{\upsilon}$ 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 \( \{ \mu_{ij}, \Sigma_{ij} \}, i = 1, \ldots, K, j = 1, \ldots, M \), where $M$ is the number of components at a state and $K$ is the number of states.

The prior distribution of $\mu_{ij}$ and $\Sigma_{ij}$ is the normal-inverse-Wishart (NIW) distribution,

\[
\mu_{ij}, \Sigma_{ij} \sim \text{NIW}(\tilde{\mu}_{ij}, \tilde{\Sigma}_{ij}, \tilde{\kappa}_{ij}, \tilde{v}_{ij}), i = 1, \ldots, K, j = 1, \ldots, M
\]

which is equivalent to

\[
\mu_{ij}|\Sigma_{ij} \sim \mathcal{N}(\tilde{\mu}_{ij}, \tilde{\kappa}_{ij}^{-1}\Sigma_{ij}) \\
\Sigma_{ij} \sim \text{IW}(\tilde{\Sigma}_{ij}, \tilde{v}_{ij})
\]

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 $\tilde{v}_{ij}$ is a positive scalar, $\tilde{v}_{ij} > k_y + 3$; and $k_y$ is the number of dependent variables.

The probability density function (PDF) of the NIW distribution is

\[
f_{\text{NIW}}(\mu, \Sigma, \tilde{\mu}, \tilde{\Sigma}, \tilde{\kappa}, \tilde{v}) = f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{v})
\]

\[
f_N(\mu, \tilde{\mu}, \tilde{\kappa}^{-1}\Sigma) = \frac{1}{\sqrt{(2\pi)^k |\tilde{\kappa}^{-1}\Sigma|}} \exp\left(-\frac{1}{2}(\mu - \tilde{\mu})'(\tilde{\kappa}^{-1}\Sigma)^{-1}(\mu - \tilde{\mu})\right)
\]

\[
f_{\text{IW}}(\Sigma, \tilde{\Sigma}, \tilde{v}) = \frac{|\tilde{\Sigma}|^{\tilde{v}/2}}{2^{\tilde{v}k/2} \Gamma_k(\tilde{v}/2)} |\Sigma|^{-\tilde{v}+k+1}/2 \exp\left(-\frac{1}{2} \text{tr}(\tilde{\Sigma}\Sigma^{-1})\right)
\]

where the parameter $\mu$ is a $k \times 1$ vector; the parameter $\Sigma$ 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 $\tilde{v}$ is a positive scalar, $\tilde{v} > k + 3$. The $\Gamma_k(.)$ is the multivariate gamma function defined as

\[
\Gamma_k(x) = \pi^{k(k-1)/4} \prod_{j=1}^{k} \Gamma(x + 1 - j/2)
\]

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

\[
\tilde{\mu} = (\tilde{\mu}_{11} \cdots \tilde{\mu}_{1M} \tilde{\mu}_{21} \cdots \tilde{\mu}_{2M} \cdots \tilde{\mu}_{KM} \cdots \tilde{\mu}_{KM})'
\]

\[
\tilde{\Sigma} = (\tilde{\Sigma}_{11} \cdots \tilde{\Sigma}_{1M} \tilde{\Sigma}_{21} \cdots \tilde{\Sigma}_{2M} \cdots \tilde{\Sigma}_{KM} \cdots \tilde{\Sigma}_{KM})'
\]

\[
\tilde{\kappa} = (\tilde{\kappa}_{11} \cdots \tilde{\kappa}_{1M} \tilde{\kappa}_{21} \cdots \tilde{\kappa}_{2M} \cdots \tilde{\kappa}_{KM} \cdots \tilde{\kappa}_{KM})'
\]

\[
\tilde{v} = (\tilde{v}_{11} \cdots \tilde{v}_{1M} \tilde{v}_{21} \cdots \tilde{v}_{2M} \cdots \tilde{v}_{KM} \cdots \tilde{v}_{KM})'
\]

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, $\tilde{v}$ 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 \( \{B_i, \Sigma_i\}, i = 1, \ldots, K \), where \( B_i \) and \( \Sigma_i \) are the mean and covariance parameters for state \( i \). Because \( B_i z_t = (z_i' \otimes I_{k_y}) \text{vec}(B_i) = (z_i' \otimes I_{k_y}) \beta_i \), where vec is the column stacking operator, the observation parameters in this section are denoted by \( \{\beta_i, \Sigma_i\}, i = 1, \ldots, K \). The prior distribution of \( \beta_i \) and \( \Sigma_i \) is the normal-inverse-Wishart (NIW) distribution,

\[
\beta_i, \Sigma_i \sim \text{NIW}(\mu_i, \Sigma_i, k_i, \nu_i), i = 1, \ldots, K
\]

which is equivalent to

\[
\beta_i | \Sigma_i \sim N(\mu_i, k_i^{-1} \otimes \Sigma_i) \\
\Sigma_i \sim \text{IW}(\hat{\Sigma}_i, \hat{\nu}_i)
\]

where the hyperparameter \( \hat{\mu}_i \) is a \( k_y k_z \times 1 \) vector; the hyperparameter \( \hat{\Sigma}_i \) is a \( k_y \times k_y \) symmetric positive definite matrix; the hyperparameter \( k_i \) is a \( k_z \times k_z \) symmetric positive definite matrix; the hyperparameter \( \hat{\nu}_i \) is a positive scalar, \( \hat{\nu}_i > k_y + 3 \); \( K \) 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 580 and “Regime-Switching Autoregression Model” on page 581.

The probability density function (PDF) of the NIW distribution is

\[
f_{\text{NIW}}(\beta, \Sigma, \mu, \tilde{\Sigma}, \tilde{k}, \tilde{\nu}) = f_N(\beta, \hat{\mu}, \hat{k}^{-1} \otimes \Sigma) f_{\text{IW}}(\Sigma, \hat{\Sigma}, \hat{\nu}) \\
f_N(\beta, \hat{\mu}, \hat{k}^{-1} \otimes \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\hat{k}^{-1} \otimes \Sigma|}} \exp\left(-\frac{1}{2} (\beta - \hat{\mu})' (\hat{k}^{-1} \otimes \Sigma)^{-1} (\beta - \hat{\mu})\right) \\
f_{\text{IW}}(\Sigma, \hat{\Sigma}, \hat{\nu}) = \frac{|\hat{\Sigma}|^{\frac{\nu}{2}} |\Sigma|^{-\frac{\nu+k}{2}}}{2^{-\frac{\nu+k}{2}} \Gamma_k (\frac{\nu}{2})} \exp\left(-\frac{1}{2} \text{tr}(\hat{\Sigma}^{-1} \Sigma^{-1})\right)
\]

where the parameter \( \beta \) is a \( k_1 k_2 \times 1 \) vector; the parameter \( \Sigma \) is a \( k_1 \times k_1 \) symmetric positive matrix; the hyperparameter \( \hat{\mu} \) is a \( k_1 k_2 \times 1 \) vector; the hyperparameter \( \hat{\Sigma} \) is a \( k_1 \times k_1 \) symmetric positive definite matrix; the hyperparameter \( \hat{k} \) is a \( k_2 \times k_2 \) symmetric positive definite matrix; and the hyperparameter \( \hat{\nu} \) is a positive scalar, where \( \hat{\nu} > k_1 + 3 \). The \( \Gamma_k(.) \) is the multivariate gamma function defined as

\[
\Gamma_k(x) = \pi^{k(k-1)/4} \prod_{j=1}^{k} \Gamma(x + \frac{1}{2})
\]

In the PRIOR statement, the arguments for the NIW function are constructed by stacking the hyperparameters for each state,

\[
\hat{\mu} = (\hat{\mu}_1 \ldots \hat{\mu}_K)' \\
\hat{\Sigma} = (\hat{\Sigma}_1 \ldots \hat{\Sigma}_K)' \\
\hat{k} = (\hat{k}_1 \ldots \hat{k}_K)' \\
\hat{\nu} = (\hat{\nu}_1 \ldots \hat{\nu}_K)'
\]

where \( \hat{\mu} \) is a \( K \times k_y k_z \) matrix, \( \hat{\Sigma} \) is a \( k_y K \times k_y \) matrix, \( \hat{k} \) is a \( k_z K \times k_z \) matrix, \( \hat{\nu} \) 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 two types of optimization algorithms: the interior point algorithm and the active set algorithm. For more information about these algorithms and their related options, see the chapter “The Nonlinear Programming Solver” in SAS/OR User’s Guide: Mathematical Programming.

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 \( \text{DIAG}(A) \), 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 \( 2 \times 2 \) matrix \( \{1 2, 3 4\} \) or the \( 1 \times 3 \) 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:

\[
\text{initial} \ {-0.1 -0.2, -0.3 -0.4} = \text{MU};
\]

You can specify the preceding example by enclosing the first matrix constant in parentheses, as follows:

\[
\text{initial} \ \{( -0.1 -0.2, -0.3 -0.4)\} = \text{MU};
\]

Operators

Operators define the operations on operands. Table 13.2 lists all built-in operators that the INITIAL statement supports.

<table>
<thead>
<tr>
<th>Operator Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>=</td>
<td>Comparison, equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>Comparison, less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Comparison, not greater than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Comparison, greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Comparison, not less than</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>//</td>
<td>Concatenation, vertical</td>
</tr>
<tr>
<td>@</td>
<td>Direct product</td>
</tr>
</tbody>
</table>

![Table 13.2 Operators]

<table>
<thead>
<tr>
<th>Operator Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>=</td>
<td>Comparison, equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>Comparison, less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Comparison, not greater than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Comparison, greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Comparison, not less than</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>//</td>
<td>Concatenation, vertical</td>
</tr>
<tr>
<td>@</td>
<td>Direct product</td>
</tr>
</tbody>
</table>
Table 13.2  continued

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Distribution</td>
<td>Links the distribution function to the matrix elements</td>
</tr>
<tr>
<td>:</td>
<td>Index creation</td>
<td>Creates an index vector</td>
</tr>
<tr>
<td>#</td>
<td>Multiplication, elementwise</td>
<td>Performs elementwise multiplication</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication, matrix</td>
<td>Performs matrix multiplication</td>
</tr>
<tr>
<td>-</td>
<td>Sign reverse</td>
<td>Reverses the signs of elements</td>
</tr>
<tr>
<td>[]</td>
<td>Subscripts</td>
<td>Selects submatrices</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Subtracts corresponding matrix elements</td>
</tr>
<tr>
<td>`</td>
<td>Transpose</td>
<td>Transposes a matrix</td>
</tr>
</tbody>
</table>

For more information about each operator, see the section “Details of Operators” on page 593.

Table 13.3 shows the precedence of matrix operators.

Table 13.3  Precedence of Operators

<table>
<thead>
<tr>
<th>Priority Group</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>I (highest)</td>
<td>[ ] (subscripts) ` (transpose)</td>
</tr>
<tr>
<td>II</td>
<td>- (sign reverse)</td>
</tr>
<tr>
<td>III</td>
<td>* # @</td>
</tr>
<tr>
<td>IV</td>
<td>- (subtraction) +</td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VI (lowest)</td>
<td>= &lt; &lt;= &gt; &gt;= ~</td>
</tr>
</tbody>
</table>

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 $\text{MU}(K, I)$ and $\text{SIGMA}(K, I, J)$. The other type does not; examples are $\text{I}(n)$ and $\text{DIAG}(A)$.

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR(k, l, i, j)</td>
<td>Autoregressive parameter of the lag l of the jth dependent variable, ( y_{jt-l} ), to the ith dependent variable at time t, ( y_{it} ), for state k in regime-switching autoregression</td>
</tr>
<tr>
<td>CONST(k, i)</td>
<td>Intercept parameter of the ith 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</td>
</tr>
<tr>
<td>COV(k, i, j)</td>
<td>Covariance of innovations parameter between the ith and jth error processes for state k in regime-switching regression or regime-switching autoregression, or the alias of SIGMA for Gaussian HMM or GM HMM</td>
</tr>
<tr>
<td>ISPV(i)</td>
<td>The ith element in the initial state probability vector</td>
</tr>
<tr>
<td>LTREND(k, i)</td>
<td>Linear trend parameter of the ith 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</td>
</tr>
<tr>
<td>MCP(i, j)</td>
<td>The (i, j)th element in the mixture component probabilities in the GM HMM</td>
</tr>
<tr>
<td>MU(k, i)</td>
<td>Mean parameter of the ith time series, ( y_{it} ), for state k in the Gaussian HMM, or alias for CONST for regime-switching regression or regime-switching autoregression</td>
</tr>
<tr>
<td>MU(k, c, i)</td>
<td>Mean parameter of the ith time series, ( y_{it} ), for the cth component at state k in the GM HMM</td>
</tr>
<tr>
<td>QTREND(k, i)</td>
<td>Quadratic trend parameter of the ith time series, ( y_{it} ), for state k in regime-switching regression or regime-switching autoregression when TREND=QUAD is specified in the MODEL statement</td>
</tr>
<tr>
<td>SD(k, i, j)</td>
<td>Alias of SDUMMY</td>
</tr>
<tr>
<td>SDUMMY(k, i, j)</td>
<td>The jth seasonal dummy of the ith time series at time t, ( y_{it} ), for state k in regime-switching regression or regime-switching autoregression, where ( j = 1, \ldots, (nseason-1) ), where nseason is the value of the NSEASON= option in the MODEL statement</td>
</tr>
<tr>
<td>SIGMA(k, i, j)</td>
<td>Covariance parameter between the ith and jth time series, ( y_{it} ) and ( y_{jt} ), for state k in the Gaussian HMM, or the alias of COV for regime-switching regression or regime-switching autoregression</td>
</tr>
<tr>
<td>SIGMA(k, c, i, j)</td>
<td>Covariance parameter between the ith and jth time series, ( y_{it} ) and ( y_{jt} ), for cth component at state k in the GM HMM</td>
</tr>
<tr>
<td>TPM(i, j)</td>
<td>The (i, j)th element in the transition probability matrix</td>
</tr>
<tr>
<td>XL(k, l, i, j)</td>
<td>Exogenous parameter of the lag l of the jth exogenous variable, ( x_{jt-l} ), to the ith dependent variable at time t, ( y_{it} ), for state k in regime-switching regression or regime-switching autoregression</td>
</tr>
</tbody>
</table>

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 \( \text{ISPV} \) 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, \( \text{func}_1 \), that has one vector argument \( I \), where \( I = (i_1 i_2 \ldots i_n)' \), returns a vector 
\[ R = (r_1 r_2 \ldots r_n)' \], where \( r_k = \text{func}_1(i_k) \), \( k = 1, \ldots, n \). \( \text{ispv} \) is a type of \( \text{func}_1 \).

• A function, \( \text{func}_2 \), that has two vector arguments \( I \) and \( J \), where \( I = (i_1 i_2 \ldots i_n)' \) and \( J = (j_1 j_2 \ldots j_n)' \), returns a matrix,

\[
R = \begin{pmatrix}
    r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\
    r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{n_J,1} & r_{n_J,2} & \cdots & r_{n_J,n_J}
\end{pmatrix}
\]

where \( r_{k,m} = \text{func}_2(i_k, j_m) \), \( k = 1, \ldots, n_I, m = 1, \ldots, n_J \). \( \text{const} \), \( \text{ltrend} \), \( \text{mu} \), \( \text{qtrend} \), and \( \text{tpm} \) are types of \( \text{func}_2 \).

• A function, \( \text{func}_3 \), that has \( N \) vector arguments \( I_1, I_2, \ldots, I_N \), where \( N > 2 \) and \( N \) is odd and where \( I_n = (i_{n,1} i_{n,2} \ldots i_{n,j_n})' \), \( n = 1, \ldots, N \), returns a block matrix,

\[
R = \begin{pmatrix}
    B_1 \\
    B_2 \\
    \vdots \\
    B_{j_n}
\end{pmatrix}
\]

where \( B_k = \text{func}_3(i_{1,k}, I_2, \ldots, I_N) \), \( k = 1, \ldots, j_1 \), and \( B_k \) is constructed in the same way as \( \text{func}_{n-1}(I_2, \ldots, I_N) \).

For example, a function, \( \text{func}_3 \), that has three vector arguments \( K, I, \) and \( J \), where \( K = (k_1 k_2 \ldots k_n)' \), \( I = (i_1 i_2 \ldots i_n)' \), and \( J = (j_1 j_2 \ldots j_n)' \), returns a matrix,

\[
R = \begin{pmatrix}
    r_{1,1} & r_{1,2} & \cdots & r_{1,n_J} \\
    r_{2,1} & r_{2,2} & \cdots & r_{2,n_J} \\
    \vdots & \vdots & \ddots & \vdots \\
    r_{n_J,1} & r_{n_J,2} & \cdots & r_{n_J,n_J}
\end{pmatrix}
\]

where \( r_{u,v} = \text{func}_3(k_{d_1(u,n_I)}, i_{d_2(u,n_I)}, j_v) \), \( u = 1, \ldots, n_J, v = 1, \ldots, n_J \), and \( d_1(m,n) \) is the integer such that \( d_1(m,n)n \geq m > (d_1(m,n) - 1)n \) and \( d_2(m,n) = m - (d_1(m,n) - 1)n \). \( \text{cov} \) and \( \text{mu} \) for GM HMM and \( \text{sigma} \) for Gaussian HMM are types of \( \text{func}_3 \).

• A function, \( \text{func}_4 \), that has \( N \) vector arguments \( I_1, I_2, \ldots, I_N \), where \( N > 2 \) and \( N \) is even and where \( I_n = (i_{n,1} i_{n,2} \ldots i_{n,j_n})' \), \( n = 1, \ldots, N \), 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 \( B_{k,l} = \text{func}_4(i_{1,k}, i_{2,l}, I_3, \ldots, I_N) \), \( k = 1, \ldots, j_1, l = 1, \ldots, j_2 \), and \( B_{k,l} \) is constructed in the same way as \( \text{func}_{n-2}(I_3, \ldots, I_N) \).
For example, a function, \( \text{FUNC}_4 \), that has four vector arguments \( K, L, I, \) and \( J \), where \( K = (k_1 k_2 \ldots k_{n_K})' \), \( L = (l_1 l_2 \ldots l_{n_L})' \), \( I = (i_1 i_2 \ldots i_{n_I})' \), and \( J = (j_1 j_2 \ldots j_{n_J})' \), returns a matrix,

\[
R = \begin{pmatrix}
  r_{1,1} & r_{1,2} & \cdots & r_{1,n_{L \times n_J}} \\
  r_{2,1} & r_{2,2} & \cdots & r_{2,n_{L \times n_J}} \\
  \vdots & \vdots & \ddots & \vdots \\
  r_{n_{K \times n_I},1} & r_{n_{K \times n_I},2} & \cdots & r_{n_{K \times n_I},n_{L \times n_J}}
\end{pmatrix}
\]

where \( r_{u,v} = \text{FUNC}_4(k_{d_1(u,n_{n_I})}, l_{d_1(v,n_{n_J})}, i_{d_2(u,n_{n_I})}, j_{d_2(v,n_{n_J})}) \), \( u = 1, \ldots, n_{K \times n_I} \), \( v = 1, \ldots, n_{L \times n_J} \), where \( d_1(m,n) \) is the integer such that \( d_1(m,n) n > m > (d_1(m,n) - 1)n \) and \( d_2(m,n) = m - (d_1(m,n) - 1)n \). \( \text{AR} \) and \( \text{SIGMA} \) for GM HMM and \( \text{XL} \) are types of \( \text{FUNC}_4 \).

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:

```plaintext
model y1 y2 / type=gaussian nstate=3;
```

In order to initialize the second row of \( \text{TPM} \) to be \( \{0.1 0.7 0.2\} \), you can use the following statement:

```plaintext
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:

```plaintext
initial tpm(2, ) = {0.1 0.7 0.2};
```

To get all coefficients of the covariance matrix for state 2, you can use \( \text{SIGMA}(2, \{1 2\}, \{1 2\}) \), or \( \text{SIGMA}(2, , ) \), or \( \text{SIGMA}(2) \). To get all coefficients of exogenous variables on dependent variables, you can use \( \text{SIGMA}(\{1 2 3\}, \{1 2\}, \{1 2\}) \), or \( \text{SIGMA}(, , ) \), or \( \text{SIGMA}() \), or even just \( \text{SIGMA} \).

Another type of function does not refer to parameters but generates useful matrices. Table 13.5 lists all built-in functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAG(A)</td>
<td>Creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix</td>
</tr>
<tr>
<td>I(n)</td>
<td>Creates an ( n \times n ) identity matrix</td>
</tr>
<tr>
<td>J(m,n,elem)</td>
<td>Creates an ( m \times n ) matrix whose elements are all equal to ( \text{elem} )</td>
</tr>
<tr>
<td>SHAPE(A,m,n)</td>
<td>Creates an ( m \times n ) matrix whose elements are from matrix ( A )</td>
</tr>
</tbody>
</table>

For more information about each function in Table 13.5, see the section “Details of Functions” on page 597.
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 \( i = 1, \ldots, n \), \( j = 1, \ldots, p \), where \( \text{matrix1} \) and \( \text{matrix2} \) are both \( n \times p \) matrices.

For example, \( \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \) results in \( \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix} \).

- `matrix + scalar` adds the scalar value to each element of the matrix. For example, you can obtain \( \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix} \) from \( \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + 1 \).

- `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 \( \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix} \) from \( \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \).
  - 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 \( \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix} \) from \( \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \).

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

\[ \text{initial} \, \text{SIGMA}(1, \{1,2\}, \{1,2\}) = \text{SIGMA}(2, \{3,4\}, \{3,4\}); \]

is transformed into the following equivalent INITIAL statement with scalar parameters:

\[ \text{initial} \, \text{SIGMA}(1,1,1) = \text{SIGMA}(2,3,3), \]
\[ \text{SIGMA}(1,1,2) = \text{SIGMA}(2,3,4), \]
\[ \text{SIGMA}(1,2,1) = \text{SIGMA}(2,4,3), \]
\[ \text{SIGMA}(1,2,2) = \text{SIGMA}(2,4,4); \]
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 $\text{matrix1}$ and $\text{matrix2}$.

```
\text{matrix1} \| \text{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 $\text{matrix1}$ plus the number of columns in $\text{matrix2}$.

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 $\text{matrix1}$ and $\text{matrix2}$.

```
\text{matrix1} // \text{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 $\text{matrix1}$ plus the number of rows in $\text{matrix2}$.

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 $\text{matrix1}$ and $\text{matrix2}$.

```
\text{matrix1} @ \text{matrix2}
```

For matrices $A$ and $B$, the direct product is denoted by $A \otimes B$. The number of rows in the new matrix equals the product of the number of rows in $\text{matrix1}$ and the number of rows in $\text{matrix2}$; the number of columns in the new matrix equals the product of the number of columns in $\text{matrix1}$ and the number of columns in $\text{matrix2}$.
Specifically, if $A$ is an $n \times p$ matrix and $B$ is an $m \times q$ matrix, then the Kronecker product $A \otimes B$ is the following $nm \times pq$ block matrix:

$$A \otimes B = \begin{bmatrix}
A_{11}B & \cdots & A_{1p}B \\
\vdots & \ddots & \vdots \\
A_{n1}B & \cdots & A_{np}B
\end{bmatrix}$$

For example, $\{1 \ 2, \ 3 \ 4\} \otimes \{0 \ 2\}$ returns $\{0 \ 2 \ 0 \ 4, \ 0 \ 6 \ 0 \ 8\}$, and $\{0 \ 2\} \otimes \{1 \ 2, \ 3 \ 4\}$ returns $\{0 \ 0 \ 2 \ 4, \ 0 \ 0 \ 6 \ 8\}$. Note that the direct product of two matrices is not commutative.

**Distribution Operator:** $\sim$

The distribution operator links the matrix to its distribution function.

`matrix \sim distribution-function`

This operator is used in the PRIOR statement. For more information, see the section “PRIOR Statement” on page 572.

For example, `ISPV \sim DIR(\{1,1,1\})` means that the initial state probability vector (ISPV) follows a Dirichlet distribution with an argument of a $3 \times 1$ 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.

\[
\text{matrix1} \times \text{matrix2}
\]

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 \( A \times B \) 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, \( \{1 2, 3 4\} \times \{1, 2\} \) 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 \( \text{matrix} \). The sign reversal operator is also called the unary minus operator.

\[
- \text{matrix}
\]

For example, \( -\{-1 7 6, 2 0 -8\} \) returns \( \{-7 -6, -2 0 8\} \).

**Subscripts:** []

Subscripts are used with matrices to select submatrices, where \( \text{rows}, \text{columns} \), and \( \text{elements} \) 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.

\[
\text{matrix}[\text{rows, columns}]
\]

\[
\text{matrix}[\text{elements}]
\]

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.

\[
\text{matrix1} - \text{matrix2}
\]

computes a new matrix whose elements are formed by subtracting the corresponding elements of \( \text{matrix2} \) from those of \( \text{matrix1} \).

For example, \( \{1 2 3, 4 5 6\} - \{1 1 1, 1 1 1\} \) returns \( \{0 1 2, 3 4 5\} \).

\[
\text{matrix} - \text{scalar}
\]

subtracts the \( \text{scalar} \) value from each element of the \( \text{matrix} \).

For example, \( \{1 2 3, 4 5 6\} - 1 \) returns \( \{0 1 2, 3 4 5\} \).

\[
\text{matrix} - \text{vector}
\]

subtracts the \( \text{vector} \) from each element of the \( \text{matrix} \):
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 \( \text{matrix} \), producing the transpose of \( \text{matrix} \).

\[ \text{matrix}' \]

If \( v \) is the value in the \( i \)th row and \( j \)th column of \( \text{matrix} \), then the transpose of \( \text{matrix} \) contains \( v \) in the \( j \)th row and \( i \)th column. If \( \text{matrix} \) 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 \( \text{DIAG} \) function creates a diagonal matrix from a vector or extracts the diagonal elements of a matrix.

\[ \text{DIAG(\text{matrix})} \]

The \( \text{matrix} \) argument can be either a square matrix or a vector:

- If \( \text{matrix} \) is a square matrix, the \( \text{DIAG} \) function creates a vector from the diagonal elements of the matrix.
  
  For example, \( \text{DIAG} \{\text{1 \ 2 \ 3, \ 4 \ 5 \ 6, \ 7 \ 8 \ 9}\} \) returns \( \{1, \ 5, \ 9\} \).

- If \( \text{matrix} \) is a vector, the \( \text{DIAG} \) function creates a matrix whose diagonal elements are the values in the vector. All off-diagonal elements are zeros.

  For example, \( \text{DIAG} \{1 \ 5 \ 9\} \) or \( \text{DIAG} \{(1, \ 5, \ 9)\} \) returns \( \{1 \ 0 \ 0, \ 0 \ 5 \ 0, \ 0 \ 0 \ 9\} \).

You can call the \( \text{DIAG} \) function repeatedly. For example, \( \text{DIAG}(\text{DIAG}(\{1 \ 2 \ 3, \ 4 \ 5 \ 6, \ 7 \ 8 \ 9\})) \) returns \( \{1 \ 0 \ 0, \ 0 \ 5 \ 0, \ 0 \ 0 \ 9\} \).

**I Function**
The \( \text{I} \) function creates an identity matrix that contains \( \text{dim} \) rows and columns.

\[ \text{I(\text{dim})} \]

The diagonal elements of an identity matrix are ones; all other elements are zeros. The value of \( \text{dim} \) must be an integer greater than or equal to 1. Noninteger operands are rounded to the nearest integer.

For example, \( \text{I(3)} \) 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 \( \{1 2 3 4\} \).

**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

\[
\begin{align*}
\text{AIC} &= -2 \ell + 2r \\
\text{AICC} &= -2 \ell + 2r T / (T - r - 1) \\
\text{BIC} &= -2 \ell + r \log(T) \\
\text{HQC} &= -2 \ell + 2r \log(\log(T))
\end{align*}
\]

where \( \ell \) 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 OUTSTAT= data table from the PROC HMM statement, the OUT= data tables from the DECODE, ESTIMATE, EVALUATE, FILTER, FORECAST, and SMOOTH statements, and the OUTMODEL= data table from the SCORE statement. The column information for each table is described in the following sections.

**OUTSTAT= Data Table Generated from the PROC HMM Statement**

The output data table contains the following variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelIndex</td>
<td>model index, if multiple models are estimated or scored</td>
</tr>
<tr>
<td>NState</td>
<td>number of states, if you specify a range of numbers of states in the NSTATE=n1 : n2 option</td>
</tr>
<tr>
<td>YLag</td>
<td>order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=n1 : n2 option</td>
</tr>
<tr>
<td>LogLikelihood</td>
<td>log likelihood</td>
</tr>
<tr>
<td>AIC</td>
<td>Akaike’s information criterion</td>
</tr>
<tr>
<td>AICC</td>
<td>corrected Akaike’s information criterion</td>
</tr>
<tr>
<td>BIC</td>
<td>Bayesian information criterion (also referred to as the Schwarz Bayesian criterion, SBC)</td>
</tr>
<tr>
<td>HQC</td>
<td>Hannan-Quinn criterion</td>
</tr>
</tbody>
</table>
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=n₁ : n₂ option
- **YLag**: order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=n₁ : n₂ option
- **Section ID Variable**: values of the variable specified in the SECTION= option in the ID statement
- **Time ID Variable**: values of the variable 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=n₁ : n₂ option
- **YLag**: order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=n₁ : n₂ option
- **Index**: index of the parameter
- **Type**: type of values in the Stateₖ variables: EST for estimate and STD for standard error
- **Parameter**: parameter name
- **Stateₖ**: value for state k, k = 1, . . . , K, where K is the number of states if you specify the NSTATE=K option; otherwise, K = max (n₁, n₂), the maximum of the range of numbers of states if you specify a range of numbers of states in the NSTATE=n₁ : n₂ option

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=n₁ : n₂ option
- **YLag**: order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG=n₁ : n₂ option
- **Section ID Variable**: values of the variable specified in the SECTION= option in the ID statement
- **Time ID Variable**: values of the variable 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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelIndex</td>
<td>model index, if multiple models are estimated or scored</td>
</tr>
<tr>
<td>NState</td>
<td>number of states, if you specify a range of numbers of states in the NSTATE= option</td>
</tr>
<tr>
<td>YLag</td>
<td>order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= option</td>
</tr>
<tr>
<td>Section ID Variable</td>
<td>values of the variable specified in the SECTION= option in the ID statement</td>
</tr>
<tr>
<td>Time ID Variable</td>
<td>values of the variable specified in the TIME= option in the ID statement</td>
</tr>
<tr>
<td>State&lt;sub&gt;k&lt;/sub&gt;</td>
<td>filtered probability value for state ( k, k = 1, \ldots, 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</td>
</tr>
</tbody>
</table>

### OUT= Data Table Generated from the FORECAST Statement

The output data table contains the following variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelIndex</td>
<td>model index, if multiple models are estimated or scored</td>
</tr>
<tr>
<td>NState</td>
<td>number of states, if you specify a range of numbers of states in the NSTATE= option</td>
</tr>
<tr>
<td>YLag</td>
<td>order of the autoregressive process, if you specify a range of orders of the autoregressive processes in the YLAG= option</td>
</tr>
<tr>
<td>Section ID Variable</td>
<td>values of the variable specified in the SECTION= option in the ID statement</td>
</tr>
<tr>
<td>Time ID Variable</td>
<td>values of the variable specified in the TIME= option in the ID statement</td>
</tr>
<tr>
<td>Step</td>
<td>value ( h ) in the ( h )-step-ahead forecast</td>
</tr>
<tr>
<td>State&lt;sub&gt;k&lt;/sub&gt;</td>
<td>forecasted probability value for state ( k, k = 1, \ldots, 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</td>
</tr>
<tr>
<td>{DV&lt;sub&gt;p&lt;/sub&gt;}_Forecast</td>
<td>mean forecast for dependent variable ( p, p = 1, \ldots, k_y ), where ( k_y ) is the number of dependent variables; ( {DV&lt;sub&gt;p&lt;/sub&gt;} ) stands for the name of dependent variable ( p )</td>
</tr>
<tr>
<td>Q1</td>
<td>first quantile value, ( (1 - \alpha)/2 ), where ( \alpha ) is specified in the ALPHA=( \alpha ) option in the FORECAST statement</td>
</tr>
<tr>
<td>{DV&lt;sub&gt;p&lt;/sub&gt;}_Q1</td>
<td>first quantile forecast for dependent variable ( p, p = 1, \ldots, k_y ), where ( k_y ) is the number of dependent variables; ( {DV&lt;sub&gt;p&lt;/sub&gt;} ) stands for the name of dependent variable ( p )</td>
</tr>
<tr>
<td>Q2</td>
<td>second quantile value, ( (1 + \alpha)/2 ), where ( \alpha ) is specified in the ALPHA=( \alpha ) option in the FORECAST statement</td>
</tr>
<tr>
<td>{DV&lt;sub&gt;p&lt;/sub&gt;}_Q2</td>
<td>second quantile forecast for dependent variable ( p, p = 1, \ldots, k_y ), where ( k_y ) is the number of dependent variables; ( {DV&lt;sub&gt;p&lt;/sub&gt;} ) stands for the name of dependent variable ( p )</td>
</tr>
</tbody>
</table>
Q3 third quantile value, 1/2, for median

\{DV_p\}_Q3 median forecast for dependent variable \(p, p = 1, \ldots, k_y\), where \(k_y\) is the number of dependent variables; \{DV_p\} stands for the name of 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 specified in the SECTION= option in the ID statement
- **Time ID Variable** values of the variable specified in the TIME= option in the ID statement
- **Statek** smoothed probability value for state \(k, k = 1, \ldots, K\), 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>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Option values for the optimization algorithm</td>
<td>PRINTLEVEL=(n), where (n = 2) or 3</td>
</tr>
<tr>
<td>ARCoef</td>
<td>Coefficients of autoregressive variables for a regime-switching autoregression model</td>
<td>Default</td>
</tr>
<tr>
<td>ODS Table Name</td>
<td>Description</td>
<td>Option</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>ConstCoef</td>
<td>Coefficients of constants for a regime-switching regression model or regime-switching autoregression model</td>
<td>Default</td>
</tr>
<tr>
<td>Cov</td>
<td>Covariance matrices for a regime-switching regression model or regime-switching autoregression model</td>
<td>Default</td>
</tr>
<tr>
<td>ExogCoef</td>
<td>Coefficients of exogenous variables for a regime-switching regression model or regime-switching autoregression model</td>
<td>Default</td>
</tr>
<tr>
<td>FinalObjectiveFunction</td>
<td>Objective function value at the end of the optimization</td>
<td>PRINTLEVEL=3</td>
</tr>
<tr>
<td>FinalParameterEstimates</td>
<td>Final parameter values at the end of the optimization</td>
<td>PRINTLEVEL=3</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Log likelihood and information criteria</td>
<td>Default</td>
</tr>
<tr>
<td>InitialObjectiveFunction</td>
<td>Objective function value at the start of the optimization</td>
<td>PRINTLEVEL=n, where n =1, 2, or 3</td>
</tr>
<tr>
<td>InitialParameterEstimates</td>
<td>Initial parameter values at the start of the optimization</td>
<td>PRINTLEVEL=n, where n =1, 2, or 3</td>
</tr>
<tr>
<td>ISPV</td>
<td>Initial state probability vector</td>
<td>Default</td>
</tr>
<tr>
<td>IterHistory</td>
<td>Optimization iterations</td>
<td>PRINTITERFREQ=n, n &gt; 0</td>
</tr>
<tr>
<td>LinearCoef</td>
<td>Coefficients of linear trends for a regime-switching regression model or regime-switching autoregression model</td>
<td>Default</td>
</tr>
<tr>
<td>MCP</td>
<td>Mixture component probabilities for a GM HMM</td>
<td>Default</td>
</tr>
<tr>
<td>ModelInfo</td>
<td>Model information</td>
<td>Default</td>
</tr>
<tr>
<td>Mu</td>
<td>Mean vectors, $\mu$, for Gaussian or GM HMM</td>
<td>Default</td>
</tr>
<tr>
<td>NObs</td>
<td>Observation information</td>
<td>Default</td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Model parameter estimates</td>
<td>Default</td>
</tr>
<tr>
<td>PriorHyperparm</td>
<td>Prior hyperparameters</td>
<td>METHOD=MAP</td>
</tr>
<tr>
<td>QuadCoef</td>
<td>Coefficients of quadratic trends for a regime-switching regression model or regime-switching autoregression model</td>
<td>Default</td>
</tr>
</tbody>
</table>
Table 13.6  continued

<table>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeasCoef</td>
<td>Coefficients of seasonal dummies for a regime-switching regression model or</td>
<td>Default</td>
</tr>
<tr>
<td></td>
<td>regime-switching autoregression model</td>
<td></td>
</tr>
<tr>
<td>Sigma</td>
<td>Covariance matrices, $\sigma$, for a Gaussian or GM HMM</td>
<td>Default</td>
</tr>
<tr>
<td>TPM</td>
<td>Transition probability matrix</td>
<td>Default</td>
</tr>
</tbody>
</table>

## Examples: HMM Procedure

### Example 13.1: Discovering the Hidden Market States

To buy or to sell, that is the question! If you know it is a bull market, you might choose to buy; if it is a bear market, you might choose to sell. 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).

Consider the following CRSP NYSE/NYSEMKT/Nasdaq/Arca Value-Weighted Market Index (KYINDNO=1000200, Center for Research in Security Prices at the University of Chicago’s Booth School of Business 2018) from January 19, 1926, to December 29, 2017. You can get access to 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 prices into the data table `vwmiO`:

```sas
libname crsplib "XXX";
data vwmiO;
  set crspInd.sfs_dind(where=(KYINDNO=1000200 and caldt='19JAN1926'd and caldt='29DEC2017'd));
  format date MMDDYY10.;
  date = caldt; price = aind;
  keep date price;
run;
```

The following code generates the weekly returns through the daily data and saves them in two data tables. One data table, `vwmi`, is the full-sample data table, which contains all weekly returns. The other data table, `vwmiIn`, is the in-sample data table, 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 evaluating the models.

```sas
%let ds = vwmi;
%let cutDate = '31DEC2000'd;
data &ds. &ds.In;
```
Example 13.1: Discovering the Hidden Market States

```sas
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;
run;
```

Before estimating any model, make a copy of the client-side data on the server:

```sas
data mycas.&ds.In; set &ds.In; run;
data mycas.&ds.; set &ds.; run;
```

To model the weekly returns, how many market states should be considered? How many AR lags should a market 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:

```sas
%macro estimateRSAR(myds, inEstDs, kStart, kEnd, pStart, pEnd, method, maxiter, qMultiStart);
proc hmm data=&myds.
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;
run;
%mend estimateRSAR;
```

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 estimates in the ML method would be reasonable. The \(k\)-state RS AR\((p)\) models, \(k = 2, \ldots, 9\), \(p = 1, \ldots, 5\), 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 anymore because the initial values are good.

\[
\begin{align*}
\ast & \text{(2) } \text{ML + initial values estimated from MAP's AR(0) for AR(0)} \\
& \text{and initial values estimated from AR(p-1) for AR(p), } p > 0; \\
\ast & \text{be aware that the following macro might take tens of minutes to finish;} \\
\ast & \text{uncomment it to run;} \\
\ast & \%\text{estimateRSAR(myds=mycas.\&ds.In, inEstDs=mycas.\&ds.InMAPLearn_k2to9_p0to0,} \\
& \text{kStart=2, kEnd=9, pStart=0, pEnd=5,} \\
& \text{method=ML, maxiter=128, qMultiStart=0);} \\
\end{align*}
\]

After estimating all 48 of the two- to nine-state RS AR(0) to AR(5) models, you get AICs. The following statements print the AICs, which are shown in Output 13.1.1:

\[
\begin{align*}
\ast & \text{print fit statistics in matrix form;} \\
& \text{data &ds.InMLStat_k2to9_p0to5; set mycas.\&ds.InMLStat_k2to9_p0to5; run;} \\
& \text{proc sort data=&ds.InMLStat_k2to9_p0to5; by modelIndex; run;} \\
& \%\text{macro printFitStat(fitStat);} \\
& \text{data &ds.In&fitStat.;} \\
& \text{set &ds.InMLStat_k2to9_p0to5 end=eof;} \\
& \text{array &fitStat.s(8,6) _TEMPORARY_;} \\
& \text{array yLags(6) yLag0-yLag5;} \\
& \text{&fitStat.s[nState-1,yLag+1] = &fitStat.;} \\
& \text{if eof then do;} \\
& \text{do i = 1 to 8;} \\
& \text{nState = i+1;} \\
& \text{do j = 1 to 6; yLags(j) = &fitStat.s[i,j]; end;} \\
& \text{output;} \\
& \text{end;} \\
& \text{keep nState yLag0-yLag5;} \\
& \text{run;} \\
& \text{proc print data=&ds.In&fitStat. label noobs} \\
& \text{style(header)={textalign=center};} \\
& \text{var nState yLag0-yLag5;} \\
& \text{label nState='k' yLag0='p = 0' yLag1='p = 1' yLag2='p = 2'} \\
& \text{yLag3='p = 3' yLag4='p = 4' yLag5='p = 5';} \\
& \text{run;} \\
& \%\text{mend;} \\
& \%\text{printFitStat(AIC);} \\
\end{align*}
\]

\[
\begin{align*}
\ast & \text{Uncomment the following macros if you want to see other fit statistics;} \\
\ast & \%\text{printFitStat(logLikelihood);} \\
\ast & \%\text{printFitStat(AICC);} \\
\ast & \%\text{printFitStat(BIC);} \\
\ast & \%\text{printFitStat(HQC);} \\
\end{align*}
\]

Output 13.1.1 indicates that the smallest AIC corresponds to the eight-state RS AR(4) model.
**Example 13.1: Discovering the Hidden Market States**

**Output 13.1.1** AICs for 48 RS AR Models

<table>
<thead>
<tr>
<th>k</th>
<th>p = 0</th>
<th>p = 1</th>
<th>p = 2</th>
<th>p = 3</th>
<th>p = 4</th>
<th>p = 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16987.55</td>
<td>16979.59</td>
<td>16981.64</td>
<td>16985.00</td>
<td>16988.80</td>
<td>16992.07</td>
</tr>
<tr>
<td>3</td>
<td>16820.74</td>
<td>16811.38</td>
<td>16816.45</td>
<td>16821.02</td>
<td>16825.88</td>
<td>16826.56</td>
</tr>
<tr>
<td>4</td>
<td>16772.85</td>
<td>16764.64</td>
<td>16754.79</td>
<td>16757.74</td>
<td>16763.77</td>
<td>16767.28</td>
</tr>
<tr>
<td>5</td>
<td>16728.05</td>
<td>16733.38</td>
<td>16738.14</td>
<td>16743.40</td>
<td>16750.67</td>
<td>16753.11</td>
</tr>
<tr>
<td>6</td>
<td>16697.18</td>
<td>16684.36</td>
<td>16684.90</td>
<td>16689.57</td>
<td>16695.59</td>
<td>16700.58</td>
</tr>
<tr>
<td>7</td>
<td>16701.64</td>
<td>16693.28</td>
<td>16682.39</td>
<td>16685.51</td>
<td>16690.77</td>
<td>16695.31</td>
</tr>
<tr>
<td>8</td>
<td>16685.88</td>
<td>16688.43</td>
<td>16687.35</td>
<td>16668.80</td>
<td>16650.11</td>
<td>16659.51</td>
</tr>
<tr>
<td>9</td>
<td>16711.81</td>
<td>16699.75</td>
<td>16698.66</td>
<td>16707.04</td>
<td>16679.98</td>
<td>16684.87</td>
</tr>
</tbody>
</table>

The following statements extract the model information for the AIC-selected eight-state RS AR(4) model:

* select 8-state RS-AR(4) model according to AIC or AICC;
* adjust data to be the same sample size for AR(5);

```plaintext
data mycas.&ds.InBackup; set mycas.&ds.In; run;
data mycas.&ds.In; set mycas.&ds.InBackup(where=(date>='30JAN1926'd)); run;
%estimateRSAR(myds=mycas.&ds.In, inEstDs=mycas.&ds.InMLLearn_k2to9_p0to5, kStart=8, kEnd=8, pStart=4, pEnd=4, method=ML, maxiter=0, qMultiStart=0);
data mycas.&ds.In; set mycas.&ds.InBackup; run;
```

The selected model has 112 parameters, whose exact values are not of interest in this example. The following statements calculate the sample mean and variance for each regime and then plot the corresponding Gaussian kernels in **Output 13.1.2**, which shows how each regime looks. Regimes 3, 5, and 8 seem like a bull market; they have significantly positive returns with low risk. The remaining regimes seem like a bear market; they have significantly negative returns or very high risk (or both). The regimes and the market states they represent are discussed further later.

```plaintext*
*print sample mean and variance for each regime according to smoothed probabilities, the weight;
proc hmm data=mycas.&ds.in(where=(date>='30JAN1926'd));
    score inmodel=mycas.&ds.inmlmodel_k8to8_p4to4;
    smooth out=mycas.&ds.inSmooth_k8to8_p4to4;
run;
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.inTruncated &ds.in(where=(date>='25FEB1926'd));
    by date;
    if(a=b);
run;
%macro getMeanAndStdErrOfAState(k);
data &ds.inMeanVar;
    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;
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');
%end;
output;
keep mean1-mean&k. sd1-sd&k. ;
run;
%mend;
%getMeanAndStdErrOfAState(8);

%macro plotLearning(myData,myColumn,k);
   proc sgplot data=&myData.;
      refline 0 / axis=x lineattrs=(thickness=3);
      histogram &myColumn. / nbins=200;
      %do i = 1 %to &k.;
         density &myColumn. / type=normal(mu=&&mu&i. sigma=&&sigma&i.)
            name="regime&i.";
         legendlabel="Expected Gaussian Dist. for Regime &i.";
      %end;
      keylegend %do i = 1 %to &k.; "regime&i." %end;
run;
%mend plotLearning;

%plotLearning(&ds.In,returnw,8);
Next, the INMODEL= option in the SCORE statement is used to forecast the VaRs at different levels, such as 1%, 5%, and 10%, as shown in Output 13.1.3. 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;
```
Chapter 13: The HMM Procedure

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;
%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;
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;
%mend evaluateVaR;
According to the $p$-values in the “Pr > ChiSq” column in Output 13.1.3, 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.

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 shown by comparing the average weekly log likelihoods of the in-sample period and the out-of-sample period. As shown in Output 13.1.4, 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).
Chapter 13: The HMM Procedure

\[ oosLL = (\text{fullLL} - \text{isLL})/855; \]
\[ \text{if(a=b);} \]
\[ \text{drop date isLL fullLL;} \]
\[ \text{run;} \]

\text{data inLL;}\n\text{set mycas.\&ds.InMLStat_k2to9_p0to5;}
\[ \text{isLL = logLikelihood} \div (3999-5); \]
\[ \text{keep modelIndex nState yLag isLL;} \]
\text{run;}
\text{proc sort data=inLL; by modelIndex; run;}

\text{data avgLL;}\n\text{merge inLL(in=a) oosLL(in=b);}
\[ \text{by modelIndex;} \]
\[ \text{if(a=b and (modelIndex=1 or modelIndex=41 or modelIndex=48));} \]
\[ \text{drop modelIndex;} \]
\text{run;}
\text{proc print data=avgLL noobs label;}
\[ \text{label nState='k' yLag='p' isLL='In-Sample' oosLL='Out-Of-Sample';} \]
\text{run;}

\textbf{Output 13.1.4} Comparison of Average Weekly Log Likelihoods

<table>
<thead>
<tr>
<th>k</th>
<th>p</th>
<th>In-Sample</th>
<th>Out-Of-Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>-2.12513</td>
<td>-2.14688</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>-2.05835</td>
<td>-2.10316</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>-2.05494</td>
<td>-2.12056</td>
</tr>
</tbody>
</table>

Finally, the decoded regimes provide a historical view of what happened, given all the available data. The following statements generate the decoded regimes for each week in the DECODE statement, and plot them in Output 13.1.5:

\[ * \text{plot decoded results;} \]
\text{proc hmm data=mycas.\&ds.;}
\[ \text{score inmodel=mycas.\&ds.inmlmodel_k8to8_p4to4;} \]
\[ \text{decode out=mycas.\&ds.Decode_k8to8_p4to4;} \]
\text{run;}
\text{data &ds.Decode_k8to8_p4to4; set mycas.\&ds.Decode_k8to8_p4to4; run;}
\text{proc sort data=&ds.Decode_k8to8_p4to4; by date; run;}
\text{proc sgplot data=&ds.Decode_k8to8_p4to4;}
\[ \text{scatter x=DATE y=state / group=state;} \]
\text{run;}

As shown in Output 13.1.5, the eight regimes seem to be categorized into three groups:

- Regimes 2, 3, and 5 belong to first group, the bullish-like market. Although there are very upward-moving days (regime 3) and somewhat downward-moving days (regime 2), the main trend is upward (regime 5). The risk in this group is pretty low.

- Regimes 6, 7, and 8 belong to second group, the bearish-like market. Although there are some good days (regime 8), there are more bad days (regime 6) and even very bad days (regime 7). This group has higher risk than the first group.
• Regimes 1 and 4 belong to the third group, the crisis-like market, where the risk is extremely high. Notice their appearance around the 1987 financial crisis, the 1997 Asian financial crisis, the Y2K crash, and the 2008 financial crisis.

When the market is in one group, it lasts for a long time before switching to the other groups. In fact, the transition probability matrix tells the similar 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.

Although the details are omitted here, if you compare these three groups to the three regimes that are found in the BIC-selected three-state RS AR(0) model, you will find that they match very well.

**Output 13.1.5 Decoded Regime for Each Week**
Chapter 13: The HMM Procedure

Example 13.2: Clustering Time Series

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, Girosi, 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, the process that one HMM on each cluster is estimated and the time series are moved between clusters according to their likelihoods on those HMMs 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 bi-state Gaussian HMMs. For the first Gaussian HMM, the parameters are

\[
\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad A = \begin{pmatrix} 0.6 & 0.4 \\ 0.4 & 0.6 \end{pmatrix}, \quad B = \begin{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \\ \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & 3 \\ 3 & 1 \end{pmatrix}, \quad \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}
\]

For the second Gaussian HMM, the parameters are

\[
\pi = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad A = \begin{pmatrix} 0.4 & 0.6 \\ 0.6 & 0.4 \end{pmatrix}, \quad B = \begin{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \\ \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & 3 \\ 3 & 1 \end{pmatrix}, \quad \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}
\]

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:

```plaintext
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 c1portion = 0.5;

  %let clpi1 = 0.5;
```

```plaintext```
Example 13.2: Clustering Time Series

%let c1a11 = 0.6;
%let c1a22 = 0.6;
%let c1mu1 = 0;
%let c1sigma1 = 1;
%let c1mu2 = 3;
%let c1sigma2 = 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<=%c1portion.) then cluster = 1;
    * else cluster = 2;
    * (2) all time series for one cluster are in one block;
    if(sec/&nSections.<=%c1portion.) 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.pi1;
          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;
st (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;

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.
The following statements display the parameter estimates for cluster 1 and cluster 2 in Output 13.2.2 and Output 13.2.3, respectively:

```latex
* 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 ($N = 10,000, T = 200$)

| Parameter | Estimate | Standard Error | t Value | Pr > |t| |
|-----------|----------|----------------|---------|------|---|
| TPM1_1    | 0.600116 | 0.000979       | 612.82  | <.0001 |
| TPM1_2    | 0.399884 | 0.000979       | 408.35  | <.0001 |
| TPM2_1    | 0.401123 | 0.000977       | 410.57  | <.0001 |
| TPM2_2    | 0.598877 | 0.000977       | 612.98  | <.0001 |
| MU1_1     | 2.997355 | 0.002636       | 1137.11 | <.0001 |
| MU2_1     | -0.000368| 0.002626       | -0.14   | 0.8885 |
| SIGMA1_1_1| 1.002381 | 0.003487       | 287.43  | <.0001 |
| SIGMA2_1_1| 0.994514 | 0.003457       | 287.69  | <.0001 |
In the second scenario, the sample size of one time series is increased to 400:

```r
%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 ($N = 10,000, T = 400$)

| Parameter | Estimate | Standard Error | t Value | Pr > |t| |
|-----------|----------|----------------|---------|-------|---|
| 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|

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**:

```r
* 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 \((N = 10,000, T = 400)\)

| Parameter | Estimate   | Standard Error | t Value | Pr > |t| |
|-----------|------------|----------------|---------|-------|---|
| TPM1_1    | 0.599184   | 0.000689       | 669.96  | <.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       | 668.85  | <.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 \((N = 10,000, T = 400)\)

| Parameter | Estimate   | Standard Error | t Value | Pr > |t| |
|-----------|------------|----------------|---------|-------|---|
| TPM1_1    | 0.399555   | 0.000827       | 483.00  | <.0001|
| TPM1_2    | 0.600445   | 0.000827       | 725.85  | <.0001|
| TPM2_1    | 0.600698   | 0.000827       | 726.30  | <.0001|
| TPM2_2    | 0.399302   | 0.000827       | 482.79  | <.0001|
| MU1_1     | -0.000477  | 0.001852       | -0.26   | 0.7966|
| MU2_1     | 2.998405   | 0.001852       | 1619.15 | <.0001|
| SIGMA1_1_1| 1.001528   | 0.002449       | 408.92  | <.0001|
| SIGMA2_1_1| 1.001306   | 0.002449       | 408.85  | <.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.

```sas
%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;
```
Example 13.2: Clustering Time Series

%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 ($N = 10,000$, $T = 20$)

<table>
<thead>
<tr>
<th>niterations</th>
<th>converged</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>1</td>
<td>0.7552</td>
</tr>
</tbody>
</table>

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 ($N = 10,000$, $T = 20$)

| Parameter | Estimate | Standard Error | t Value | Pr > |t| |
|-----------|----------|----------------|---------|-------|
| 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   | Estimate | Standard Error | t Value | Pr > |t| |
|-------------|----------|----------------|---------|------|------|
| TPM1_1      | 0.634639 | 0.002894       | 219.26  | <.0001 |
| TPM1_2      | 0.365361 | 0.002894       | 126.23  | <.0001 |
| TPM2_1      | 0.361656 | 0.002875       | 125.78  | <.0001 |
| TPM2_2      | 0.638344 | 0.002875       | 222.00  | <.0001 |
| MU1_1       | 0.011987 | 0.007950       | 1.51    | 0.1316 |
| MU2_1       | 3.010552 | 0.007647       | 393.69  | <.0001 |
| SIGMA1_1_1  | 1.029845 | 0.010807       | 95.30   | <.0001 |
| SIGMA2_1_1  | 0.981346 | 0.010152       | 96.66   | <.0001 |
References


# Chapter 14
The SEVSELECT Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: SEVSELECT Procedure</td>
<td>628</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>630</td>
</tr>
<tr>
<td>Getting Started: SEVSELECT Procedure</td>
<td>630</td>
</tr>
<tr>
<td>A Simple Example of Fitting Predefined Distributions</td>
<td>631</td>
</tr>
<tr>
<td>An Example with Left-Truncation and Right-Censoring</td>
<td>636</td>
</tr>
<tr>
<td>An Example of Modeling Regression Effects</td>
<td>641</td>
</tr>
<tr>
<td>Syntax: SEVSELECT Procedure</td>
<td>645</td>
</tr>
<tr>
<td>Functional Summary</td>
<td>646</td>
</tr>
<tr>
<td>PROC SEVSELECT Statement</td>
<td>649</td>
</tr>
<tr>
<td>BY Statement</td>
<td>658</td>
</tr>
<tr>
<td>CLASS Statement</td>
<td>659</td>
</tr>
<tr>
<td>DISPLAY Statement</td>
<td>659</td>
</tr>
<tr>
<td>DISPLAYOUT Statement</td>
<td>660</td>
</tr>
<tr>
<td>DIST Statement</td>
<td>661</td>
</tr>
<tr>
<td>EFFECT Statement</td>
<td>663</td>
</tr>
<tr>
<td>LOSS Statement</td>
<td>665</td>
</tr>
<tr>
<td>NLOPTIONS Statement</td>
<td>667</td>
</tr>
<tr>
<td>OUTPUT Statement</td>
<td>668</td>
</tr>
<tr>
<td>OUTSCORELIB Statement</td>
<td>671</td>
</tr>
<tr>
<td>SCALEMODEL Statement</td>
<td>673</td>
</tr>
<tr>
<td>SELECTION Statement</td>
<td>675</td>
</tr>
<tr>
<td>WEIGHT Statement</td>
<td>677</td>
</tr>
<tr>
<td>Programming Statements</td>
<td>678</td>
</tr>
<tr>
<td>Details: SEVSELECT Procedure</td>
<td>678</td>
</tr>
<tr>
<td>Predefined Distributions</td>
<td>678</td>
</tr>
<tr>
<td>Censoring and Truncation</td>
<td>688</td>
</tr>
<tr>
<td>Parameter Estimation Method</td>
<td>690</td>
</tr>
<tr>
<td>Parameter Initialization</td>
<td>692</td>
</tr>
<tr>
<td>Estimating Regression Effects</td>
<td>693</td>
</tr>
<tr>
<td>Empirical Distribution Function Estimation Methods</td>
<td>698</td>
</tr>
<tr>
<td>Statistics of Fit</td>
<td>704</td>
</tr>
<tr>
<td>Defining a Severity Distribution Model with the FCMP Procedure</td>
<td>709</td>
</tr>
<tr>
<td>Predefined Utility Functions</td>
<td>721</td>
</tr>
<tr>
<td>Scoring Functions</td>
<td>726</td>
</tr>
<tr>
<td>Custom Objective Functions</td>
<td>733</td>
</tr>
<tr>
<td>Input Data Tables</td>
<td>736</td>
</tr>
</tbody>
</table>
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 (BIC), 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:

```sas
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:

```sas
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:

```sas
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 $\Phi$ denotes the CDF of the standard normal distribution:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\log(x) - \mu}{\sigma} \right)^2}$$

and

$$F(x; \mu, \sigma) = \Phi\left( \frac{\log(x) - \mu}{\sigma} \right)$$

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:

```sas
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:

```sas
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 630, 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):

```sas
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 14.1 through Figure 14.5. First, information about the input data table is displayed followed by the “Model Selection” table, as shown in Figure 14.1. 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 14.1 Data Table Information and Model Selection Table**

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Converged</th>
<th>AICC</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burr</td>
<td>Yes</td>
<td>322.50845</td>
<td>No</td>
</tr>
<tr>
<td>Exp</td>
<td>Yes</td>
<td>508.12287</td>
<td>No</td>
</tr>
<tr>
<td>Gamma</td>
<td>Yes</td>
<td>320.50264</td>
<td>No</td>
</tr>
<tr>
<td>Igauss</td>
<td>Yes</td>
<td>319.61652</td>
<td>No</td>
</tr>
<tr>
<td>Logn</td>
<td>Yes</td>
<td>319.56579</td>
<td>Yes</td>
</tr>
<tr>
<td>Pareto</td>
<td>Yes</td>
<td>510.28172</td>
<td>No</td>
</tr>
<tr>
<td>Gpd</td>
<td>Yes</td>
<td>510.20576</td>
<td>No</td>
</tr>
<tr>
<td>Weibull</td>
<td>Yes</td>
<td>334.82373</td>
<td>No</td>
</tr>
</tbody>
</table>

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 14.2 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.
The plot in Figure 14.3 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 14.1, which indicates that the AICC values of the Igauss and Logn models are very close.
**Figure 14.3** Comparison of PDF Estimates of the Fitted Models

![Comparison of PDF Estimates of the Fitted Models](image)

**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 14.4. 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.
The parameter estimates of the Burr distribution are shown in Figure 14.5. These estimates are used in the next example.
**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.

```sas
/*----- 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:

```sas
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:

```sas
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 14.6 through Figure 14.13. In addition to the estimates of the range, mean, and standard deviation of $Y$, the “Descriptive Statistics for $y$” table shown in Figure 14.6 also indicates the number of observations that are left-truncated or right-censored. The “Model Selection” table in Figure 14.6 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 14.6** Summary Results for the Truncated and Censored Data

<table>
<thead>
<tr>
<th>Descriptive Statistics for $y$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
<td>100</td>
</tr>
<tr>
<td>Observations Used for Estimation</td>
<td>100</td>
</tr>
<tr>
<td>Minimum</td>
<td>2.30264</td>
</tr>
<tr>
<td>Maximum</td>
<td>8.34116</td>
</tr>
<tr>
<td>Mean</td>
<td>4.62007</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>1.23627</td>
</tr>
<tr>
<td>Left Truncated Observations</td>
<td>23</td>
</tr>
<tr>
<td>Right Censored Observations</td>
<td>14</td>
</tr>
</tbody>
</table>

The “Model Selection” table prepared for this example is shown in Figure 14.6. It indicates that the lognormal model is chosen by all the criteria.

**Figure 14.7** Comparing All Statistics of Fit for the Truncated and Censored Data

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Converged</th>
<th>AICC Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logn</td>
<td>Yes</td>
<td>298.92672 Yes</td>
</tr>
<tr>
<td>Burr</td>
<td>Yes</td>
<td>302.66229 No</td>
</tr>
<tr>
<td>Gamma</td>
<td>Yes</td>
<td>299.45293 No</td>
</tr>
<tr>
<td>Weibull</td>
<td>Yes</td>
<td>309.26779 No</td>
</tr>
</tbody>
</table>

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 14.7. It indicates that the lognormal model is chosen by all the criteria.

**Figure 14.7** Comparing All Statistics of Fit for the Truncated and Censored Data

<table>
<thead>
<tr>
<th>All Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Logn</td>
</tr>
<tr>
<td>Burr</td>
</tr>
<tr>
<td>Gamma</td>
</tr>
<tr>
<td>Weibull</td>
</tr>
</tbody>
</table>

Asterisk (*) denotes the best model in the column.

The plot that compares EDF and CDF estimates is shown in Figure 14.8. 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.
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 14.9. 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.
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 14.10. It indicates that the EDF matches the CDF very closely. In contrast, the P-P plot for the Weibull distribution, also shown in Figure 14.10, indicates a poor fit.

- 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 14.11; these plots confirm the conclusions that are arrived at by comparing the P-P plots.
**Figure 14.11** 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 14.12** 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 14.12** Burr Model Summary for the Truncated and Censored Data

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial Value</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theta</td>
<td>4.78102</td>
<td>1.05367E-8</td>
<td>Infty</td>
</tr>
<tr>
<td>Alpha</td>
<td>2.00000</td>
<td>1.05367E-8</td>
<td>Infty</td>
</tr>
<tr>
<td>Gamma</td>
<td>2.00000</td>
<td>1.05367E-8</td>
<td>Infty</td>
</tr>
</tbody>
</table>

**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 14.5). One of the ways in which you can specify the initial values is as follows:

```plaintext
/*------ 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 14.13. 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 14.13** Burr Model Optimization Summary for the Truncated and Censored Data

**The SEVSELECT Procedure**

**Burr Distribution**

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Iterations</td>
</tr>
<tr>
<td>Function Calls</td>
</tr>
<tr>
<td>Log Likelihood</td>
</tr>
</tbody>
</table>

**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 \(x_j\) (\(j = 1, \ldots, k\)), 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 \(\Theta\) and \(\beta_j (j = 1, \ldots, k)\) 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 $\theta$ denotes the scale parameter and $\Omega$ denotes the set of nonscale parameters. The scale $\theta$ is affected by the regressors as

$$ \theta = \theta_0 \cdot \exp\left(\sum_{j=1}^{k} \beta_j x_j\right) $$

where $\theta_0$ 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 $\mu$ is a log-transformed scale parameter. This can be verified by replacing $\mu$ with a parameter $\theta = e^{\mu}$, which results in the following expressions for the PDF $f$ and the CDF $F$ in terms of $\theta$ and $\sigma$, respectively, where $\Phi$ 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 $f(x; \theta, \sigma) = \frac{1}{\theta} f(\frac{x}{\theta}; 1, \sigma)$ condition and the CDF satisfies the $F(x; \theta, \sigma) = F(\frac{x}{\theta}; 1, \sigma)$ condition. This makes $\theta$ 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 $X_1$, $X_2$, and $X_3$ as follows:

$$ \mu = \log(\theta) = 1 + 0.75 X_1 - X_2 + 0.25 X_3 $$

```plaintext
/*----------- Lognormal Model with Regressors -----------*/
data test_sev3(keep=y x1-x3);
  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:
The following PROC SEVSELECT step fits the lognormal, Burr, and gamma distribution models to these data. The regressors are specified in the SCALEMODEL statement.

``` SAS
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 14.14 through Figure 14.18. The descriptive statistics of all the variables are shown in Figure 14.14.

**Figure 14.14** Summary Results for the Regression Example

### The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Descriptive Statistics for y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
</tr>
<tr>
<td>Observations Used for Estimation</td>
</tr>
<tr>
<td>Minimum</td>
</tr>
<tr>
<td>Maximum</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Standard Deviation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Descriptive Statistics for Regressors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>x1</td>
</tr>
<tr>
<td>x2</td>
</tr>
<tr>
<td>x3</td>
</tr>
</tbody>
</table>

The comparison of the fit statistics of all the models is shown in Figure 14.15. 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 14.15** Comparison of Statistics of Fit for the Regression Example

<table>
<thead>
<tr>
<th>Distribution</th>
<th>All Fit Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-2 Log Likelihood</td>
</tr>
<tr>
<td>Logn</td>
<td>187.49609*</td>
</tr>
<tr>
<td>Burr</td>
<td>190.69154</td>
</tr>
<tr>
<td>Gamma</td>
<td>188.91483</td>
</tr>
</tbody>
</table>

*Asterisk (*) denotes the best model in the column.*
The model information and the convergence results of the lognormal model are shown in Figure 14.16. 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 14.16 Convergence Results for the Lognormal Model with Regressors

The SEVSELECT Procedure

Logn Distribution

<table>
<thead>
<tr>
<th>Model Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
<td>Logn</td>
</tr>
<tr>
<td>Description</td>
<td>Lognormal Distribution</td>
</tr>
<tr>
<td>Distribution Parameters</td>
<td>2</td>
</tr>
<tr>
<td>Regression Parameters</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Convergence Status</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence criterion (GCONV=1E-8) satisfied.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization Iteration History</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Iter</td>
<td>Function Calls</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Technique</td>
<td>Trust Region</td>
</tr>
<tr>
<td>Iterations</td>
<td>3</td>
</tr>
<tr>
<td>Function Calls</td>
<td>10</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-93.74804701</td>
</tr>
</tbody>
</table>

The final parameter estimates of the lognormal model are shown in Figure 14.17. 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 $\mu$. 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 $\mu$ 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).
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 14.18. The estimate that is reported for the parameter $\theta$ is the base value for the scale parameter $\theta$. 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 14.18** Parameter Estimates for the Gamma Model with Regressors

| Parameter | DF | Estimate | Standard Error | t Value | 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 14.1 summarizes the statements and options that control the SEVSELECT procedure.

### Table 14.1  Functional Summary

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies the response variable to model along with censoring and truncation effects</td>
<td>LOSS</td>
<td></td>
</tr>
<tr>
<td>Specifies the weight variable</td>
<td>WEIGHT</td>
<td></td>
</tr>
<tr>
<td>Specifies distributions to fit</td>
<td>DIST</td>
<td></td>
</tr>
<tr>
<td>Specifies the classification variables</td>
<td>CLASS</td>
<td></td>
</tr>
<tr>
<td>Specifies the constructed regression effects</td>
<td>EFFECT</td>
<td></td>
</tr>
<tr>
<td>Specifies the regression effects to model</td>
<td>SCALEMODEL</td>
<td></td>
</tr>
<tr>
<td>Specifies the options for selecting the best subset of regression effects</td>
<td>SELECTION</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to display (an alternative to the PRINT= option)</td>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>Specifies the ODS tables to save as CAS output tables</td>
<td>DISPLAYOUT</td>
<td></td>
</tr>
<tr>
<td>Specifies the scoring functions and quantiles to write</td>
<td>OUTPUT</td>
<td></td>
</tr>
<tr>
<td>Specifies the library to write scoring functions to</td>
<td>OUTSCORELIB</td>
<td></td>
</tr>
<tr>
<td>Specifies the optimization options</td>
<td>NLOPTONS</td>
<td></td>
</tr>
<tr>
<td>Specifies programming statements that define an objective function</td>
<td>Programming statements</td>
<td></td>
</tr>
<tr>
<td><strong>Input and Output Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the input data table</td>
<td>PROC SEVSELECT</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the input data table for parameter estimates</td>
<td>PROC SEVSELECT</td>
<td>INEST=</td>
</tr>
<tr>
<td>Specifies the output data table for estimates of scoring functions and quantiles</td>
<td>OUTPUT</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies the output data table for parameter estimates</td>
<td>PROC SEVSELECT</td>
<td>OUTTEST=</td>
</tr>
<tr>
<td>Specifies the output data table for model information</td>
<td>PROC SEVSELECT</td>
<td>OUTMODELINFO=</td>
</tr>
<tr>
<td>Specifies the output data table for statistics of fit</td>
<td>PROC SEVSELECT</td>
<td>OUTSTAT=</td>
</tr>
<tr>
<td>Specifies the output item store for context and estimation results</td>
<td>PROC SEVSELECT</td>
<td>OUTSTORE=</td>
</tr>
<tr>
<td><strong>Data Interpretation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies left-censoring</td>
<td>LOSS</td>
<td>LEFTCENSORED=</td>
</tr>
<tr>
<td>Specifies left-truncation</td>
<td>LOSS</td>
<td>LEFTTRUNCATED=</td>
</tr>
</tbody>
</table>
Table 14.1  

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the probability of observability</td>
<td>LOSS</td>
<td>PROBOBSERVED=</td>
</tr>
<tr>
<td>Specifies right-censoring</td>
<td>LOSS</td>
<td>RIGHTCENSORED=</td>
</tr>
<tr>
<td>Specifies right-truncation</td>
<td>LOSS</td>
<td>RIGHTTRUNCATED=</td>
</tr>
<tr>
<td>Distribution Processing Options</td>
<td>DIST</td>
<td>INFUNCDEF=</td>
</tr>
<tr>
<td>Specifies the CAS table to read the distribution definitions from</td>
<td>DIST</td>
<td>LISTONLY</td>
</tr>
<tr>
<td>Specifies that distributions be listed in the log without estimating any models that use them</td>
<td>DIST</td>
<td>OUTFUNCDEF=</td>
</tr>
<tr>
<td>Specifies the CAS table to write the distribution definitions to</td>
<td>DIST</td>
<td>VALIDATEONLY</td>
</tr>
<tr>
<td>Specifies that distributions be validated without estimating any models that use them</td>
<td>DIST</td>
<td></td>
</tr>
<tr>
<td>Model Estimation Options</td>
<td>SCALEMODEL</td>
<td>DFMIXTURE=</td>
</tr>
<tr>
<td>Specifies the method for computing mixture distribution</td>
<td>SCALEMODEL</td>
<td>INFORMATIVE</td>
</tr>
<tr>
<td>Specifies that informative missingness be used to model missing values of regressors</td>
<td>SCALEMODEL</td>
<td>INIT=</td>
</tr>
<tr>
<td>Specifies initial values for model parameters</td>
<td>DIST</td>
<td>INITSAMPLE</td>
</tr>
<tr>
<td>Specifies the sample to be used for computing initial parameter estimates</td>
<td>PROC SEVSELECT</td>
<td>OBJECTIVE=</td>
</tr>
<tr>
<td>Specifies the objective function symbol</td>
<td>PROC SEVSELECT</td>
<td>OFFSET=</td>
</tr>
<tr>
<td>Specifies the offset variable in the scale regression model</td>
<td>SCALEMODEL</td>
<td>VARDEF=</td>
</tr>
<tr>
<td>Specifies the denominator for computing covariance estimates</td>
<td>PROC SEVSELECT</td>
<td></td>
</tr>
<tr>
<td>Regression Effect Selection Options</td>
<td>SELECTION</td>
<td>HIERARCHY=</td>
</tr>
<tr>
<td>Specifies whether and how the model hierarchy requirement is applied</td>
<td>SCALEMODEL</td>
<td>INCLUDE=</td>
</tr>
<tr>
<td>Specifies the set of regression effects to be forced in all models</td>
<td>SELECTION</td>
<td>METHOD=</td>
</tr>
<tr>
<td>Specifies the selection method</td>
<td>SELECTION</td>
<td>ORDERSELECT</td>
</tr>
<tr>
<td>Specifies the selection order for displaying effects in the selected model</td>
<td>PROC SEVSELECT</td>
<td>SELECTNLOTECH=</td>
</tr>
<tr>
<td>Specifies the optimization technique to use for the intermediate selection steps</td>
<td>PROC SEVSELECT</td>
<td></td>
</tr>
<tr>
<td>Specifies the set of regression effects to begin the selection process</td>
<td>SCALEMODEL</td>
<td>START=</td>
</tr>
<tr>
<td>Specifies how to apply the STOP= criterion</td>
<td>SELECTION</td>
<td>STOPHORIZON=</td>
</tr>
<tr>
<td>Description</td>
<td>Statement</td>
<td>Option</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Regression Effect Selection Method Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies a criterion for choosing the best model at each step</td>
<td>SELECTION</td>
<td>CHOOSE=</td>
</tr>
<tr>
<td>Specifies that the competitive form of the stepwise selection method be used</td>
<td>SELECTION</td>
<td>COMPETITIVE</td>
</tr>
<tr>
<td>Specifies the maximum number of effects in the model</td>
<td>SELECTION</td>
<td>MAXEFFECTS=</td>
</tr>
<tr>
<td>Specifies the maximum number of selection steps</td>
<td>SELECTION</td>
<td>MAXSTEPS=</td>
</tr>
<tr>
<td>Specifies the minimum number of effects in the model</td>
<td>SELECTION</td>
<td>MINEFFECTS=</td>
</tr>
<tr>
<td>Specifies a criterion to decide which effects enter or leave the model at each step</td>
<td>SELECTION</td>
<td>SELECT=</td>
</tr>
<tr>
<td>Specifies a criterion for stopping the selection process</td>
<td>SELECTION</td>
<td>STOP=</td>
</tr>
<tr>
<td><strong>Empirical Distribution Function (EDF) Estimation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the nonparametric method of CDF estimation</td>
<td>PROC SEVSELECT</td>
<td>EMPIRICALCDF=</td>
</tr>
<tr>
<td><strong>EMPIRICALCDF=MODIFIEDKM Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the $\alpha$ value for the lower bound on risk set size</td>
<td>PROC SEVSELECT</td>
<td>ALPHA=</td>
</tr>
<tr>
<td>Specifies the $c$ value for the lower bound on risk set size</td>
<td>PROC SEVSELECT</td>
<td>C=</td>
</tr>
<tr>
<td>Specifies the absolute lower bound on risk set size</td>
<td>PROC SEVSELECT</td>
<td>RSLB=</td>
</tr>
<tr>
<td><strong>EMPIRICALCDF=TURNBULL Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies that the final EDF estimates be maximum likelihood estimates</td>
<td>PROC SEVSELECT</td>
<td>ENSUREMLE</td>
</tr>
<tr>
<td>Specifies the relative convergence criterion</td>
<td>PROC SEVSELECT</td>
<td>EPS=</td>
</tr>
<tr>
<td>Specifies the maximum number of iterations</td>
<td>PROC SEVSELECT</td>
<td>MAXITER=</td>
</tr>
<tr>
<td>Specifies the threshold below which an EDF estimate is deemed to be 0</td>
<td>PROC SEVSELECT</td>
<td>ZEROPROB=</td>
</tr>
<tr>
<td><strong>OUT= Data Table Generation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the variables to copy from the DATA= data table to the OUT= data table</td>
<td>OUTPUT</td>
<td>COPYVARS=</td>
</tr>
<tr>
<td>Specifies the scoring functions to estimate</td>
<td>OUTPUT</td>
<td>FUNCTIONS=</td>
</tr>
<tr>
<td>Specifies the quantiles to estimate</td>
<td>OUTPUT</td>
<td>QUANTILES=</td>
</tr>
<tr>
<td><strong>OUTEST= Data Table Generation Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies that the OUTEST= data table contain covariance estimates</td>
<td>PROC SEVSELECT</td>
<td>COVOUT</td>
</tr>
</tbody>
</table>
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 738.

---

Table 14.1  
continued

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies that only the selected regression parameters be written</td>
<td><code>PROC SEVSELECT</code></td>
<td><code>SELECTOUT</code></td>
</tr>
<tr>
<td>Specifies that estimates of parameters not in the final model be 0</td>
<td><code>PROC SEVSELECT</code></td>
<td><code>ZEROEST</code></td>
</tr>
</tbody>
</table>

**Scoring Function Generation Options**  
Writes scoring functions of all models to one package  
      `OUTSCORELIB` | `COMMONPACKAGE`  
| Specifies the output data set 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=` |
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.

**BIC** specifies the Schwarz Bayesian information criterion (BIC) 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 $-2 \times \log(L)$ 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 704.

**DATA=CAS-libref.data-table** names the input data table for PROC SEVSELECT 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 630.

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 630. 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 737.

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 630. 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 738.

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 630. 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 740.

`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 630. 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 740.

`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 CCDM 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 630. 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 740.

`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.

**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-option) > < = (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-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.
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.

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
displaysthe 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 738.
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 691.

**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 738.

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 $cn^a$ proposed by Lai and Ying (1991). You can specify the values of $c$ and $a$ by using the C= and ALPHA= options, respectively. By default, the relative lower bound is used with values of $c = 1$ and $a = 0.5$. However, you can modify the default by using the following options:
Chapter 14: The SEVSELECT Procedure

**ALPHA** | **A=** *number*

specifies the value to use for $\alpha$ when the lower bound on the risk set size is defined as $cn^{\alpha}$. 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 $cn^{\alpha}$. 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 698.

**INITSAMPLE** *(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.
**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 733.

**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 14.2 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>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCENDING</td>
<td>Reverses the sort order</td>
</tr>
<tr>
<td>MISSING</td>
<td>Treats missing values as valid levels</td>
</tr>
<tr>
<td>ORDER=</td>
<td>Specifies the sort order for the levels</td>
</tr>
<tr>
<td>PARAM=</td>
<td>Specifies the parameterization of the variable</td>
</tr>
<tr>
<td>REF=</td>
<td>Specifies the reference level of the variable</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Allows design columns for a variable to enter or leave the model independently</td>
</tr>
</tbody>
</table>

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 742. 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 subsetted 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 742. 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 (BURRE), 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 678.

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.Svrtdist 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 709.

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.Svrtdist library.

_PREDEFINED_
specifies the list of eight predefined distributions: BURRE, EXP, GAMMA, GPD, IGAUSS, LOGN, PARETO, and WEIBULL. Although the TWEEDIE and STWEEDIE distributions are available in the Sashelp.Svrtdist 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.Svrtdist library, even if you specify the Sashelp.Svrtdist 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 693.

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 692. 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 svtdist 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 50 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 14.3 summarizes the options available in the EFFECT statement.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collection Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the constituents of the collection effect</td>
</tr>
<tr>
<td><strong>Multimember Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the levels of the multimember effect</td>
</tr>
<tr>
<td>NOEFFECT</td>
<td>Specifies that observations whose levels are all missing for the multimember variables should have 0 values in the corresponding design matrix columns</td>
</tr>
<tr>
<td>STDIZE</td>
<td>Standardizes the design matrix entries so that each observation has a sum of 1</td>
</tr>
<tr>
<td>WEIGHT=</td>
<td>Specifies the weight variable for the contributions of each classification effect</td>
</tr>
<tr>
<td><strong>Polynomial Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>DEGREE=</td>
<td>Specifies the degree of the polynomial</td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays details of the specified polynomial</td>
</tr>
<tr>
<td>MDEGREE=</td>
<td>Specifies the maximum degree of any variable in a term of the polynomial</td>
</tr>
<tr>
<td>NOSEPARATE</td>
<td>Treats the polynomial as a single effect with multiple degrees of freedom</td>
</tr>
<tr>
<td>STANDARDIZE=</td>
<td>Specifies centering and scaling suboptions for the variables that define the polynomial</td>
</tr>
<tr>
<td><strong>Spline Effects Options</strong></td>
<td></td>
</tr>
<tr>
<td>BASIS=</td>
<td>Specifies the type of basis (B-spline basis or truncated power function basis) for the spline effect</td>
</tr>
<tr>
<td>DATABOUNDARY</td>
<td>Uses the extremes of the data as boundary knots for a B-spline basis</td>
</tr>
<tr>
<td>DEGREE=</td>
<td>Specifies the degree of the spline effect</td>
</tr>
<tr>
<td>DETAILS</td>
<td>Displays the knots and locations for each spline basis function</td>
</tr>
<tr>
<td>KNOTMAX=</td>
<td>Requests equally spaced right-side boundary knots starting at the variables’ maximum and ending at the KNOTMAX= value</td>
</tr>
<tr>
<td>KNOTMETHOD=</td>
<td>Specifies how to construct the knots for the spline effect</td>
</tr>
</tbody>
</table>
Table 14.3  continued

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNOTMIN=</td>
<td>Requests equally spaced left-side boundary knots starting at the KNOTMIN= value and ending at the variables’ minimum value</td>
</tr>
<tr>
<td>NATURALCUBIC</td>
<td>Specifies a natural cubic spline basis for the spline effect</td>
</tr>
<tr>
<td>SEPARATE</td>
<td>Treats the spline basis for each variable as a separate effect when multiple variables are specified</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Treats each design matrix column as a separate effect for selection methods</td>
</tr>
</tbody>
</table>

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

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 688.

**LEFTTRUNCATED | LT=**

**LT=variable-name < (left-truncation-option) >**

**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 688.

You can specify the following `left-truncation-option` for an alternative interpretation of the left-truncation threshold:

**PROBOBSERVED | POBS=**

**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 689.

**RIGHTCENSORED | RC=**

**RC=variable-name**

**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 688.

**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 688.

**RIGHTTRUNCATED | RT=**

**variable-name**

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 688.

**NLOPTIONS Statement**

```plaintext
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:

```plaintext
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 40. For more information about the optimization methods, see the section “Choosing an Optimization Algorithm” on page 64.
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

**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 630.

**data-table** 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’:

```plaintext
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 709.
If you want to evaluate the scoring function at a specific value of the response variable, then specify a number \textit{arg}, which is enclosed in parentheses immediately after the \textit{function}. If you do not specify \textit{arg} or if you specify a missing value as \textit{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:

<table>
<thead>
<tr>
<th>Right-Censored</th>
<th>Left-Censored</th>
<th>( v )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>( y )</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>( l )</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>( r )</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>( (l + r)/2 )</td>
</tr>
</tbody>
</table>

You can specify the suffix of the variable that contains the estimate of the scoring function by specifying a valid SAS name as a \textit{variable}. If you do not specify a \textit{variable}, then PROC SEVSELECT uses \textit{function} as the suffix of the variable name.

To illustrate the \textit{FUNCTIONS=} option with an example, assume that you specify the following \textit{DIST} and \textit{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 \textit{mycas.Score} data table:

- **EXP\_CDF** contains the CDF estimate \( F_{\exp}(v, \hat{\theta}) \), 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 \( f_{\exp}(1000, \hat{\theta}) \), 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 \textit{mycas.Score} data table:

- **LOGN\_CDF** contains the CDF estimate \( F_{\logn}(v, \hat{\mu}, \hat{\sigma}) \), 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 \( f_{\logn}(1000, \hat{\mu}, \hat{\sigma}) \), 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 \textit{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
dvalues 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

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).

**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 100\( p_i \) 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 \( Q_{Burr}(p, \hat{\theta}, \hat{\alpha}, \hat{\gamma}) \), 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 737.

**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 OUTTEST= 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:

```sas
outscorelib sasuser.scorefunc;
```

When the SEVSELECT procedure runs successfully, the `temp-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 630. 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.
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 726.

**SCALEMODEL Statement**

**SCALEMODEL**  

```
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 X in the linear model structure $X\beta$ 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 47 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 $X\beta$ 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 693.

You can specify the following `scalemodel-options` in the SCALEMODEL statement:

\[ \text{DFMIXTURE} = \text{method-name} < (\text{method-options}) > \]

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 696.

**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 \( A \times x \), 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 62 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 694.

**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 35 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
```

This 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 58.

The SEVSELECT procedure supports a specific set of values for the following *method-options*:

```
CHOOSE=criterion
```

This 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
```

This 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 35 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 35 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 733.

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 $\Theta$. 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 14.4. 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 $\mu$ 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 709.
Table 14.4  Predefined PROC SEVSELECT Distributions

<table>
<thead>
<tr>
<th>Name</th>
<th>Distribution</th>
<th>Parameters</th>
<th>PDF ($f$)</th>
<th>CDF ($F$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BURR</td>
<td>Burr</td>
<td>$\theta &gt; 0, \alpha &gt; 0, \gamma &gt; 0$</td>
<td>$f(x) = \frac{\alpha \gamma^\alpha}{(x+\gamma)^\gamma(\alpha+1)}$</td>
<td>$F(x) = 1 - \left(\frac{1}{1+x/\gamma}\right)^\alpha$</td>
</tr>
<tr>
<td>EXP</td>
<td>Exponential</td>
<td>$\theta &gt; 0$</td>
<td>$f(x) = \frac{1}{\theta}e^{-\frac{x}{\theta}}$</td>
<td>$F(x) = 1 - e^{-\frac{x}{\theta}}$</td>
</tr>
<tr>
<td>GAMMA</td>
<td>Gamma</td>
<td>$\theta &gt; 0, \alpha &gt; 0$</td>
<td>$f(x) = \frac{z^\alpha e^{-\frac{x}{\theta}}}{\Gamma(\alpha)}$</td>
<td>$F(x) = \frac{1}{\Gamma(\alpha)}$</td>
</tr>
<tr>
<td>GPD</td>
<td>Generalized Pareto</td>
<td>$\theta &gt; 0, \xi &gt; 0$</td>
<td>$f(x) = \frac{1}{\theta} (1 + \xi \gamma)^{-1-1/\xi}$</td>
<td>$F(x) = 1 - (1 + \xi \gamma)^{-1/\xi}$</td>
</tr>
<tr>
<td>IGAUSS</td>
<td>Inverse Gaussian (Wald)</td>
<td>$\theta &gt; 0, \alpha &gt; 0$</td>
<td>$f(x) = \frac{1}{\sqrt{2\pi \gamma ^2}} e^{-\frac{(z-1)^2}{2\gamma ^2}}$</td>
<td>$F(x) = \Phi \left( (z-1) \sqrt{\frac{\alpha}{\gamma}} \right) + \Phi \left( -(z+1) \sqrt{\frac{\alpha}{\gamma}} \right) e^{2\alpha}$</td>
</tr>
<tr>
<td>LOGN</td>
<td>Lognormal</td>
<td>$\mu$ (no bounds), $\sigma &gt; 0$</td>
<td>$f(x) = \frac{1}{x\sigma \sqrt{2\pi}} e^{-\frac{1}{2} \left( \frac{\log(x) - \mu}{\sigma} \right)^2}$</td>
<td>$F(x) = \Phi \left( \frac{\log(x) - \mu}{\sigma} \right)$</td>
</tr>
<tr>
<td>PARETO</td>
<td>Pareto</td>
<td>$\theta &gt; 0, \alpha &gt; 0$</td>
<td>$f(x) = \frac{\alpha \theta^\alpha}{(x+\theta)^{\gamma+1}}$</td>
<td>$F(x) = 1 - \left( \frac{\theta}{x+\theta} \right)^\alpha$</td>
</tr>
<tr>
<td>TWEEDIE</td>
<td>Tweedie**</td>
<td>$p &gt; 1, \mu &gt; 0, \phi &gt; 0$</td>
<td>$f(x) = a(x, \phi) \exp \left[ \frac{1}{\phi} \left( \frac{x \mu ^{1-p} - \kappa(\mu, p)}{1-p} \right) \right]$</td>
<td>$F(x) = \int_0^x f(t) dt$</td>
</tr>
<tr>
<td>STWEEDIE</td>
<td>Scaled Tweedie**</td>
<td>$\theta &gt; 0, \lambda &gt; 0, 1 &lt; p &lt; 2$</td>
<td>$f(x) = a(x, \theta, \lambda, p) \exp \left( -\frac{x}{\theta} - \lambda \right)$</td>
<td>$F(x) = \int_0^x f(t) dt$</td>
</tr>
<tr>
<td>WEIBULL</td>
<td>Weibull</td>
<td>$\theta &gt; 0, \tau &gt; 0$</td>
<td>$f(x) = \frac{\tau z^{\tau-1} e^{-z^\tau}}{\beta}$</td>
<td>$F(x) = 1 - e^{-z^\tau}$</td>
</tr>
</tbody>
</table>

**For more information, see the section “Tweedie Distributions” on page 680.**

Notes:
1. $z = x/\theta$, wherever $z$ is used.
2. $\theta$ 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. $\Phi(y) = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{y}{\sqrt{2}} \right) \right)$ 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 \( \mu \) is the mean of the distribution, \( \phi \) 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 \( \alpha \) and scale parameter \( \theta \). 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 \( \lambda, \alpha, \theta \) are related to the natural parameters \( \mu, \phi, p \) of the Tweedie distribution as

\[
\lambda = \frac{\mu^{2-p}}{\phi(2 - p)}
\]

\[
\alpha = \frac{2 - p}{p - 1}
\]

\[
\theta = \phi(p - 1)\mu^{p-1}
\]

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 \( \mu, \phi, p \). 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 \( \theta \), Poisson mean parameter \( \lambda \), and the index parameter \( p \). The index parameter decides the shape parameter \( \alpha \) of the gamma distribution as

\[
\alpha = \frac{2 - p}{p - 1}
\]

The parameters \( \theta \) and \( \lambda \) of the STWEEDIE distribution are related to the parameters \( \mu \) and \( \phi \) of the TWEEDIE distribution as

\[
\mu = \lambda \theta \alpha
\]

\[
\phi = \frac{(\lambda \theta \alpha)^{2-p}}{\lambda(2 - p)} = \frac{\theta}{(p - 1)(\lambda \theta \alpha)^{p-1}}
\]
Predefined Distributions

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 $\alpha$ 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 693. The gamma scale parameter $\theta$ 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 $\theta$. Then, you can easily estimate how the mean of the distribution 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 $\lambda$ 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$\%$ percentile is denoted by $\pi_p$ ($0 \leq p \leq 1$). By definition, $\pi_p$ satisfies

$$F(\pi_p) \leq p \leq F(\pi_p)$$

where $F(\pi_p)$ = $\lim_{h \to 0} F(\pi_p - h)$. PROC SEVSELECT uses the following practical method of computing $\pi_p$. 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 $\epsilon$ 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 $\alpha$ 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:

$$l = \sum_{i=1}^{n} \log \left( \frac{y_i^{\alpha-1} e^{-y_i/\theta}}{\theta^\alpha \Gamma(\alpha)} \right) = (\alpha - 1) \sum_{i=1}^{n} \log(y_i) - \frac{1}{\theta} \sum_{i=1}^{n} y_i - n\alpha \log(\theta) - n \log(\Gamma(\alpha))$$

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 $\theta$:

$$\hat{\theta} = \frac{\sum_{i=1}^{n} y_i}{n\alpha} = \frac{m_1}{\alpha}$$

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:

\[
\psi(\alpha) \approx \log(\alpha) - \frac{1}{\alpha} \left( 0.5 + \frac{1}{12\alpha + 2} \right)
\]

This approximation is within 1.4% of the true value for all values of \( \alpha > 0 \) except when \( \alpha \) is arbitrarily close to the positive root of the digamma function (which is approximately 1.461632). Even for the values of \( \alpha \) that are close to the positive root, the absolute error between true and approximate values is still acceptable (\(|\psi(\alpha) - \psi(\alpha)| < 0.005 \) for \( \alpha > 1.07 \)). Solving the equation that arises from this approximation yields the following estimate of \( \alpha \):

\[
\hat{\alpha} = \frac{3 - d + \sqrt{(d - 3)^2 + 24d}}{12d}
\]

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 \( \alpha \):

\[
\hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}
\]

If \( m_2 - m_1^2 < \epsilon \) (almost zero sample variance), then \( \alpha \) is initialized as follows:

\[
\hat{\alpha} = 1
\]

After computing the estimate of \( \alpha \), the estimate of \( \theta \) 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:

\[
F(x) = 1 - \left( 1 + \frac{\xi x}{\theta} \right)^{-1/\xi}
\]

\[
= 1 - \left( \frac{\theta/\xi}{x + \theta/\xi} \right)^{1/\xi}
\]

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 \( \mu \) and shape parameter \( \lambda \), 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, \( \theta \):

\[
f(x) = \sqrt{\frac{\alpha \theta}{2\pi x^3}} \exp\left(-\frac{\alpha(x - \theta)^2}{2x \theta}\right)
\]

\[
F(x) = \Phi \left( \left( \frac{x}{\theta} - 1 \right) \sqrt{\frac{\alpha \theta}{x}} \right) + \Phi \left( -\left( \frac{x}{\theta} + 1 \right) \sqrt{\frac{\alpha \theta}{x}} \right) \exp (2\alpha)
\]

The parameters \( \theta \) (scale) and \( \alpha \) (shape) of this alternate form are related to the parameters \( \mu \) and \( \lambda \) 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:

$$\hat{\mu} = 2\log(m1) - \frac{\log(m2)}{2}, \quad \hat{\sigma} = \sqrt{\log(m2) - 2\log(m1)}$$

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:

$$\hat{\theta} = \frac{m_1m_2}{m_2 - 2m_1^2}, \quad \hat{\alpha} = \frac{2(m_2 - m_1^2)}{m_2 - 2m_1^2}$$

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 $\alpha$ and by computing $\hat{\theta} = \frac{\alpha + 2}{\alpha + 1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. The parameter $\mu$ 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 $\phi$ as

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^p}$$
STWEEDIE  STWEEDIE is a compound Poisson-gamma mixture distribution with mean $\mu = \lambda \theta \alpha$, where $\alpha$ is the shape parameter of the gamma random variables in the mixture and the parameter $p$ is determined solely by $\alpha$. First, the parameter $p$ is initialized by assuming that the sample is generated from a gamma distribution with shape parameter $\alpha$ 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

$$
\hat{\mu} = m_1 \quad \hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu} \hat{p}}
$$

Based on the relationship between the parameters of TWEEDIE and STWEEDIE distributions described in the section “Tweedie Distributions” on page 680, values of $\theta$ and $\lambda$ 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 $q_1$ and $q_3$ denote the estimates of the 25th and 75th percentiles, respectively. Using the formula for the CDF of Weibull distribution, they can be written as

$$
1 - \exp(-(q_1/\theta)^r) = 0.25 \\
1 - \exp(-(q_3/\theta)^r) = 0.75
$$

Simplifying and solving these two equations yields the following initial values,

$$
\hat{\theta} = \exp\left(\frac{r \log(q_1) - \log(q_3)}{r - 1}\right), \quad \hat{r} = \frac{\log(\log(4))}{\log(q_3) - \log(\theta)}
$$

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 14.5. 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 14.5  Parameter Initialization for Predefined Distributions

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Parameter</th>
<th>Name for INIT Option</th>
<th>Default Initial Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BURR</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\sqrt{\frac{m_2 m_3}{2m_3 - 3m_1 m_2}}$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>alpha</td>
<td>$1 + \frac{m_2}{2m_3 - 3m_1 m_2}$</td>
</tr>
<tr>
<td></td>
<td>$\gamma$</td>
<td>gamma</td>
<td>2</td>
</tr>
<tr>
<td>EXP</td>
<td>$\theta$</td>
<td>theta</td>
<td>$m_1$</td>
</tr>
<tr>
<td>GAMMA</td>
<td>$\theta$</td>
<td>theta</td>
<td>$m_1 / \alpha$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>alpha</td>
<td>$\frac{3 - d + \sqrt{(d - 3)^2 + 24d}}{12d}$</td>
</tr>
<tr>
<td>GPD</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\frac{m_1 m_2}{(2(m_2 - m_1^2))}$</td>
</tr>
<tr>
<td></td>
<td>$\xi$</td>
<td>xi</td>
<td>$(m_2 - 2m_1^2)/(2(m_2 - m_1^2))$</td>
</tr>
<tr>
<td>IGAUSS</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\frac{m_1}{(m_2 - m_1^2)}$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>alpha</td>
<td>$\frac{m_1^2}{(m_2 - m_1^2)}$</td>
</tr>
<tr>
<td>LOGN</td>
<td>$\mu$</td>
<td>mu</td>
<td>$2 \log(m_1) - \log(m_2) / 2$</td>
</tr>
<tr>
<td></td>
<td>$\sigma$</td>
<td>sigma</td>
<td>$\sqrt{\log(m_2) - 2 \log(m_1)}$</td>
</tr>
<tr>
<td>PARETO</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\frac{m_1 m_2}{(m_2 - 2m_1^2)}$</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>alpha</td>
<td>$2(m_2 - m_1^2)/(m_2 - 2m_1^2)$</td>
</tr>
<tr>
<td>TWEEDIE</td>
<td>$\mu$</td>
<td>mu</td>
<td>$m_1$</td>
</tr>
<tr>
<td></td>
<td>$\phi$</td>
<td>phi</td>
<td>$(m_2 - m_1^2)/m_1^p$</td>
</tr>
<tr>
<td></td>
<td>$\lambda$</td>
<td>lambda</td>
<td>$(\alpha + 2)/(\alpha + 1)$</td>
</tr>
<tr>
<td></td>
<td>$p$</td>
<td>p</td>
<td>$\frac{(\alpha + 2)(\alpha + 1)}{\alpha}$</td>
</tr>
<tr>
<td>STWEEDIE</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\frac{(m_2 - m_1^2)(p - 1)}{m_1}$</td>
</tr>
<tr>
<td></td>
<td>$\lambda$</td>
<td>lambda</td>
<td>$m_1^p / (\alpha (m_2 - m_1^2)(p - 1))$</td>
</tr>
<tr>
<td></td>
<td>$p$</td>
<td>p</td>
<td>$(\alpha + 2)/(\alpha + 1)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>where $\alpha = \frac{3 - d + \sqrt{(d - 3)^2 + 24d}}{12d}$</td>
</tr>
<tr>
<td>WEIBULL</td>
<td>$\theta$</td>
<td>theta</td>
<td>$\exp\left(\frac{r \log(q_1) - \log(q_3)}{r - 1}\right)$</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>tau</td>
<td>$\log(\log(4))/\left(\log(q_3) - \log(\hat{\theta})\right)$</td>
</tr>
</tbody>
</table>

Notes:
1. $m_k$ denotes the $k$th raw moment.
2. $d = \log(m_1) - (\sum \log(y_i))/n$
3. $q_1$ and $q_3$ 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
Censoring and Truncation

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 690) and how the empirical distribution function (EDF) is estimated (see the section “Empirical Distribution Function Estimation Methods” on page 698).

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 691) and an unconditional estimate of the empirical distribution function (see the section “EDF Estimates and Truncation” on page 703).

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 703. In such cases, PROC SEVSELECT uses conditional estimates of the CDF for computational or visual comparison to the EDF estimates.

Let \(t_{\text{min}}^l = \min_i \{t_i^l\}\) be the smallest value of the left-truncation threshold \(t_i^l\) is the left-truncation threshold for observation \(i\) and \(t_{\text{max}}^r = \max_i \{t_i^r\}\) 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

  \[
  \hat{F}_c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\text{min}}^l)}{\hat{F}(t_{\text{max}}^r) - \hat{F}(t_{\text{min}}^l)}
  \]

  If an observation is left-truncated but not right-truncated, then

  \[
  \hat{F}_c(y) = \frac{\hat{F}(y) - \hat{F}(t_{\text{min}}^l)}{1 - \hat{F}(t_{\text{min}}^l)}
  \]

  If an observation is right-truncated but not left-truncated, then

  \[
  \hat{F}_c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\text{max}}^r)}
  \]
● 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

\[
\hat{F}^c(y) = \frac{\hat{F}(y)}{\hat{F}(t_{\text{max}}^r)}
\]

If you specify regression effects, then \(\hat{F}(y), \hat{F}(t_{\text{min}}^l), \) and \(\hat{F}(t_{\text{max}}^r)\) are all computed from a mixture distribution, as described in the section “CDF and PDF Estimates with Regression Effects” on page 696.

---

**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 \(\Theta\). 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 \(\tau^l\) 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 \(\tau_h\) 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:

\[
L = \prod_{E} f_{\theta}(y) \prod_{E_t, t^l = t} f_{\theta}(y) / F_{\theta}(l^r) \prod_{C} F_{\theta}(c^l) - F_{\theta}(c^r) \prod_{C_t, t^l = t} F_{\theta}(c^l) - F_{\theta}(c^r) / F_{\theta}(l^r - l^l) \]

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 709. Some aspects of the optimization process can be controlled by using the \(NLOPT\) 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 690, the likelihood of the data is as follows:

\[
L = \prod_{E} f_{\theta}(y) \prod_{E_t, t^l = t} f_{\theta}(y) / F_{\theta}(l^r) \prod_{C} F_{\theta}(c^l) - F_{\theta}(c^r) \prod_{C_t, t^l = t} F_{\theta}(c^l) - F_{\theta}(c^r) / F_{\theta}(l^r - l^l) \]

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 \(t^l = t^l\)) 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 \(\theta\), then the theory states that as \(N \to \infty\), the distribution of \(\hat{\theta}\), the estimate of \(\theta\), converges to a normal distribution with mean \(\theta\) and covariance \(\hat{C}\) such that \(I(\theta) \cdot \hat{C} \to 1\), where \(I(\theta) = -E \left[ \nabla^2 \log(L(\theta)) \right]\) 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 \( G = \nabla^2(-\log(L(\theta))) \) denotes the Hessian matrix of the negative of log likelihood, then the covariance estimate is computed as

\[
\hat{C} = \frac{N}{d} G^{-1}
\]

where \( d \) is a denominator that is determined by the `VARDEF=` option. If `VARDEF=\text{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=\text{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 \( \theta_i \) 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 \( G \) is computed as \( G = \nabla^2 \log(U(\theta)) \), 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 `dist_PARMINIT` subroutine in the distribution model. For more information, see the section “Defining a Severity Distribution Model with the FCMP Procedure” on page 709.

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 693.

- 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 693.

**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 \( x_j, j = 1, \ldots, k \), denote the \( k \) regression effects. Let \( \beta_j \) 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 \( \Theta \). This model is usually referred to as the error model. The regression effects are modeled by extending the error model to the following form:

\[
Y \sim \exp(\sum_{j=1}^{k} \beta_j x_j) \cdot \mathcal{F}(\Theta)
\]

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 \( \theta \) denote the scale parameter and \( \Omega \) 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 \( \theta \) is modeled by the regression effects as

\[
\theta = \theta_0 \cdot \exp(\sum_{j=1}^{k} \beta_j x_j)
\]
where \( \theta_0 \) is the base value of the scale parameter. Thus, the scale regression model consists of the following parameters: \( \theta_0, \Omega, \) and \( \beta_j (j = 1, \ldots, k) \).

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 SCALERMODEL 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 SCALERMODEL 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 \( \theta_0 \) and \( \beta_j \) parameters for all \( j \).
  
  You can specify the initial value for \( \theta_0 \) 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 \( \theta_0 \). 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 \( \beta_j \), 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 $\beta_0$ or $\beta_j$ parameters for all $j$, then PROC SEVSELECT initializes those parameters by using the following method:

Let a random variable $Y$ be distributed as $F(\theta, \Omega)$, where $\theta$ is the scale parameter. By the definition of the scale parameter, a random variable $W = Y/\theta$ is distributed as $G(\Omega)$ such that $G(\Omega) = F(1, \Omega)$. Given a random error term $e$ that is generated from a distribution $G(\Omega)$, 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 $\theta$ with the regression effects yields

$$\log(y) = \log(\theta) + \sum_{j=1}^{k} \beta_j x_j + \log(e)$$

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 $\beta_0$ and $\beta_j$:

$$\log(y) = \beta_0 + \sum_{j=1}^{k} \beta_j x_j$$

The estimates of $\beta_j (j = 1, \ldots, k)$ in the solution of this regression problem are used to initialize the respective regression parameters of the model. The estimate of $\beta_0$ is later used to initialize the value of $\theta_0$.

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 709. 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 $\theta_0$ 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 $\theta_0$ 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 $\{\hat{\lambda}_i\}$ ($i = 1, \ldots, N$) 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 \( d_k = w_k \) (\( k = 1, \ldots, N \)). 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 \( s_k (k = 1, \ldots, K) \) is computed as \( s_k = \exp(\hat{\lambda}_k) \), where \( \hat{\lambda}_k \) is the \( k \)th \( q \)-quantile from the set \( \{\hat{\lambda}_i\} \) (\( i = 1, \ldots, N \)). 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 \( q << N \), 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 (\( j = 1, \ldots, r \)), such that \( 1 \leq l_j \leq N \), then the scale of \( k \)th component distribution in the mixture is \( s_k = \exp(\hat{\lambda}_{lk}) \). 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 been computed by using the
entire input data table. PROC SEVSELECT does not compute global EDF estimates. Let \( \hat{X} \) denote a quantity
that depends on the EDF estimates. \( \hat{X} \) can be either an EDF-based initial value of a distribution parameter or
an EDF-based statistic of fit. PROC SEVSELECT estimates \( \hat{X} \) as follows: First, each grid node \( k \) computes
an estimate \( \hat{X}_k \) 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 \( \hat{X}_k \) values; that is, \( \hat{X} = \frac{1}{K} \sum_{k=1}^{K} \hat{X}_k \), 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 \( (y_i, t^L_i, t^R_i, c^R_i, c^L_i) \), \( i = 1, \ldots, N \),
where \( y_i \) is the value of the response variable, \( t^L_i \) is the value of the left-truncation threshold, \( t^R_i \) is the value
of the right-truncation threshold, \( c^R_i \) is the value of the right-censoring limit, and \( c^L_i \) is the value of the
left-censoring limit.

If an observation is not left-truncated, then \( t^L_i = t^1 \), where \( t^1 \) is the smallest value in the support of the
distribution; so \( F(t^L_i) = 0 \). If an observation is not right-truncated, then \( t^R_i = \tau_h \), where \( \tau_h \) is the largest
value in the support of the distribution; so \( F(t^R_i) = 1 \). If an observation is not right-censored, then \( c^R_i = t^1 \);
so \( F(c^R_i) = 0 \). If an observation is not left-censored, then \( c^L_i = \tau_h \); so \( F(c^L_i) = 1 \).

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 701.

The method descriptions assume that all observations are either uncensored or right-censored; that is, each observation is of the form \((y_i, t_i^l, t_i^r, c_i^r, \tau_h)\) or \((y_i, t_i^l, t_i^r, c_i^l, \tau_l)\).

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 \((-y_i, -t_i^r, -t_i^l, -c_i^l, \tau_l)\); 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 \(\delta_i\) 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

\[
F_n(y) = \begin{cases} 
0 & \text{if } y < y^{(1)} \\
\hat{F}_n(y^{(k)}) & \text{if } y^{(k)} \leq y < y^{(k+1)}, \ k = 1, \ldots, N-1 \\
\hat{F}_n(y^{(N)}) & \text{if } y^{(N)} \leq y
\end{cases}
\]

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 \((y_i^u, t_i^l, t_i^r, \tau_h)\), where \(y_i^u = c_i^l/2\).
- An observation that is both left-censored and right-censored is converted to an uncensored observation \((y_i^u, t_i^l, t_i^r, \tau_h)\), 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 \( \tau \) and \( \tau \) 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 \( \tau \). The term risk set has its origins in survival analysis; it contains the events that are at risk of failure at a particular time, \( \tau \). In other words, it contains the events that have survived up to time \( \tau \) and might fail at or after \( \tau \). 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):

\[
\hat{\sigma}_n(y_i) = \sqrt{\left(1 - \hat{F}_n(y_i)\right)^2 \cdot \sum_{\tau \leq y_i} \left( \frac{n(\tau)}{R_n(\tau)(R_n(\tau) - n(\tau))} \right)}
\]

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
Empirical Distribution Function Estimation Methods

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 \( \alpha \) 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 \( \alpha \), 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 \( I[R_n(\tau) \geq L] \) is replaced 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 \((y_i, t^L_i, t^R_i, c^L_i, c^R_i), i = 1, \ldots, N\) with notation described in the section “Notation” on page 698. For each observation, let \( A_i = (c^L_i, c^R_i) \) 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 \( B_i = (t^L_i, t^R_i] \) 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:

\[
L = \{c^L_i, 1 \leq i \leq N\} \cup \{t^L_i, 1 \leq i \leq N\}
\]

\[
R = \{c^R_i, 1 \leq i \leq N\} \cup \{t^R_i, 1 \leq i \leq N\}
\]

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)
Chapter 14: The SEVSELECT Procedure

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 \( \epsilon \). 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 \( \epsilon \) 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 \( \delta \). You can control the value of \( \delta \) by specifying the **ZERO PROB=** 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 \( s_j \) \( (j = 1, \ldots, M) \). Let \( s = \{s_j\} \) denote the set of these parameters. If \( G(s) = \nabla^2(-\log(L(s))) \) denotes the Hessian matrix of the negative of log likelihood, then the variance-covariance matrix of \( s \) is estimated as \( \hat{C}(s) = G^{-1}(s) \). 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, \( \tau_G \) denotes the smallest left-truncation threshold with a nonzero cumulative probability, and \( \tau_H \) 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 \( \tau_G \) and \( \tau_H \) by \( t^L_{\min} \) and \( t^R_{\max} \), respectively, defined as

\[
t^L_{\min} = \min\{t^L_k : 1 \leq k \leq N\}
\]
\[
t^R_{\max} = \max\{t^R_k : 1 \leq k \leq N\}
\]

These estimates of \( t^L_{\min} \) and \( t^R_{\max} \) are used to compute the conditional estimates of the CDF as described in the section “Truncation and Conditional CDF Estimates” on page 689.

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 \( t^L_i \), an observation \( j \) is added with weight \( w_j = (1 - p)/p \) and \( y_j = t^L_i \). 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 \( n(\tau) = \sum_{k=1}^{N^a} w_k I[y_k = \tau] \) and \( \tau \leq t^L_k \text{ and } \delta_k = 1 \), 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 709 and “Predefined Utility Functions” on page 721.
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,

\[
F_n(y) = \begin{cases} 
0 & \text{if } y < x[1] \\
F[k] & \text{if } x[k] \leq y < x[k + 1], k = 1, \ldots, N - 1 \\
F[N] & \text{if } x[N] \leq y
\end{cases}
\]

where \([k]\) denotes the \(k\)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:

\[
F_n(y) = \begin{cases} 
0 & \text{if } y < x[1] \\
\text{undefined} & \text{if } x[2k - 1] \leq y < x[2k], k = 1, \ldots, (N - 1)/2 \\
F[2k] = F[2k + 1] & \text{if } x[2k] \leq y < x[2k + 1], k = 1, \ldots, (N - 1)/2 \\
F[N] & \text{if } x[N] \leq y
\end{cases}
\]

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 BIC 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 \( y_i, i = 1, \ldots, N \), denote the response variable values. Let \( L \) be the likelihood as defined in the section “Likelihood Function” on page 690. 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. 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.

**BIC** The Schwarz Bayesian information criterion (BIC) is defined as

\[
\text{BIC} = -2 \log(L) + p \log(N)
\]

A model that has a smaller BIC 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 \( y_i ; i = 1, \ldots, N \) denote the sample of \( N \) values of the response variable. Let \( w_i \) denote the normalized weight of the \( i \)th observation. If \( w_{0i} \) denotes the original, unnormalized weight of the \( i \)th observation, then

\[
    w_i = N w_{0i} / \left( \sum_{i=1}^{N} w_{0i} \right).
\]

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 696.

- 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 689.

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:

\[
    \begin{align*}
        D^+ &= \max_i \left( \frac{r_i}{W} - Z_i \right) \\
        D^- &= \max_i (Z_i - \frac{r_{i-1}}{W}) \\
        KS &= \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}}
    \end{align*}
\]

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:

\[ D^+ = \max_i (F_n(Z_i) - Z_i), \text{ if } F_n(Z_i) \geq Z_i \]
\[ D^- = \max_i (Z_i - F_n(Z_i)), \text{ if } F_n(Z_i) < Z_i \]
\[ KS = \sqrt{W} \max(D^+, D^-) + \frac{0.19}{\sqrt{W}} \]

**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:

\[ AD = -W - \frac{1}{W} \sum_{i=1}^{N_n} W_i [(2r_i - 1) \log(Z_i) + (2W + 1 - 2r_i) \log(1 - Z_i)] \]

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,

\[ AD = N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(z) - z)^2}{z(1 - z)} dz \]
\[ = N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2}{z(1 - z)} dz \]
\[ = N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} \frac{(P_i - Q_i z)^2}{z(1 - z)} dz \]
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,

\[
AD = -N(Z_1 + \log(1 - Z_1) + \log(Z_K) + (1 - Z_K)) \\
+ N \sum_{i=2}^{K} [P_i^2 A_i - (Q_i - P_i)^2 B_i - Q_i^2 C_i]
\]

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

\[
AD = -N(1 + \log(1 - Z_1) + \log(Z_K)) \\
+ N \sum_{i=2}^{K} [F_n(Z_{i-1})^2 A_i - (1 - F_n(Z_{i-1}))^2 B_i]
\]

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:

\[
\text{CvM} = N \int_{-\infty}^{\infty} (F_n(y) - F(y))^2 dF(y)
\]

If the STANDARD method is used to compute the EDF, then the following formula is used:

\[
\text{CvM} = \frac{1}{12W} + \sum_{i=1}^{N_u} W_i \left( Z_i - \frac{(2r_i - 1)}{2W} \right)^2
\]

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,

\[
\text{CvM} = N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(z) - z)^2 \, dz
\]

\[
= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (F_n(Z_{i-1}) + S_i(z - Z_{i-1}) - z)^2 \, dz
\]

\[
= N \sum_{i=1}^{K+1} \int_{Z_{i-1}}^{Z_i} (P_i - Q_i z)^2 \, dx
\]

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,

\[
\text{CvM} = N \frac{Z_3}{3} + N \sum_{i=2}^{K+1} \left[ P_i^2 A_i - P_i Q_i B_i - \frac{Q_i^2 C_i}{3} \right]
\]

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

\[
\text{CvM} = N \frac{N^3}{3} + N \sum_{i=2}^{K+1} \left[ F_n(Z_{i-1})^2 (Z_i - Z_{i-1}) - F_n(Z_{i-1}) (Z_i^2 - Z_{i-1}^2) \right]
\]

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 14.6.

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 SAS Visual Data Management and Utility 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 $\infty$. 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 $\infty$). 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 DATA Step 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 678 has a distribution model associated with it. The functions and subroutines of all those models are available in the Sashelp.Svrtdist library. The order of the parameters in the signatures of the functions and subroutines is...
the same as listed in Table 14.4. 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.Svrtdist library in the CMPLIB= system option by using the OPTIONS global statement prior to using them.

Table 14.6 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 14.6 List of Functions and Subroutines That Define a Distribution Model

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Required</th>
<th>Expected to Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist_CDF</td>
<td>Function</td>
<td>YES¹</td>
<td>Cumulative distribution function value</td>
</tr>
<tr>
<td>dist_CDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the CDF</td>
</tr>
<tr>
<td>dist_CDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the CDF</td>
</tr>
<tr>
<td>dist_CONSTANTPARM</td>
<td>Subroutine</td>
<td>NO</td>
<td>Constant parameters</td>
</tr>
<tr>
<td>dist_DESCRIPTION</td>
<td>Function</td>
<td>NO</td>
<td>Description of the distribution</td>
</tr>
<tr>
<td>dist_LOGCDF</td>
<td>Function</td>
<td>YES¹</td>
<td>Log of cumulative distribution function value</td>
</tr>
<tr>
<td>dist_LOGCDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the LOGCDF</td>
</tr>
<tr>
<td>dist_LOGCDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the LOGCDF</td>
</tr>
<tr>
<td>dist_LOGPDF</td>
<td>Function</td>
<td>YES²</td>
<td>Log of probability density function value</td>
</tr>
<tr>
<td>dist_LOGPDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the LOGPDF</td>
</tr>
<tr>
<td>dist_LOGPDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the LOGPDF</td>
</tr>
<tr>
<td>dist_LOGSDF</td>
<td>Subroutine</td>
<td>NO</td>
<td>Log of survival function value</td>
</tr>
<tr>
<td>dist_LOGSDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the LOGSDF</td>
</tr>
<tr>
<td>dist_LOGSDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the LOGSDF</td>
</tr>
<tr>
<td>dist_LOWERBOUNDS</td>
<td>Subroutine</td>
<td>NO</td>
<td>Lower bounds on parameters</td>
</tr>
<tr>
<td>dist_PARMINIT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Initial values for parameters</td>
</tr>
<tr>
<td>dist_PDF</td>
<td>Function</td>
<td>YES²</td>
<td>Probability density function value</td>
</tr>
<tr>
<td>dist_PDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the PDF</td>
</tr>
<tr>
<td>dist_PDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the PDF</td>
</tr>
<tr>
<td>dist_QUANTILE</td>
<td>Function</td>
<td>NO</td>
<td>Quantile for a particular CDF value</td>
</tr>
<tr>
<td>dist_SCALETRANSFORM</td>
<td>Function</td>
<td>NO</td>
<td>Type of relationship between the first distribution parameter and the scale parameter</td>
</tr>
<tr>
<td>dist_SDF</td>
<td>Function</td>
<td>NO</td>
<td>Survival function value</td>
</tr>
<tr>
<td>dist_SDFGRADIENT</td>
<td>Subroutine</td>
<td>NO</td>
<td>Gradient of the SDF</td>
</tr>
<tr>
<td>dist_SDFHESSIAN</td>
<td>Subroutine</td>
<td>NO</td>
<td>Hessian of the SDF</td>
</tr>
<tr>
<td>dist_UPPERBOUNDS</td>
<td>Subroutine</td>
<td>NO</td>
<td>Upper bounds on parameters</td>
</tr>
</tbody>
</table>

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
  - $p_1$ Numeric value of the first parameter
  - $p_2$ Numeric value of the second parameter
  - ...  
  - $p_m$ Numeric value of the $m$th parameter
- **Return value**: Numeric value that contains the CDF value $F(x; p_1, p_2, \ldots, p_m)$

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:

$$F(x; p_1, p_2, \ldots, p_m) = F\left(\frac{x}{p_1}; 1, p_2, \ldots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \ldots, p_m) = F\left(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m\right)$$

Here is a sample structure of the function for a distribution named ‘FOO’:

```plaintext
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
  
  \( \ldots \)
  
  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:

```plaintext
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 \( \text{dist} \_\text{PDF} \) 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.

\( \text{dist} \_\text{DESCRIPTION} \)

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’:

```plaintext
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’:

```plaintext
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 mth 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’:

```plaintext
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 `dim` 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 `dim`. Each nx[i] contains the number of observations in the original data that have the value x[i].
  - **F{*}** Input numeric array of dimension `dim`. 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 703.
  - **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 mth 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 $f(x; p_1, p_2, \ldots, p_m)$

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:

$$f(x; p_1, p_2, \ldots, p_m) = \frac{1}{p_1} f\left(\frac{x}{p_1}; 1, p_2, \ldots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \ldots, p_m) = \frac{1}{\exp(p_1)} f\left(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m\right)$$

Here is a sample structure of the function for a distribution named ‘FOO’:

```plaintext
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:**
  - \( \text{cdf} \) Numeric value of the cumulative distribution function (CDF) for which the quantile should be evaluated
  - \( p_1 \) Numeric value of the first parameter
  - \( p_2 \) Numeric value of the second parameter
  - \ldots
  - \( p_m \) Numeric value of the \( m \)th parameter
- **Return value:** Numeric value that contains the quantile \( F^{-1}(\text{cdf}; p_1, p_2, \ldots, p_m) \)

Here is a sample structure of the function for a distribution named ‘FOO’:

```plaintext
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 \( \mu \) and shape \( \sigma \), 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 \( \theta \) instead of the parameter \( \mu \) such that \( \mu = \log(\theta) \). The parameter \( \theta \) would then be a scale parameter. However, if you want to specify the distribution in terms of \( \mu \) and \( \sigma \) (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 \( \theta \) and \( \sigma \), you can simply define the distribution with \( \mu \) as the first parameter and specify that it is the logarithm of the scale parameter.
Chapter 14: The SEVSELECT Procedure

- **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’:

```plaintext
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 $S(x; p_1, p_2, \ldots, p_m)$

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:

$$S(x; p_1, p_2, \ldots, p_m) = S\left(\frac{x}{p_1}; 1, p_2, \ldots, p_m\right)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \ldots, p_m) = S\left(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m\right)$$

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 \( \infty \)).

- **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.
  - \ldots
  - \( p_m \) 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’:

```plaintext
subroutine FOO_UPPERBOUNDS(p1, p2);
  outargs p1, p2;
  p1 = <upper bound for P1>;
  p2 = <upper bound for P2>;
endsub;
```

dist_coreGRADIENT

defines a subroutine that returns the gradient vector of the specified \texttt{core} function of the distribution at
the specified values of the random variable and distribution parameters. The \texttt{core} keyword can be PDF,
CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- **Type**: Subroutine
- **Required**: NO
Chapter 14: The SEVSELECT Procedure

- **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
  
  \( p_1 \) Numeric value of the first parameter
  
  \( p_2 \) Numeric value of the second parameter
  
  \( \ldots \)
  
  \( p_m \) Numeric value of the \( m \)th parameter
  
  \( \text{grad}[^*] \) Output numeric array of size \( m \) that contains the gradient vector evaluated at the specified values. If \( h \) denotes the value of the core function, then the expected order of the values in the array is as follows: \( \frac{\partial h}{\partial p_1}, \frac{\partial h}{\partial p_2}, \ldots, \frac{\partial h}{\partial p_m} \)

- **Return value:** Numeric array that contains the gradient evaluated at \( x \) for the parameter values \((p_1, p_2, \ldots, p_m)\)

Here is a sample structure of the function for the core function CDF of a distribution named ‘FOO’:

```plaintext
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_core HESSIAN**

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:**
  
  \( x \) Numeric value of the random variable at which the Hessian matrix should be evaluated
  
  \( p_1 \) Numeric value of the first parameter
  
  \( p_2 \) Numeric value of the second parameter
  
  \( \ldots \)
  
  \( p_m \) Numeric value of the \( m \)th parameter
  
  \( \text{hess}[^*] \) 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 core function, then the expected order of the values in the array is as follows:

\[
\begin{bmatrix}
\frac{\partial^2 h}{\partial p_1^2} & \frac{\partial^2 h}{\partial p_1 \partial p_2} & \frac{\partial^2 h}{\partial p_1 \partial p_3} & \cdots & \frac{\partial^2 h}{\partial p_1 \partial p_m} \\
\frac{\partial^2 h}{\partial p_2 \partial p_1} & \frac{\partial^2 h}{\partial p_2^2} & \frac{\partial^2 h}{\partial p_2 \partial p_3} & \cdots & \frac{\partial^2 h}{\partial p_2 \partial p_m} \\
\frac{\partial^2 h}{\partial p_3 \partial p_1} & \frac{\partial^2 h}{\partial p_3 \partial p_2} & \frac{\partial^2 h}{\partial p_3^2} & \cdots & \frac{\partial^2 h}{\partial p_3 \partial p_m} \\
& & & & \frac{\partial^2 h}{\partial p_m \partial p_1} \\
& & & & \frac{\partial^2 h}{\partial p_m \partial p_2} \\
& & & & \cdots \\
& & & & \frac{\partial^2 h}{\partial p_m \partial p_m}
\end{bmatrix}
\]
● **Return value:** Numeric array that contains the lower triangular portion of the Hessian matrix evaluated at \(x\) for the parameter values \((p_1, p_2, \ldots, p_m)\)

Here is a sample structure of the subroutine for the core function LOGSDF of a distribution named ‘FOO’:

```markdown
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 703.

- **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 703.
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 703.
- **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 703 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 \( k_{\text{opt}} \), which is used for computing the cutoff point:

\[
k_{\text{opt}} = k_1^2 \left( \frac{\log(k_1)}{2 \log(m_1) - \log(k_1)} \right)^{2 - 2 \log(k_1)/\log(m_1)}
\]

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 status is set to 0. If the algorithm fails, then status 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 100\(p\)th percentile.
  
  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 703.

- **Return value:** The 100\(p\)th 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 \([x]\) 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 \( \epsilon \) 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 \( \epsilon \) 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:**
  - **n** Dimension of the \( x \) and \( nx \) input arrays
  - **x{*}** Input numeric array of dimension \( n \) that contains distinct values of the random variable that are observed in the sample
  - **nx{*}** Input numeric array of dimension \( n \) in which each \( nx[i] \) contains the number of observations in the sample that have the value \( x[i] \)
  - **nRaw** Desired number of raw moments. The output array \( raw \) contains the first \( nRaw \) raw moments.
  - **raw{*}** Output array of raw moments. The \( k \)th element in the array \( (raw{k}) \) contains the \( k \)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:**
n  Dimension of the input array x
x{*}  Numeric array that contains the values to be sorted at input. The subroutine uses the
same array to return the sorted values.
flag  A numeric value that controls the sort order. If flag is 0, then the values are sorted in
an ascending order. If flag 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

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

\[
E[(X \wedge u)^{k}] = \int_{0}^{u} x^{k} f(x)dx + \int_{u}^{\infty} u^{k} f(x)dx
\]

\[
= \int_{0}^{u} x^{k} f(x)dx + u^{k} (1 - F(u))
\]

where \(f(x)\) and \(F(x)\) denote the PDF and the CDF of the distribution, respectively. The LIMMO-
MENT 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 set. 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:
scoring functions

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:

```sas
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 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:

```sas
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:

```sas
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’:

```plaintext
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 709.

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:

```plaintext
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:

```plaintext
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 $x^\star$ represents the array of regressors:

```plaintext
function SEV_BAR_FOO(y);
    /* returns value of FOO_BAR for the supplied value
of y and fitted values of P1, P2 */
endsub;
```
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:

```fortran
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:

```fortran
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 $\theta_0$ (the base value of the scale parameter) or $\log(\theta_0)$ if the distribution has a log-scale parameter. For more information about $\theta_0$, see the section “Estimating Regression Effects” on page 693.

**SEV_PARMNAME** | **SEV_PARMNAME**$_{{\text{dist}}}$

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$_{{\text{dist}}}$ is created in the sevfit package. SEV_PARMNAME$_{{\text{dist}}}$ 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:

```plaintext
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**:

  \[ \text{index} \]

  specifies the numeric value of the index of the regressor for which you want the name. The value of \text{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 \text{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 \text{index} in the SCALEMODEL statement. If you specify an OFFSET= option variable in the SCALEMODEL statement, then for an \text{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:

```plaintext
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:

- **_PDF_(x)** evaluates the probability density function (PDF) of a distribution at the current value of a data table variable \( x \).
- **_CDF_(x)** evaluates the cumulative distribution function (CDF) of a distribution at the current value of a data table variable \( x \).
- **_SDF_(x)** evaluates the survival distribution function (SDF) of a distribution at the current value of a data table variable \( x \).
Chapter 14: The SEVSELECT Procedure

_**LOGPDF_(x) evaluates the natural logarithm of the PDF of a distribution at the current value of a data table variable x.**

_**LOGCDF_(x) evaluates the natural logarithm of the CDF of a distribution at the current value of a data table variable x.**

_**LOGSDF_(x) evaluates the natural logarithm of the SDF of a distribution at the current value of a data table variable x.**

_**EDF_(x) evaluates the empirical distribution function (EDF) estimate at the current value of a data table variable x.** Internally, PROC SEVSELECT computes the estimate by using the SVRTUTIL_EDF function as described in the section “Predefined Utility Functions” on page 721. 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 721. 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 721.**

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**:}

```plaintext
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 \texttt{MYOBJ} as an objective function symbol by using the \texttt{OBJECTIVE=} option in the \texttt{PROC SEVSELECT} statement. The response variable \texttt{Y} and left-truncation variable \texttt{T} are specified in the \texttt{LOSS} statement. The distribution \texttt{D} is specified in the \texttt{DIST} statement. The remaining statements constitute a program that computes the value of the \texttt{MYOBJ} symbol.

Let the distribution \texttt{D} have the parameters \texttt{P1} and \texttt{P2}. To estimate the model for this distribution, \texttt{PROC SEVSELECT} internally converts the generic program to the following program specific to distribution \texttt{D}:

\begin{verbatim}
myobj = -D_LOGPDF(y, p1, p2);
if (not(missing(t))) then
    myobj = myobj + log(1-D_CDF(t, p1, p2));
\end{verbatim}

Note that the generic keyword functions \texttt{_LOGPDF} and \texttt{_CDF} have been replaced by the distribution-specific functions \texttt{D_LOGPDF} and \texttt{D_CDF}, respectively, and the argument list of each is expanded to include the distribution parameters. You must have defined the \texttt{D_LOGPDF} and \texttt{D_CDF} functions previously; they are assumed to be available in the library \texttt{Work.Mydist} that you specify in the \texttt{CMPLIB=} option.

The program is executed for each observation in the data table \texttt{mycas.Test} to compute the value of \texttt{MYOBJ} by using the values of variables \texttt{Y} and \texttt{T} in that observation and internally computed values of the model parameters \texttt{P1} and \texttt{P2}. The values of \texttt{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 \texttt{PROC SEVSELECT} automatically differentiates \texttt{MYOBJ} with respect to the parameters \texttt{P1} and \texttt{P2}. The optimizer iterates through various combinations of the values of \texttt{P1} and \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{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 \texttt{CMPLIB=} system option or in \texttt{Sashelp.Svrtdist}, the predefined functions library. In the preceding example, it is assumed that the functions \texttt{D_LOGPDF} and \texttt{D_CDF} are defined in the library \texttt{Work.Mydist}, which you specify in the \texttt{CMPLIB=} option.

- You can use most \texttt{DATA} step statements and functions in your program. The \texttt{DATA} step file and the data set I/O statements (for example, \texttt{INPUT}, \texttt{FILE}, \texttt{SET}, and \texttt{MERGE}) are not available. However, some \texttt{PUT} statement functionality is supported. For more information, see the section “\texttt{PROC FCMP}
and DATA Step Differences” in SAS Visual Data Management and Utility 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:

\[
S = \_SEVTHETA0 \times \exp(\_SEVBETA1 \times x_1 + \_SEVBETA2 \times x_2 + \_SEVBETA3 \times x_3);
\]

If a model contains a distribution \( D_1 \) 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:

\[
M = \_SEVTHETA0 + \_SEVBETA1 \times x_1 + \_SEVBETA2 \times x_2 + \_SEVBETA3 \times x_3;
\]

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 \(<\text{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:

- **_MODEL_** identifying name of the distribution model. The observation contains information about this distribution.
- **_TYPE_** 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 741.

<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 693.

<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 ($\beta_j$) 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:

- _MODEL_ identifying name of the distribution model. The observation contains information about this distribution.
- _DEPVAR_ name of the loss variable.
- _DESCRIPTION_ descriptive name of the model. This has a nonmissing value only if the DESCRIPTION function has been defined for this model.
- _VALID_ 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 PROC SEVSELECT writes the reason for invalidity to the SAS log.
- _PARMNAME1 . . . _PARMNAMEM

$M$ 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 _PARMNAME$j$ ($j > m$) 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:

- _MODEL_ identifying name of the distribution model. The observation contains information about this distribution.
- _NMODELPARM_ number of parameters in the distribution.
- _NESTPARM_ number of estimated parameters. This includes the regression parameters, if you specify any regression effects.
_NOBS_ number of nonmissing observations that are used for parameter estimation.

_STATUS_ status of the parameter estimation process for this model. The possible values are listed in the section “_STATUS_ Variable Values” on page 741.

_SELECTED_ 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.

<Custom Objective> value of the custom objective function that is obtained at the end of the parameter estimation process. PROC SEVSELECT creates this variable only if you specify the OBJECTIVE= 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 (BIC) 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.
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 NLOPTONIONS statement.

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 14.7 relates the ODS tables to PRINT= options.

<table>
<thead>
<tr>
<th>ODS Table Name</th>
<th>Description</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllFitStatistics</td>
<td>Statistics of fit for all the distribution models</td>
<td>PRINT=ALLFITSTATS</td>
</tr>
<tr>
<td>BSplineDetails</td>
<td>Basis details for B-spline effects</td>
<td>PRINT=ALL and EFFECT SPLINE statement</td>
</tr>
<tr>
<td>ClassLevels</td>
<td>Level information for classification variables</td>
<td>Default for CLASS statement</td>
</tr>
<tr>
<td>CollectionLevelInfo</td>
<td>Level details for collection effects</td>
<td>PRINT=ALL and EFFECT COLLECTION statement</td>
</tr>
<tr>
<td>ConvergenceStatus</td>
<td>Convergence status of parameter estimation process</td>
<td>PRINT=CONVSTATUS</td>
</tr>
<tr>
<td>DescStats</td>
<td>Descriptive statistics for the response variable</td>
<td>PRINT=DESCSTATS</td>
</tr>
<tr>
<td>Distribution</td>
<td>Information about number of parameters in a distribution’s model</td>
<td>Default</td>
</tr>
<tr>
<td>DistributionInfo</td>
<td>Detailed information about all distributions</td>
<td>PRINT=DISTINFO</td>
</tr>
<tr>
<td>EntryCandidates</td>
<td>Details about candidates for entry into the model</td>
<td>PRINT=ALL and SELECTION statement</td>
</tr>
<tr>
<td>EstimationDetails</td>
<td>Details of the estimation process for all the distribution models</td>
<td>PRINT=ESTIMATIONDETAILS</td>
</tr>
<tr>
<td>FitStatistics</td>
<td>Statistics of fit</td>
<td>PRINT=STATISTICS</td>
</tr>
<tr>
<td>InitialValues</td>
<td>Initial parameter values and bounds</td>
<td>PRINT=INITIALVALUES</td>
</tr>
<tr>
<td>ODS Table Name</td>
<td>Description</td>
<td>Option</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>IterationHistory</td>
<td>Optimization iteration history</td>
<td>PRINT=NLOHISTORY</td>
</tr>
<tr>
<td>MMLevelInfo</td>
<td>Level information for multimember effects</td>
<td>PRINT=ALL and EFFECT MULTIMEMBER statement</td>
</tr>
<tr>
<td>ModelSelection</td>
<td>Model selection summary</td>
<td>PRINT=SELECTION</td>
</tr>
<tr>
<td>OptimizationSummary</td>
<td>Optimization summary</td>
<td>PRINT=NLOSUMMARY</td>
</tr>
<tr>
<td>ParameterEstimates</td>
<td>Final parameter estimates</td>
<td>PRINT=ESTIMATES</td>
</tr>
<tr>
<td>PolyDetails</td>
<td>Number of variables and columns, polynomial degree, and standardization method</td>
<td>PRINT=ALL and EFFECT POLY statement</td>
</tr>
<tr>
<td>PolyScaling</td>
<td>Centering and scaling details</td>
<td>PRINT=ALL and EFFECT POLY statement</td>
</tr>
<tr>
<td>RegDescStats</td>
<td>Descriptive statistics for the regression effects that do not contain a CLASS variable</td>
<td>PRINT=DESCSTATS</td>
</tr>
<tr>
<td>RemovalCandidates</td>
<td>Details about candidates for removal from the model</td>
<td>PRINT=ALL and SELECTION statement</td>
</tr>
<tr>
<td>SelectedEffects</td>
<td>List of effects selected for the model</td>
<td>Default for SELECTION statement</td>
</tr>
<tr>
<td>SelectionEstimationDetails</td>
<td>Details of the estimation process for all selection steps of a distribution’s model</td>
<td>PRINT=ESTIMATIONDETAILS and SELECTION statement</td>
</tr>
<tr>
<td>SelectionInfo</td>
<td>Information about the settings for model selection</td>
<td>Default for SELECTION statement</td>
</tr>
<tr>
<td>SelectionReason</td>
<td>Reason why the particular model was selected</td>
<td>Default for SELECTION statement</td>
</tr>
<tr>
<td>SelectionSummary</td>
<td>Summary information about model selection steps</td>
<td>Default for SELECTION statement</td>
</tr>
<tr>
<td>SplineKnots</td>
<td>Knot and boundary knot values</td>
<td>PRINT=ALL and EFFECT SPLINE statement</td>
</tr>
<tr>
<td>StopReason</td>
<td>Reason for termination of model selection</td>
<td>Default for SELECTION statement</td>
</tr>
<tr>
<td>TPFSplineDetails</td>
<td>Truncated power function (TPF) spline basis details</td>
<td>PRINT=ALL and EFFECT SPLINE statement</td>
</tr>
<tr>
<td>Timing</td>
<td>Timing information for various computational stages of the procedure</td>
<td>PRINT=ALL</td>
</tr>
</tbody>
</table>

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.

**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, with 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 14.8.

<table>
<thead>
<tr>
<th>ODS Graph Name</th>
<th>Plot Description</th>
<th>PLOTS= Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDFDistPlot</td>
<td>CDF plot per distribution</td>
<td>CDFPERDIST</td>
</tr>
<tr>
<td>CDFPlot</td>
<td>Comparative CDF plot</td>
<td>CDF</td>
</tr>
<tr>
<td>PDFDistPlot</td>
<td>PDF plot per distribution</td>
<td>PDFPERDIST</td>
</tr>
</tbody>
</table>

---

---
Table 14.8 continued

<table>
<thead>
<tr>
<th>ODS Graph Name</th>
<th>Plot Description</th>
<th>PLOTS= Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDFPlot</td>
<td>Comparative PDF plot</td>
<td>PDF</td>
</tr>
<tr>
<td>PPPlot</td>
<td>P-P plot of CDF and EDF</td>
<td>PP</td>
</tr>
<tr>
<td>QQPlot</td>
<td>Q-Q plot</td>
<td>QQ</td>
</tr>
</tbody>
</table>

**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 689.

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 696.

**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 $\sigma_n$, 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 $z_\alpha$ is the $\alpha$th quantile from the standard normal distribution and $\alpha$ 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 689.

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 696.

**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 696.

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 696.

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 706, 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 \( 
\hat{F}_n(z) 
\), 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 703. 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 689.

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 696.

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 \( \text{dist} \):

- If you have defined the \( \text{dist}_\text{QUANTILE} \) function that satisfies the requirements listed in the section “\( \text{dist}_\text{QUANTILE} \)” on page 717, then that function is invoked by using \( F_n \) and estimated distribution parameters as arguments. The QUANTILE function is defined in the Sashelp.Svrtdist library for all the predefined distributions.

- If the \( \text{dist}_\text{QUANTILE} \) function is not defined, then PROC SEVSELECT numerically inverts the \( \text{dist}_\text{CDF} \) function at the CDF value of \( F_n \) for the estimated distribution parameters. If the \( \text{dist}_\text{CDF} \) function is not defined, then the \( \exp(\text{dist}_\text{LOGCDF}) \) 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 703. The CDF inversion process, whether done numerically or by evaluating the \( \text{dist}_\text{QUANTILE} \) function, needs to accept an unconditional CDF value. So the \( F_n \) value is first transformed to an unconditional estimate \( F_n^u \) as

\[
F_n^u = F_n \cdot (\hat{F}(t_{\text{max}}^I) - \hat{F}(t_{\text{min}}^I)) + \hat{F}(t_{\text{min}}^I)
\]

where \( \hat{F}(t_{\text{max}}^I) \) and \( \hat{F}(t_{\text{min}}^I) \) are as defined in the section “Truncation and Conditional CDF Estimates” on page 689.

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 696.

---

**Examples: SEVSELECT Procedure**

**Example 14.1: 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 \( \mu \) with another parameter, \( \alpha = \mu / \sigma \), and define the PDF \( f \) and the CDF \( F \) as follows:

\[
f(x; \sigma, \alpha) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left( -\frac{1}{2} \left( \frac{x}{\sigma} - \alpha \right)^2 \right)
\]

\[
F(x; \sigma, \alpha) = \frac{1}{2} \left( 1 + \text{erf} \left( \frac{1}{\sqrt{2}} \left( \frac{x}{\sigma} - \alpha \right) \right) \right)
\]
Then, you can verify that $\sigma$ 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.

```sas
/****** Define normal distribution with scale parameter *******/
proc fcmp library=sashelp.svrtdist 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_parminit(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 distSCALETRANSFORM subroutine is defined for the distribution with

```
Example 14.1: Defining a Model for the Gaussian Distribution with a Scale Parameter

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 \( \sigma \) is affected by the regressors as follows:

\[
\sigma = \exp(1 + 0.5 \, X1 + 0.75 \, X3 - 2 \, X4 + X5)
\]

The sample is simulated such that the regressor \( X2 \) is linearly dependent on regressors \( X1 \) and \( X3 \).

```sas
/*--- 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);
    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.

```sas
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:

```sas
/*--- 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;
```
Chapter 14: The SEVSELECT Procedure

```plaintext
scalemodel x1-x5;
dist Normal_s burr logn pareto weibull;
run;
```

The “Model Selection” table in Output 14.1.1 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 14.1.1 Summary of Results for Fitting the Normal Distribution with Regressors

### The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Converged</th>
<th>Likelihood</th>
<th>-2 Log</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal_s</td>
<td>Yes</td>
<td>603.95786</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Burr</td>
<td>Maybe</td>
<td>612.81685</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Logn</td>
<td>Yes</td>
<td>749.20125</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Pareto</td>
<td>Yes</td>
<td>841.07022</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Weibull</td>
<td>Yes</td>
<td>612.77496</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

#### All Fit Statistics

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal_s</td>
<td>603.9576*</td>
<td>615.9576*</td>
<td>616.86108*</td>
<td>631.58888*</td>
<td>1.52388</td>
<td>4.00152</td>
<td>0.70769</td>
</tr>
<tr>
<td>Burr</td>
<td>612.81685</td>
<td>626.81685</td>
<td>628.03424</td>
<td>645.05304</td>
<td>1.50448*</td>
<td>3.90072*</td>
<td>0.63399*</td>
</tr>
<tr>
<td>Logn</td>
<td>749.20125</td>
<td>761.20125</td>
<td>762.10448</td>
<td>776.83227</td>
<td>2.88110</td>
<td>16.20558</td>
<td>3.04825</td>
</tr>
<tr>
<td>Pareto</td>
<td>841.07022</td>
<td>853.07022</td>
<td>853.97345</td>
<td>868.70124</td>
<td>4.83810</td>
<td>31.60568</td>
<td>6.84046</td>
</tr>
<tr>
<td>Weibull</td>
<td>612.77496</td>
<td>624.77496</td>
<td>625.67819</td>
<td>640.40598</td>
<td>1.50490</td>
<td>3.90559</td>
<td>0.63458</td>
</tr>
</tbody>
</table>

* 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 14.1.2. 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 NLOPTINS 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 14.1.2 Details of the Fitted Burr Distribution Model

### The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Burr Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Information</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Distribution Parameters</td>
</tr>
<tr>
<td>Regression Parameters</td>
</tr>
</tbody>
</table>
Example 14.1: Defining a Model for the Gaussian Distribution with a Scale Parameter

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 $X_2$ from the model:

```sas
/*--- 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 14.1.3 indicate that the Burr distribution has now converged. The NORMAL_S distribution is still the best distribution according to the likelihood-based criteria.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Converged</th>
<th>-2 Log Likelihood</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal_s</td>
<td>Yes</td>
<td>603.95786</td>
<td>Yes</td>
</tr>
<tr>
<td>Burr</td>
<td>Yes</td>
<td>612.79276</td>
<td>No</td>
</tr>
<tr>
<td>Weibull</td>
<td>Yes</td>
<td>612.77496</td>
<td>No</td>
</tr>
</tbody>
</table>
Example 14.2: 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:

\[ g(x) \quad \text{PDF of the body distribution} \]
\[ G(x) \quad \text{CDF of the body distribution} \]
\[ h(x) \quad \text{PDF of the tail distribution} \]
\[ H(x) \quad \text{CDF of the tail distribution} \]
\[ \theta \quad \text{scale parameter for the body distribution} \]
\[ \Omega \quad \text{set of nonscale parameters for the body distribution} \]
\[ \xi \quad \text{shape parameter for the GPD tail distribution} \]
\[ x_r \quad \text{normalized value of the response variable at which the tail starts} \]
\[ p_n \quad \text{mixing probability} \]

Given these notations, the PDF \( f(x) \) and the CDF \( F(x) \) of the mixed-tail distribution are defined as

\[
\begin{align*}
  f(x) &= \begin{cases} 
    \frac{p_n}{G(x_b)} g(x) & \text{if } x \leq x_b \\
    (1 - p_n) h(x - x_b) & \text{if } x > x_b
  \end{cases} \\
  F(x) &= \begin{cases} 
    \frac{p_n}{G(x_b)} G(x) & \text{if } x \leq x_b \\
    p_n + (1 - p_n) H(x - x_b) & \text{if } x > x_b
  \end{cases}
\end{align*}
\]

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 $\theta$, $\Omega$, $\xi$, $x_r$, and $p_n$. The scale of the GPD tail distribution $\theta_t$ is computed as

$$
\theta_t = \frac{G(x_b; \theta, \Omega)}{g(x_b; \theta, \Omega)} \frac{(1 - p_n)}{p_n} = \theta \frac{G(x_r; \theta = 1, \Omega)}{g(x_r; \theta = 1, \Omega)} \frac{(1 - p_n)}{p_n}
$$

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.Svrtdist library. The algorithm and the utility function are described in detail in the section “Predefined Utility Functions” on page 721. 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:

```sas
/*------- Define lognormal Body-GPD tail mixed distribution -------*/
proc fcmp library=sashelp.svrtdist 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;
end;
```
Chapter 14: The SEVSELECT Procedure

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);
x[e[j]] = x[i] - Xb;
nxe[j] = nx[i];
i = i + 1;
j = j + 1;
end;
Example 14.2: Defining a Model for Mixed-Tail Distributions

/* 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_CONSTANTPARAM subroutine. The signature of the LOGNGPD_CONSTANTPARAM subroutine lists only the constant parameter \( Pn \).

- The LOGNGPD_PARMINIT subroutine initializes the parameter \( x_r \) by first using the SVRTUTI-HILLCUTOFF utility function to compute an estimate of the cutoff point \( \hat{\alpha} \) and then computing \( x_r = \hat{\alpha}^e \). If SVRTUTIL_HILLCUTOFF fails to compute a valid estimate, then the SVRTUTIL_PERCENTILE utility function is used to set \( \hat{\alpha} \) 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 \( Xi \) 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 \( Pn \), 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;```
Chapter 14: The SEVSELECT Procedure

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 14.2.1 and Output 14.2.2. The “Model Selection” table in Output 14.2.1 indicates that all models converged. The “All Fit Statistics” table in Output 14.2.1 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 14.2.1 Summary of Fitting Mixed-Tail Distribution

The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log</th>
<th>Converged</th>
<th>Likelihood</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logngpd</td>
<td>Yes</td>
<td>3640</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Burr</td>
<td>Yes</td>
<td>3687</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Logn</td>
<td>Yes</td>
<td>3862</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Gpd</td>
<td>Yes</td>
<td>5344</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

All Fit Statistics

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log</th>
<th>Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logngpd</td>
<td>3640*</td>
<td>3650*</td>
<td>3650*</td>
<td>3674*</td>
<td>0.87957*</td>
<td>0.73810*</td>
<td>0.12350*</td>
<td></td>
</tr>
<tr>
<td>Burr</td>
<td>3687</td>
<td>3693</td>
<td>3693</td>
<td>3708</td>
<td>0.96199</td>
<td>1.12542</td>
<td>0.17778</td>
<td></td>
</tr>
<tr>
<td>Logn</td>
<td>3862</td>
<td>3866</td>
<td>3866</td>
<td>3875</td>
<td>1.38088</td>
<td>2.78456</td>
<td>0.36306</td>
<td></td>
</tr>
<tr>
<td>Gpd</td>
<td>5344</td>
<td>5348</td>
<td>5348</td>
<td>5358</td>
<td>7.15101</td>
<td>72.88461</td>
<td>14.86748</td>
<td></td>
</tr>
</tbody>
</table>

Asterisk (*) denotes the best model in the column.

The detailed results for the LOGNGPD distribution are shown in Output 14.2.2. The initial values table shows the fixed value of the $Pn$ 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 $Pn$ has not changed from its initial value.

Output 14.2.2 Detailed Results for the LOGNGPD Distribution

The SEVSELECT Procedure

Logngpd Distribution

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Logngpd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Lognormal Body-GPD Tail Distribution (Mu, Sigma, Xi, and Xr are free parameters; Pn is fixed at 0.8)</td>
</tr>
<tr>
<td>Distribution Parameters</td>
<td></td>
</tr>
</tbody>
</table>

Initial Parameter Values and Bounds

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial Value</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mu</td>
<td>1.32964</td>
<td>-Infy</td>
<td>Infy</td>
</tr>
<tr>
<td>Sigma</td>
<td>0.71508</td>
<td>1.05367E-8</td>
<td>Infy</td>
</tr>
<tr>
<td>Xi</td>
<td>0.29858</td>
<td>1.05367E-8</td>
<td>Infy</td>
</tr>
<tr>
<td>Xr</td>
<td>1.39446</td>
<td>1.05367E-8</td>
<td>Infy</td>
</tr>
<tr>
<td>Pn</td>
<td>0.80000</td>
<td>Constant</td>
<td>Constant</td>
</tr>
</tbody>
</table>

Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.
The following SAS statements use the parameter estimates to compute the value where the tail region is estimated to start ($x_b = e^{\bar{\mu} x_r}$) and the scale of the GPD tail distribution ($\theta_t = \frac{G(x_b)}{g(x_b)} \frac{1-p_n}{p_n}$):

```sas
/*-------- 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 14.2.3** Start of the Tail and Scale of the GPD Tail Distribution

<table>
<thead>
<tr>
<th>Parameter</th>
<th>DF</th>
<th>Estimate</th>
<th>Standard Error</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mu</td>
<td>1</td>
<td>1.61424</td>
<td>0.02518</td>
<td>64.10</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>Sigma</td>
<td>1</td>
<td>0.31740</td>
<td>0.01544</td>
<td>20.56</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>Xi</td>
<td>1</td>
<td>0.53941</td>
<td>0.09017</td>
<td>5.98</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>Xr</td>
<td>1</td>
<td>1.19757</td>
<td>0.03974</td>
<td>30.14</td>
<td>&lt;0.001</td>
<td></td>
</tr>
<tr>
<td>Pn</td>
<td>1</td>
<td>0.80000</td>
<td>Constant</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

The computed values of $x_b$ and $\theta_t$ are shown as $x_b$ and $\theta_t$ in **Output 14.2.3**. 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 $Xr$ 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 $Xr$ 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.
Example 14.3: 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 $\mu$ is affected by $k$ regression effects $x_j$, $j = 1, \ldots, k$, as follows,

$$
\mu = \mu_0 \exp \left( \sum_{j=1}^{k} \beta_j x_j \right)
$$

where $\mu_0$ represents the base value of the mean (you can think of $\mu_0$ as $\exp(\beta_0)$, where $\beta_0$ 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 $\theta$,

$$
\theta = \theta_0 \exp \left( \sum_{j=1}^{k} \beta_j x_j \right)
$$

where $\theta_0$ represents the base value of the scale parameter. As described in the section “Tweedie Distributions” on page 680, 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 $\lambda$ 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 $x1$, $x2$, and $x3$ as follows:

$$
\mu = 5 \exp(0.25 x1 - x2 + 3 x3)
$$

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 14.3.1 and Output 14.3.2. The model information and the convergence results are shown in Output 14.3.1.
Output 14.3.1 Convergence Results for the STWEEDIE Model with Regressors

The SEVSELECT Procedure

Stweedie Distribution

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Distribution Parameters</td>
</tr>
<tr>
<td>Regression Parameters</td>
</tr>
</tbody>
</table>

Convergence Status

Convergence criterion (FCONV=2.220446E-16) satisfied.

Optimization Summary

<table>
<thead>
<tr>
<th>Optimization Technique</th>
<th>Dual Quasi-Newton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>40</td>
</tr>
<tr>
<td>Function Calls</td>
<td>182</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-1044.321437</td>
</tr>
</tbody>
</table>

The final parameter estimates of the STWEEDIE regression model are shown in Output 14.3.2. The estimate that is reported for the parameter $\theta$ is the estimate of the base value $\theta_0$. The estimates of regression coefficients $\beta_1$, $\beta_2$, and $\beta_3$ are indicated by the rows of $x_1$, $x_2$, and $x_3$, respectively.

Output 14.3.2 Parameter Estimates for the STWEEDIE Model with Regressors

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Theta</td>
</tr>
<tr>
<td>Lambda</td>
</tr>
<tr>
<td>$\lambda$</td>
</tr>
<tr>
<td>$x_1$</td>
</tr>
<tr>
<td>$x_2$</td>
</tr>
<tr>
<td>$x_3$</td>
</tr>
</tbody>
</table>

If your goal is to explain the influence of regression effects on the scale parameter, then the output displayed in Output 14.3.2 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 $\mu$ and $\theta$, $\mu$ 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 $\beta_j$ are identical for the mean and the scale model, and the base value $\mu_0$ of the mean model is

$$
\mu_0 = \theta_0 \lambda^{\frac{2-p}{p-1}}
$$
The estimate of $\mu_0$ 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 $\Omega$, then the MLE of $g$ has an asymptotic normal distribution with mean $g(\hat{\Omega})$ and covariance $C = (\partial g)^{\prime} \Sigma (\partial g)$, where $\hat{\Omega}$ is the MLE of $\Omega$, $\Sigma$ is the estimate of covariance matrix of $\Omega$, and $\partial g$ is the gradient vector of $g$ with respect to $\Omega$ evaluated at $\hat{\Omega}$. For $\mu_0$, the function is $g(\Omega) = \theta_0 \lambda (2 - p)/(p - 1)$. The gradient vector is

$$
\partial g = \left( \frac{\partial g}{\partial \theta_0}, \frac{\partial g}{\partial \lambda}, \frac{\partial g}{\partial p}, \frac{\partial g}{\partial \beta_1}, \ldots, \frac{\partial g}{\partial \beta_k} \right)
$$

$$
= \left( \frac{\mu_0}{\theta_0}, \frac{\mu_0}{\lambda}, \frac{-\mu_0}{(p - 1)(2 - p)}, 0, \ldots, 0 \right)
$$

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:

```sas
/**** 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]);
  end;
```

Example 14.4: 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

\[ \text{Stderr} = \sqrt{\text{varMu0}}; \]
\[ \text{df} = \&\text{samplesize-6}; \]
\[ T = \mu_0/\text{Stderr}; \]
\[ \text{probT} = (1-\text{probt}(T, \text{df}))*2; \]
\[ \text{output}; \]
\[ \text{stop}; \]
\[ \text{end}; \]
\[ \text{run}; \]

\text{proc print data=Mu0 noobs s=';'; run;}

The estimates of \( \mu_0 \) that the preceding DATA step prepares are shown in Output 14.3.3. These estimates and the estimates of \( \beta_j \) as shown in Output 14.3.2 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 14.3.3** Estimate of the Base Value Mu0 of the Mean Parameter

| Parameter | Estimate | Standard Error | t Value | Approx Pr > |t|
|-----------|----------|----------------|---------|--------------|
| Mu0       | 4.47631  | 0.42334        | 10.5738 | 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 \( \phi \) is the dispersion parameter. Using the relationship between the parameters TWEEDIE and STWEEDIE distributions as described in the section “Tweedie Distributions” on page 680, the regression model for the dispersion parameter is

\[
\log(\phi) = (2 - p) \log(\mu) - \log(\lambda(2 - p))
\]

\[
= ((2 - p) \log(\mu_0) - \log(\lambda(2 - p))) + (2 - p) \sum_{j=1}^{k} \beta_j x_j
\]

Subsequently, the regression model for the variance is

\[
\log(V) = 2 \log(\mu) - \log(\lambda(2 - p))
\]

\[
= (2 \log(\mu_0) - \log(\lambda(2 - p))) + 2 \sum_{j=1}^{k} \beta_j x_j
\]

In summary, PROC SEVSELECT enables you to estimate regression effects on various parameters and statistics of the Tweedie model.

**Example 14.4: 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):

```sas
/* Grouped dental insurance claims data (Klugman, Panjer, and Willmot 1998) */
data gdental;
  input lowerbd upperbd count @@;
datalines;
  0 25 30 31 50 100 57 150 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:

```sas
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 LIBNAME statement, see SAS Cloud Analytic Services:
Language Reference.

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.

```sas
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:

```sas
/* 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 14.4.1. According to the “Model Selection” table in Output 14.4.1, all distribution models have converged. The “All Fit Statistics” table in Output 14.4.1 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 14.4.1** Statistics of Fit for Interval-Censored Data

### The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Model Selection Converged</th>
<th>AICC Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burr</td>
<td>Yes</td>
<td>51.41112 No</td>
</tr>
<tr>
<td>Exp</td>
<td>Yes</td>
<td>44.64768 Yes</td>
</tr>
<tr>
<td>Gamma</td>
<td>Yes</td>
<td>47.63969 No</td>
</tr>
<tr>
<td>Igauss</td>
<td>Yes</td>
<td>48.05874 No</td>
</tr>
<tr>
<td>Logn</td>
<td>Yes</td>
<td>47.34027 No</td>
</tr>
<tr>
<td>Pareto</td>
<td>Yes</td>
<td>47.16908 No</td>
</tr>
<tr>
<td>Gpd</td>
<td>Yes</td>
<td>47.16908 No</td>
</tr>
<tr>
<td>Weibull</td>
<td>Yes</td>
<td>47.47700 No</td>
</tr>
</tbody>
</table>

### All Fit Statistics

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burr</td>
<td>41.41112*</td>
<td>47.41112</td>
<td>51.41112</td>
<td>48.31888</td>
<td>0.08974*</td>
<td>0.00103*</td>
<td>0.00009816*</td>
</tr>
<tr>
<td>Exp</td>
<td>42.14768</td>
<td>44.14768</td>
<td>44.64768</td>
<td>44.45026*</td>
<td>0.26412</td>
<td>0.09936</td>
<td>0.01866</td>
</tr>
<tr>
<td>Gamma</td>
<td>41.92541</td>
<td>45.92541</td>
<td>47.63969</td>
<td>46.53058</td>
<td>0.19569</td>
<td>0.04608</td>
<td>0.00759</td>
</tr>
<tr>
<td>Igauss</td>
<td>42.34445</td>
<td>46.34445</td>
<td>48.05874</td>
<td>46.94962</td>
<td>0.34514</td>
<td>0.12301</td>
<td>0.02562</td>
</tr>
<tr>
<td>Logn</td>
<td>41.62598</td>
<td>45.62598</td>
<td>47.34027</td>
<td>46.23115</td>
<td>0.16853</td>
<td>0.01884</td>
<td>0.00333</td>
</tr>
<tr>
<td>Pareto</td>
<td>41.45480</td>
<td>45.45480</td>
<td>47.16908</td>
<td>46.05997</td>
<td>0.11423</td>
<td>0.00739</td>
<td>0.0009084</td>
</tr>
<tr>
<td>Gpd</td>
<td>41.45480</td>
<td>45.45480</td>
<td>47.16908</td>
<td>46.05997</td>
<td>0.11423</td>
<td>0.00739</td>
<td>0.0009084</td>
</tr>
<tr>
<td>Weibull</td>
<td>41.76272</td>
<td>45.76272</td>
<td>47.47700</td>
<td>46.36789</td>
<td>0.17238</td>
<td>0.03293</td>
<td>0.00472</td>
</tr>
</tbody>
</table>

*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 BIC 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 BIC statistics. If the sample size would have been larger, the exponential distribution might not have won according to the likelihood-based statistics.
Example 14.5: 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

\[
\begin{align*}
f(x; \Theta, p) &= \sum_{i=1}^{K} p_i f_i(x; \Theta_i) \\
F(x; \Theta, p) &= \sum_{i=1}^{K} p_i F_i(x; \Theta_i)
\end{align*}
\]

where \( \Theta_i \) denotes the parameters of component distribution \( i \) and \( \Theta \) denotes the parameters of the mixture distribution, which is a union of all the \( \Theta_i \) parameters. \( p \) 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 \( \theta_1 \) and \( \theta_2 \) 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 \( p \). 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 \( \Phi_1 \) and \( \Phi_2 \) denote the sets of nonscale parameters of the first and second components, respectively, and \( \Phi \) denotes a union of \( \Phi_1 \) and \( \Phi_2 \). For the mixture to have the scale parameter \( \theta \), 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 \( \Phi' \), \( \Phi'_1 \), and \( \Phi'_2 \) 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 \( \rho \) 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:

\[
\begin{align*}
    f(x; \theta, \rho, \Phi, p) &= \frac{1}{\theta} \left( p f_1\left(\frac{x}{\theta}; \Phi_1\right) + (1 - p) f_2\left(\frac{x}{\theta}; \rho, \Phi_2\right) \right) \\
    F(x; \theta, \rho, \Phi, p) &= p F_1\left(\frac{x}{\theta}; \Phi_1\right) + (1 - p) F_2\left(\frac{x}{\theta}; \rho, \Phi_2\right)
\end{align*}
\]

This can be generalized to a mixture of \( K \) components by introducing the \( K - 1 \) ratio parameters \( \rho_i \) that
relate the scale parameters of each of the \( K \) components to the scale parameter \( \theta \) of the mixture distribution
as follows:

\[
\begin{align*}
    \theta_1 &= \theta \\
    \theta_i &= \rho_i \theta; \ i \in [2, K]
\end{align*}
\]

In order to illustrate this approach, define a mixture of two lognormal distributions by using the following
PDF function:

\[
\begin{align*}
    f(x; \mu, \sigma_1, p_2, \rho_2, \sigma_2) &= \frac{(1 - p_2)}{\sigma_1 x \sqrt{2\pi}} \exp\left( \frac{-(\log(x) - \mu)^2}{2\sigma_1^2} \right) + \\
    &\quad \frac{p_2}{\sigma_2 x \sqrt{2\pi}} \exp\left( \frac{-(\log(x) - \mu - \log(\rho_2))^2}{2\sigma_2^2} \right)
\end{align*}
\]

You can verify that \( \mu \) serves as the log of the scale parameter \( \theta \) (\( \mu = \log(\theta) \)).

The following PROC FCMP steps encode this formulation in a distribution named SLOGNMIX2 for use
with PROC SEVSELECT:

```plaintext
/*- Define mixture of 2 lognormal distributions with a log-scale parameter */
proc fcmp library=sashelp.svrtdist 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;
        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.
           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)));
    endif;
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;
Example 14.5: Defining a Finite Mixture Model That Has a Scale Parameter  ♦  769

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 \( p_2 \) and \( \rho_2 \) 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:
  \[
  F^{-1}(0.5) = \frac{(\exp(\mu_1) + \exp(\mu_2))/2}{\exp(\mu)(1 + \rho_2)/2}
  \]
  Solution of this equation yields the value of \( \mu \) in terms of \( \rho_2 \) 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 \( \sigma_1 \) and \( \sigma_2 \):
  \[
  \sigma_1^2 = 2(\log(m_1) - \mu)
  \]
  \[
  \sigma_2^2 = 2(\log(m_1) - \mu - \log(\rho_2))
  \]
  Note that \( \sigma_1 \) has a valid value only if \( \log(m_1) > \mu \). Among the many possible methods of ensuring this condition, the SLOGNMIX2_PARMINIT subroutine uses the method of doing a linear search over \( \rho_2 \).

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 (15.94) is not equal to the average of the medians of the two components (7.39 and 20.09, respectively):
Chapter 14: The SEVSELECT Procedure

```sas
/*-------- 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 14.5.1, confirms that
the two-component mixture is certainly a better fit to these data than the single lognormal distribution.

Output 14.5.1  Comparison of Fitting One versus Two Lognormal Components to Mixture Data

The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slognmix2</td>
<td>38343*</td>
<td>38353*</td>
<td>38353*</td>
<td>38386*</td>
<td>0.94405*</td>
<td>1.30448*</td>
<td>0.20102*</td>
</tr>
<tr>
<td>Logn</td>
<td>39073</td>
<td>39077</td>
<td>39077</td>
<td>39090</td>
<td>3.57373</td>
<td>23.45814</td>
<td>4.09737</td>
</tr>
</tbody>
</table>

Asterisk (*) denotes the best model in the column.

The detailed results for the SLOGNMIX2 distribution are shown in Output 14.5.2. According to the “Initial
Parameter Values and Bounds” table, the initial value of ρ2 is not 0.5, indicating that a linear search was
conducted to ensure log(m1) > μ.
Example 14.6: Scale Regression with Rich Regression Effects

**Output 14.5.2** Detailed Estimation Results for the SLOGNMIX2 Distribution

The SEVSELECT Procedure

**Slognmix2 Distribution**

<table>
<thead>
<tr>
<th>Model Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Distribution Parameters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Parameter Values and Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Mu</td>
</tr>
<tr>
<td>Sigma1</td>
</tr>
<tr>
<td>P2</td>
</tr>
<tr>
<td>Rho2</td>
</tr>
<tr>
<td>Sigma2</td>
</tr>
</tbody>
</table>

**Convergence Status**

Convergence criterion (GCONV=1E-8) satisfied.

**Optimization Summary**

<table>
<thead>
<tr>
<th>Optimization Technique</th>
<th>Trust Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>9</td>
</tr>
<tr>
<td>Function Calls</td>
<td>24</td>
</tr>
<tr>
<td>Log Likelihood</td>
<td>-19171.54807</td>
</tr>
</tbody>
</table>

**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 14.6: 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 47 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:

```bash
/* 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 14.6.1. 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 14.6.1** Class Level Information Table

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>carType</td>
<td>2</td>
<td>SUV, Sedan</td>
</tr>
<tr>
<td>gender</td>
<td>2</td>
<td>Female, Male</td>
</tr>
<tr>
<td>education</td>
<td>3</td>
<td>AdvancedDegree, College, High School</td>
</tr>
</tbody>
</table>

The regression modeling results for the lognormal distribution are shown in Output 14.6.2. 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.
Example 14.6: Scale Regression with Rich Regression Effects

Output 14.6.2 Initial Values for the Scale Regression Model with Class and Interaction Effects

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial Value</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mu</td>
<td>4.85228</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>Sigma</td>
<td>0.52348</td>
<td>1.05367E-8</td>
<td>Infy</td>
</tr>
<tr>
<td>carType SUV</td>
<td>0.54686</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType Sedan</td>
<td>Redundant</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>gender Female</td>
<td>0.34893</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>gender Male</td>
<td>Redundant</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carSafety</td>
<td>-0.63504</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>income</td>
<td>-0.24031</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType SUV * education AdvancedDegree</td>
<td>0.32719</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType SUV * education College</td>
<td>0.68899</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType SUV * education High School</td>
<td>Redundant</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType Sedan * education AdvancedDegree</td>
<td>-0.44650</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType Sedan * education College</td>
<td>-0.26834</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carType Sedan * education High School</td>
<td>Redundant</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>income * gender Female</td>
<td>0.00843</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>income * gender Male</td>
<td>Redundant</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
<tr>
<td>carSafety * income</td>
<td>-0.04744</td>
<td>-709.78271</td>
<td>709.78271</td>
</tr>
</tbody>
</table>

The convergence and optimization summary information in Output 14.6.3 indicates that the scale regression model for the lognormal distribution has converged with the default optimization technique in five iterations.

Output 14.6.3 Optimization Summary for the Scale Regression Model with Class and Interaction Effects

<table>
<thead>
<tr>
<th>Convergence Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence criterion (GCONV=1E-8) satisfied.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Iterations</td>
</tr>
<tr>
<td>Function Calls</td>
</tr>
<tr>
<td>Log Likelihood</td>
</tr>
</tbody>
</table>

The “Parameter Estimates” table in Output 14.6.4 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.
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 14.6.5.
Example 14.7: Scale Regression Model Selection

This example extends “Example 14.6: 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 14.6 to find the best set of scale regression effects for three severity distributions—lognormal (Logn), Burr, and Weibull:

```sas
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 14.7.1.

**Example 14.7: Scale Regression Model Selection**

**Output 14.7.1 Selection Method Settings**

<table>
<thead>
<tr>
<th>The SEVSELECT Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logn Distribution</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Selection Information</td>
</tr>
<tr>
<td>Selection Method</td>
</tr>
<tr>
<td>Select Criterion</td>
</tr>
<tr>
<td>Stop Criterion</td>
</tr>
<tr>
<td>Effect Hierarchy Enforced</td>
</tr>
<tr>
<td>Stop Horizon</td>
</tr>
</tbody>
</table>
For each severity distribution, PROC SEVSELECT reports the summary of the selection process, as shown in Output 14.7.2 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 14.7.2 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 14.7.2** Stepwise Selection Summary for the Lognormal Distribution

<table>
<thead>
<tr>
<th>Selection Summary</th>
<th>Effect Entered</th>
<th>Number Effects In</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step</strong></td>
<td></td>
<td></td>
<td><strong>SBC</strong></td>
</tr>
<tr>
<td>0 Intercept</td>
<td>1</td>
<td>35975.2456</td>
<td></td>
</tr>
<tr>
<td>1 carType*education</td>
<td>2</td>
<td>34121.0008</td>
<td></td>
</tr>
<tr>
<td>2 carSafety</td>
<td>4</td>
<td>33709.8419</td>
<td></td>
</tr>
<tr>
<td>3 income*gender</td>
<td>5</td>
<td>33298.5871</td>
<td></td>
</tr>
<tr>
<td>4 gender</td>
<td>*</td>
<td>33159.7903*</td>
<td></td>
</tr>
</tbody>
</table>

* Optimal Value Of Criterion

Stepwise selection stopped because adding or removing an effect does not improve the SBC criterion.

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 14.7.3. 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 14.7.3  Stepwise Selection Summary for the Weibull Distribution

The SEVSELECT Procedure

Weibull Distribution Selection Details

<table>
<thead>
<tr>
<th>Step Entered</th>
<th>Effect Entered</th>
<th>Effect Removed</th>
<th>Number Effects In</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Intercept</td>
<td></td>
<td>1</td>
<td>36381.5042</td>
</tr>
<tr>
<td>1</td>
<td>carType*education</td>
<td></td>
<td>2</td>
<td>34477.5474</td>
</tr>
<tr>
<td>2</td>
<td>carSafety*income</td>
<td></td>
<td>3</td>
<td>34092.5186</td>
</tr>
<tr>
<td>3</td>
<td>gender</td>
<td></td>
<td>4</td>
<td>33684.7598</td>
</tr>
<tr>
<td>4</td>
<td>carSafety</td>
<td></td>
<td>5</td>
<td>33573.1854</td>
</tr>
<tr>
<td>5</td>
<td>income</td>
<td></td>
<td>6</td>
<td>33517.6460</td>
</tr>
<tr>
<td>6</td>
<td>carSafety*income</td>
<td></td>
<td>5</td>
<td>33509.5021*</td>
</tr>
</tbody>
</table>

* 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 14.7.4. 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 14.6.4, 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 14.7.4  Selected Scale Regression Model for the Lognormal Distribution

<table>
<thead>
<tr>
<th>Convergence Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence criterion (GCONV=1E-8) satisfied.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization Technique</td>
</tr>
<tr>
<td>Iterations</td>
</tr>
<tr>
<td>Function Calls</td>
</tr>
<tr>
<td>Log Likelihood</td>
</tr>
</tbody>
</table>
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 14.7.5 compares the final models of the three severity distributions—lognormal, Burr, and Weibull.

**Output 14.7.5** Comparison of the Selected Scale Regression Models for Different Severity Distributions

| Parameter                          | DF | Estimate  | Standard Error | t Value | Approx Pr > |t| |
|-----------------------------------|----|-----------|----------------|---------|-------------|---|
| **Lognormal**                     | 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 Sedan * 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      |

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...
Example 14.7: Scale Regression Model Selection

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 \( \text{dist} \) 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 \( \text{dist} \), the \( \text{dist}_F \) 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 \( \text{dist}_{\text{QUANTILE}} \) 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 \( \text{dist}_{\text{MEAN}} \) and \( \text{dist}_{\text{QUANTILE}} \) functions are already defined and available to you in the function library, Sashelp.Svrtdist, that PROC SEVSELECT includes by default.

Output 14.7.6, Output 14.7.7, and Output 14.7.8 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 14.7.6 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.
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 14.7.7 Estimates of the Median for Final Models of All Severity Distributions

<table>
<thead>
<tr>
<th>Obs</th>
<th>gender</th>
<th>income</th>
<th>carType</th>
<th>carSafety</th>
<th>Logn_MEDIAN</th>
<th>Burr_MEDIAN</th>
<th>Weibull_MEDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Male College</td>
<td>1.00896</td>
<td>SUV</td>
<td>0.85747</td>
<td>225.018</td>
<td>228.080</td>
<td>227.398</td>
</tr>
<tr>
<td>2</td>
<td>Male AdvancedDegree</td>
<td>0.51942</td>
<td>Sedan</td>
<td>0.01410</td>
<td>77.354</td>
<td>76.174</td>
<td>80.966</td>
</tr>
<tr>
<td>3</td>
<td>Female College</td>
<td>0.38431</td>
<td>Sedan</td>
<td>0.36374</td>
<td>114.936</td>
<td>112.269</td>
<td>122.104</td>
</tr>
<tr>
<td>4</td>
<td>Male College</td>
<td>0.67508</td>
<td>Sedan</td>
<td>0.14417</td>
<td>80.247</td>
<td>79.211</td>
<td>82.285</td>
</tr>
<tr>
<td>5</td>
<td>Male College</td>
<td>0.70732</td>
<td>SUV</td>
<td>0.84460</td>
<td>253.613</td>
<td>255.198</td>
<td>258.289</td>
</tr>
<tr>
<td>6</td>
<td>Male High School</td>
<td>0.15000</td>
<td>Sedan</td>
<td>0.27925</td>
<td>122.475</td>
<td>121.244</td>
<td>125.136</td>
</tr>
<tr>
<td>7</td>
<td>Male High School</td>
<td>0.47704</td>
<td>SUV</td>
<td>0.15692</td>
<td>229.111</td>
<td>227.094</td>
<td>237.373</td>
</tr>
<tr>
<td>8</td>
<td>Male AdvancedDegree</td>
<td>0.77170</td>
<td>Sedan</td>
<td>0.61586</td>
<td>42.704</td>
<td>42.372</td>
<td>45.001</td>
</tr>
<tr>
<td>9</td>
<td>Female AdvancedDegree</td>
<td>0.60157</td>
<td>Sedan</td>
<td>0.69361</td>
<td>66.162</td>
<td>64.988</td>
<td>71.522</td>
</tr>
<tr>
<td>10</td>
<td>Male AdvancedDegree</td>
<td>0.90709</td>
<td>Sedan</td>
<td>0.00579</td>
<td>67.723</td>
<td>67.313</td>
<td>70.143</td>
</tr>
</tbody>
</table>

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 14.7.8: 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.
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:

```plaintext
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 14.7.9 summarizes the selection settings.
Output 14.7.9 Selection Method Settings for Forward Selection

The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Selection Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection Method</td>
</tr>
<tr>
<td>Select Criterion</td>
</tr>
<tr>
<td>Stop Criterion</td>
</tr>
<tr>
<td>Effect Hierarchy Enforced</td>
</tr>
<tr>
<td>Stop Horizon</td>
</tr>
</tbody>
</table>

The selection summary for the lognormal distribution, shown in Output 14.7.10, 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.
**Example 14.7: Scale Regression Model Selection**

**Output 14.7.10**  Forward Selection Summary for the Lognormal Distribution

The SEVSELECT Procedure

Lognormal Distribution
Selection Details

<table>
<thead>
<tr>
<th>Effect Entered</th>
<th>Step Number</th>
<th>Effects In</th>
<th>AICC</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>0</td>
<td>1</td>
<td>35766.6044</td>
<td>35786.1512</td>
</tr>
<tr>
<td>carSafety</td>
<td>1</td>
<td>2</td>
<td>34230.4771</td>
<td>34256.5378</td>
</tr>
<tr>
<td>2 gender</td>
<td>2</td>
<td>3</td>
<td>33895.0222</td>
<td>33927.5962</td>
</tr>
<tr>
<td>3 income</td>
<td>3</td>
<td>4</td>
<td>33656.7655</td>
<td>33695.8519</td>
</tr>
<tr>
<td>4 education</td>
<td>4</td>
<td>5</td>
<td>33594.6989</td>
<td>33646.8076*</td>
</tr>
<tr>
<td>5 carType*gender</td>
<td>5</td>
<td>6</td>
<td>33591.2653</td>
<td>33649.8839</td>
</tr>
<tr>
<td>6 carSafety*income</td>
<td>6</td>
<td>7</td>
<td>33590.2351*</td>
<td>33655.3629</td>
</tr>
<tr>
<td>7 income*gender</td>
<td>7</td>
<td>8</td>
<td>33591.8039</td>
<td>33663.4401</td>
</tr>
</tbody>
</table>

* Optimal Value Of Criterion

Selection stopped because all effects are in the model.

Selected Effects: Intercept carType gender carType*gender carSafety income education carSafety*income

The selection summary for the Weibull distribution is shown in **Output 14.7.11**. It indicates that the selection stopped at the local minimum of the AICC statistic at step 5 because of the shorter stop horizon.

**Output 14.7.11**  Forward Selection Summary for the Weibull Distribution

The SEVSELECT Procedure

Weibull Distribution
Selection Details

<table>
<thead>
<tr>
<th>Effect Entered</th>
<th>Step Number</th>
<th>Effects In</th>
<th>AICC</th>
<th>SBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>0</td>
<td>1</td>
<td>36202.4284</td>
<td>36221.9751</td>
</tr>
<tr>
<td>carSafety</td>
<td>1</td>
<td>2</td>
<td>34560.1129</td>
<td>34586.1736</td>
</tr>
<tr>
<td>2 gender</td>
<td>2</td>
<td>3</td>
<td>34200.9756</td>
<td>34233.5496</td>
</tr>
<tr>
<td>3 income</td>
<td>3</td>
<td>4</td>
<td>33935.3440</td>
<td>33974.4303</td>
</tr>
<tr>
<td>4 education</td>
<td>4</td>
<td>5</td>
<td>33857.1156</td>
<td>33909.2243*</td>
</tr>
<tr>
<td>5 carType*gender</td>
<td>5</td>
<td>6</td>
<td>33856.5340*</td>
<td>33915.1526</td>
</tr>
<tr>
<td>6 carSafety*income</td>
<td>6</td>
<td>7</td>
<td>33857.1143</td>
<td>33922.2421</td>
</tr>
</tbody>
</table>

* Optimal Value Of Criterion

Selection stopped at a local minimum of the AICC criterion.
As the results in Output 14.7.10 and Output 14.7.11 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 14.7.12 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 696.
The fit summary of the final model for the Weibull distribution is shown in Output 14.7.13. 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 14.7.13** Selected Scale Regression Model for the Weibull Distribution

| 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       |

**Output 14.7.12 continued**

| 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 14.8: 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 $\Theta$, 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 14.2: Defining a Model for Mixed-Tail Distributions” on page 752. The input sample is the same one that is used in that example.

```plaintext
/*--- 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;

  * Cramér-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 set 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 14.8.1. 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 14.8.1 with those of Output 14.2.1 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.
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.SvrtDist 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.
Chapter 14: The SEVSELECT Procedure

/******* Definitions distribution functions that compute the mean *********/
proc fcmp library=sashelp.svrtdist 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));
  ends;
  function EXP_MEAN(x, Theta);
    return (Theta);
  ends;
  function GAMMA_MEAN(x, Theta, Alpha);
    return (Theta*Alpha);
  ends;
  function GPD_MEAN(x, Theta, Xi);
    if not(Xi < 1) then
      return (.); /* first moment does not exist */
    return (Theta/(1 - Xi));
  ends;
  function IGAUSS_MEAN(x, Theta, Alpha);
    return (Theta);
  ends;
  function LOGN_MEAN(x, Mu, Sigma);
    return (exp(Mu + Sigma*Sigma/2.0));
  ends;
  function PARETO_MEAN(x, Theta, Alpha);
    if not(Alpha > 1) then
      return (.); /* first moment does not exist */
    return (Theta/(Alpha - 1));
  ends;
  function STWEEDIE_MEAN(x, Theta, Lambda, P);
    return (Theta* Lambda * (2 - P) / (P - 1));
  ends;
  function TWEEDIE_MEAN(x, P, Mu, Phi);
    return (Mu);
  ends;
  function WEIBULL_MEAN(x, Theta, Tau);
    return (Theta*gamma(1 + 1/Tau));
  ends;
quit;

For your further convenience, the dist_QUANTILE function (which computes the quantile of the dist distribution) is also defined in the Sashelp.Svrtdist 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 727, 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);
Example 14.9: Predicting Mean and Value-at-Risk by Using Scoring Functions

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.svrtdist work.scorefuncs)

PROC SEVSELECT detects all the distribution functions that are available in the current CMPLIB= search
path (which always includes the Sashelp.Svrtdist 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_ + ( 7.6472238665070E-01 * x(1));
    _logscale_= _logscale_ + ( 2.9920532114045E+00 * x(2));
    _logscale_= _logscale_ + (-1.0078910539257E+00 * x(3));
    _logscale_= _logscale_ + ( 2.5883411376730E-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:

```sas
%let varLevel=0.975;
/*--- Compute scores (mean and var) for the ---
--- scoring data by using the scoring functions -----*/
```

The “All Fit Statistics” table in Output 14.9.1 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:

```sas
/*--- 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 14.9.1 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:

```sas
/*--- Set VaR level ---*/
%let varLevel=0.975;
```
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 14.9.1  Comparison of Fitted Scale Models for Mean and VaR Illustration

The SEVSELECT Procedure

<table>
<thead>
<tr>
<th>Distribution</th>
<th>-2 Log Likelihood</th>
<th>AIC</th>
<th>AICC</th>
<th>SBC</th>
<th>KS</th>
<th>AD</th>
<th>CvM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp</td>
<td>1515</td>
<td>1527</td>
<td>1527</td>
<td>1552</td>
<td>5.18217</td>
<td>696.65946*</td>
<td>7.97923</td>
</tr>
<tr>
<td>Gamma</td>
<td>448.28222</td>
<td>462.28222</td>
<td>462.50986</td>
<td>491.78448</td>
<td>6.13516</td>
<td>2014</td>
<td>12.46485</td>
</tr>
<tr>
<td>Igauss</td>
<td>444.44512</td>
<td>458.44512</td>
<td>458.67276</td>
<td>487.94738</td>
<td>6.15964</td>
<td>2073</td>
<td>12.45994</td>
</tr>
<tr>
<td>Logn</td>
<td>444.43670*</td>
<td>458.43670*</td>
<td>458.66434*</td>
<td>487.93895*</td>
<td>6.15245</td>
<td>2042</td>
<td>12.47822</td>
</tr>
<tr>
<td>Pareto</td>
<td>1515</td>
<td>1529</td>
<td>1530</td>
<td>1559</td>
<td>5.21366</td>
<td>701.36259</td>
<td>8.00820</td>
</tr>
<tr>
<td>Gpd</td>
<td>1515</td>
<td>1529</td>
<td>1529</td>
<td>1559</td>
<td>5.18217*</td>
<td>696.66044</td>
<td>7.97923*</td>
</tr>
<tr>
<td>Weibull</td>
<td>527.28676</td>
<td>541.28676</td>
<td>541.51440</td>
<td>570.78902</td>
<td>6.06493</td>
<td>1635</td>
<td>12.13031</td>
</tr>
</tbody>
</table>

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:

```sas
/**** 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_);
      _x_(_i_) = 1 to dim(_x_);
      if (_x_(_i_) = .R) then _xparmIgauss(_i_) = 0;
      else _xparmIgauss(_i_) = _x_(_i_);
    end;
  ```
Chapter 14: The SEVSELECT Procedure

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 14.9.2 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 14.9.2 Comparison of Mean and VaR Estimates of Two Scoring Methods

The COMPARE Procedure
Comparison of WORK.SCORESWITHOUTEST with WORK.SCORES
(Method=RELATIVE(0.022), Criterion=1.0E-12)

NOTE: All values compared are within the equality criterion used. However, 34 of the values compared are not exactly equal.
References


# Chapter 15
The TSMODEL Procedure

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: TSMODEL Procedure</td>
<td>798</td>
</tr>
<tr>
<td>Comparison of the TSMODEL and TIMESERIES Procedures</td>
<td>798</td>
</tr>
<tr>
<td>Comparison of the TSMODEL and TIMEDATA Procedures</td>
<td>798</td>
</tr>
<tr>
<td>Using CAS Sessions and CAS Engine Librefs</td>
<td>801</td>
</tr>
<tr>
<td>Getting Started: TSMODEL Procedure</td>
<td>802</td>
</tr>
<tr>
<td>Syntax: TSMODEL Procedure</td>
<td>804</td>
</tr>
<tr>
<td>Functional Summary</td>
<td>805</td>
</tr>
<tr>
<td>PROC TSMODEL Statement</td>
<td>806</td>
</tr>
<tr>
<td>BY Statement</td>
<td>810</td>
</tr>
<tr>
<td>ID Statement</td>
<td>811</td>
</tr>
<tr>
<td>INSCALARS Statement</td>
<td>817</td>
</tr>
<tr>
<td>OUTARRAYS Statement</td>
<td>818</td>
</tr>
<tr>
<td>OUTSCALARS Statement</td>
<td>818</td>
</tr>
<tr>
<td>PRINT Statement</td>
<td>818</td>
</tr>
<tr>
<td>VAR Statement</td>
<td>819</td>
</tr>
<tr>
<td>REQUIRE Statement</td>
<td>819</td>
</tr>
<tr>
<td>SUBMIT Statement</td>
<td>820</td>
</tr>
<tr>
<td>ENDSUBMIT Statement</td>
<td>821</td>
</tr>
<tr>
<td>SAS Programming Statements</td>
<td>821</td>
</tr>
<tr>
<td>Details: TSMODEL Procedure</td>
<td>821</td>
</tr>
<tr>
<td>Accumulation</td>
<td>822</td>
</tr>
<tr>
<td>Missing Value Interpretation</td>
<td>823</td>
</tr>
<tr>
<td>Summary Statistics</td>
<td>823</td>
</tr>
<tr>
<td>SAS Programming Statements</td>
<td>823</td>
</tr>
<tr>
<td>Predefined Symbols</td>
<td>824</td>
</tr>
<tr>
<td>Auxiliary Tables</td>
<td>824</td>
</tr>
<tr>
<td>Table Output</td>
<td>828</td>
</tr>
<tr>
<td>Printed Output</td>
<td>831</td>
</tr>
<tr>
<td>Examples: TSMODEL Procedure</td>
<td>832</td>
</tr>
<tr>
<td>Example 15.1: Accumulating Transactional Data into Time Series Data</td>
<td>832</td>
</tr>
<tr>
<td>Example 15.2: User-Defined Program Statements</td>
<td>834</td>
</tr>
<tr>
<td>Example 15.3: Using Auxiliary Tables</td>
<td>835</td>
</tr>
<tr>
<td>References</td>
<td>838</td>
</tr>
</tbody>
</table>
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:

```plaintext
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:

```plaintext
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 15.1 shows the differences between the processing of programming statements in PROC TSMODEL compared to PROC TIMEDATA.

### Table 15.1 Comparison of TSMODEL and TIMEDATA

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>TSMODEL</th>
<th>TIMEDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement parsing</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>PROC FCMP subroutines and functions</td>
<td>Does not support as many built-in PROC FCMP functions and subroutines.</td>
<td>Supports built-in PROC FCMP functions and subroutines.</td>
</tr>
<tr>
<td>RUN_MACRO</td>
<td>Not supported because it cannot be performed by the CAS server when it executes your program statements in the context of your CAS session.</td>
<td>Can call the PROC FCMP subroutine RUN_MACRO to run a SAS macro, which can then call a SAS procedure.</td>
</tr>
<tr>
<td>ID statement with a SAS date or datetime variable</td>
<td>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.</td>
<td>An ID statement is not required.</td>
</tr>
<tr>
<td>Custom intervals</td>
<td>Custom intervals are not supported at this time.</td>
<td>Custom intervals are supported.</td>
</tr>
</tbody>
</table>
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.”

---

**Table 15.1 continued**

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>TSMODEL</th>
<th>TIMEDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulation</td>
<td>ACCUMULATE=NULL, FIRST, LAST, NOBS, and MEDIAN are supported.</td>
<td>ACCUMULATE=NULL, FIRST, LAST, NOBS, and MEDIAN are supported.</td>
</tr>
<tr>
<td>modes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output data</td>
<td>Can be printed and plotted through other SAS procedures.</td>
<td>Supports PRINT and PLOT options to display data.</td>
</tr>
</tbody>
</table>
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.

```sas
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;
        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 15.1.
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 15.2.

```plaintext
proc sgplot data=mycas.arrays(where=(customer='Bill'));
  series x=Date y=balance;
run;
```
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 15.2 summarizes the statements and options that control the TSMODEL procedure.

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies BY-group processing</td>
<td>BY</td>
<td></td>
</tr>
<tr>
<td>Specifies variables to analyze</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>Specifies the time ID variable</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>Specifies the arrays to output</td>
<td>OUTARRAYS</td>
<td></td>
</tr>
<tr>
<td>Specifies the scalars to output</td>
<td>OUTSCALARS</td>
<td></td>
</tr>
<tr>
<td>Specifies the scalars that are input</td>
<td>INSCALARS</td>
<td></td>
</tr>
<tr>
<td>Specifies the packages to include</td>
<td>REQUIRE</td>
<td></td>
</tr>
<tr>
<td>Specifies the beginning of program statements</td>
<td>SUBMIT</td>
<td></td>
</tr>
<tr>
<td>Specifies the end of program statements</td>
<td>ENDSUBMIT</td>
<td></td>
</tr>
<tr>
<td>Print results from program execution</td>
<td>PRINT</td>
<td></td>
</tr>
<tr>
<td><strong>Table Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specifies the auxiliary input tables</td>
<td>PROC TSMODEL</td>
<td>AUXDATA=</td>
</tr>
<tr>
<td>Specifies the input table</td>
<td>PROC TSMODEL</td>
<td>DATA=</td>
</tr>
<tr>
<td>Specifies the output table</td>
<td>PROC TSMODEL</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specifies the array output table</td>
<td>PROC TSMODEL</td>
<td>OUTARRAY=</td>
</tr>
<tr>
<td>Specifies the scalar output table</td>
<td>PROC TSMODEL</td>
<td>OUTSCALAR=</td>
</tr>
<tr>
<td>Specifies the summary statistics output table</td>
<td>PROC TSMODEL</td>
<td>OUTSUM=</td>
</tr>
<tr>
<td>Specifies the table to contain the BY-group message output log</td>
<td>PROC TSMODEL</td>
<td>OUTLOG=</td>
</tr>
<tr>
<td>Specifies whether messages are output to the OUTLOG= table by severity</td>
<td>PROC TSMODEL</td>
<td>LOGCONTROL=</td>
</tr>
<tr>
<td>Specifies the input table for scalar variables</td>
<td>PROC TSMODEL</td>
<td>INSCALAR=</td>
</tr>
<tr>
<td>Specifies a collector object for an output table</td>
<td>PROC TSMODEL</td>
<td>OUTOBJ=</td>
</tr>
<tr>
<td>Specifies a repeater object for an input table</td>
<td>PROC TSMODEL</td>
<td>INOBJ=</td>
</tr>
</tbody>
</table>

**Accumulation and Seasonality Options**

| Specifies the accumulation frequency | ID   | INTERVAL= |
| Specifies the length of the seasonal cycle   | PROC TSMODEL | SEASONALITY= |
**Table 15.2 continued**

<table>
<thead>
<tr>
<th>Description</th>
<th>Statement</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the interval alignment</td>
<td>ID</td>
<td>ALIGN=</td>
</tr>
<tr>
<td>Specifies the starting time ID value</td>
<td>ID</td>
<td>START=</td>
</tr>
<tr>
<td>Specifies the ending time ID value</td>
<td>ID</td>
<td>END=</td>
</tr>
<tr>
<td>Specifies the accumulation statistic ID</td>
<td>ID, VAR</td>
<td>ACCUMULATE=</td>
</tr>
<tr>
<td>Specifies how to interpret missing values ID</td>
<td>ID, VAR</td>
<td>SETMISSING=</td>
</tr>
<tr>
<td>Specifies the method for trimming BY groups</td>
<td>ID</td>
<td>TRIMID=</td>
</tr>
</tbody>
</table>

**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
  
  PROC TSMODEL 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 801. For more information, see the section “Auxiliary Tables” on page 824.

**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

*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 801.

*data-table* specifies the name of the input data table.
\textbf{INOBJ=}(\textit{object-name=}\textit{CAS-libref.data-table} ...) specifies pairs, each of which binds a repeater object specified by \textit{object-name} with an input table specified by \textit{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:

\begin{verbatim}
inobj=(inest=mycas.outest
    inspec=mycas.outspec)
\end{verbatim}

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 \texttt{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 \texttt{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 \textit{SAS Visual Forecasting: Time Series Packages}. For more information about package access, see the section \enquote{REQUIRE Statement} on page 819.

\textbf{INSCALAR=}\textit{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.

\textit{CAS-libref.data-table} is a two-level name, where \textit{CAS-libref} refers to the \texttt{caslib} and session identifier, and \textit{data-table} specifies the name of the input data table. For more information about this two-level name, see the \texttt{DATA=} option and the section \enquote{Using CAS Sessions and CAS Engine Librefs} on page 801.

When you specify BY variables for the \texttt{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 \texttt{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 \texttt{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 \texttt{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 801.

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 801.

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 801.

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 819.

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 801.

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 801.

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 828.
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:
**Chapter 15: The TSMODEL Procedure**

**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.

**HOUR**
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 822.

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.

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 not specified, 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.

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:
interprets a missing value as having the value \( n \). You can specify any number for \( n \), 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 15.1 and Output 15.1.1.

---

**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:

\[ \text{var-list defines the variables in the var-list as numeric scalars.} \]
\[ \text{var-list $n$ defines the variables in the var-list as character scalars with length } n. \]

The following statement defines scalar variables rc, holdout, and nchanges as numeric scalars, and defines selected as character with length 32.

\[ \text{outscalar rc selected $32$ holdout nchanges; } \]

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:

\[ \text{PRINT OUTLOG < WHERE where-clause> ; } \]
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:

```sas
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, see SAS Visual Forecasting: Time Series Packages.

---

**SUBMIT Statement**

```sas
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 798. The use of SUBMIT and ENDSUBMIT statements to delineate the program block is a major difference between PROC TSMODEL and PROC TIMEDATA.

You can specify the following `submit-options`:

- **FILE=** 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:
Chapter 15: The TSMODEL Procedure

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:

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>19MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>19MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>11MAY1999</td>
<td>50</td>
</tr>
<tr>
<td>12MAY1999</td>
<td>20</td>
</tr>
<tr>
<td>23MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>

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
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>40</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>90</td>
</tr>
</tbody>
</table>

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>20</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>30</td>
</tr>
</tbody>
</table>

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is
  
<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>10</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>20</td>
</tr>
</tbody>
</table>
• If the ACCUMULATE=MAMAXIMUM option is specified, the resulting time series is

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>30</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>50</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01MAR1999</td>
<td>14.14</td>
</tr>
<tr>
<td>01APR1999</td>
<td>.</td>
</tr>
<tr>
<td>01MAY1999</td>
<td>17.32</td>
</tr>
</tbody>
</table>

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 821. For more information, see the “FCMP Procedure” chapter in the *SAS Visual Data Management and Utility 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

_FORMAT_ time format, which is either implied by the INTERVAL= option or specified in the FORMAT= option in the ID statement

_HORIZON_ the time ID value that is one period beyond _TEND_

_INTERVAL_ time interval, which is specified in the INTERVAL= option in the ID statement

_LEAD_ forecast horizon or lead, which is specified in the LEAD= option in the PROC TSMODEL statement

_LENGTH_ length of the time series that is associated with the current BY group

_SERIES_ 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 _SERIES_ might not be unique and could also vary for multiple executions of the procedure.

_SEASONALITY_ length of the seasonal cycle, which is specified in the SEASONALITY= option in the PROC TSMODEL statement or implied by the INTERVAL= option in the ID statement

_TEND_ the last time ID value over all the BY groups. _TEND_ is independent of the value of the TRIMID= option in the ID statement, but the value of _TEND_ is the same as the ending value of the output time ID variable span if you specify TRIMID=NONE.

_TSTART_ the first time ID value over all the BY groups. _TSTART_ is independent of the value of the TRIMID= option in the ID statement, but the value of _TSTART_ is the same as the starting value of the output time ID variable span if you specify TRIMID=NONE.

Predefined Array Values

_SEASON_ season index values

_CYCLES_ 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.
- $t^b_P$ denotes the beginning time ID of the primary (DATA) series.
- $t^e_P$ denotes the ending time ID of the primary (DATA) series.
- $t^b_A$ denotes the beginning time ID of the AUXDATA series.
- $t^e_A$ denotes the ending time ID of the AUXDATA series.
- $[t^b_P, t^e_P]$ denotes the time span for the primary (DATA) series (also known as the reference time span).
- $[t^b_A, t^e_A]$ denotes the time span for the AUXDATA series.
**Case 1:**

\[
\text{DATA} \quad t^b_P \quad \quad \quad \quad \quad t^e_P
\]

\[
\text{AUXDATA} \quad t^b_A \quad \quad \quad \quad \quad t^e_A
\]

In this case, \([t^b_P, t^e_P] \subseteq [t^b_A, t^e_A]\): the auxiliary time span includes the reference span as a subset. Values in the AUXDATA series to the left of \(t^b_P\) and values to the right of \(t^e_P\) are truncated from the AUXDATA series. AUXDATA series values in \([t^b_P, t^e_P]\) are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

**Case 2:**

\[
\text{DATA} \quad t^b_P \quad \quad \quad \quad \quad t^e_P
\]

\[
\text{AUXDATA} \quad t^b_A \quad \quad \quad \quad \quad t^e_A
\]

In this case, \([t^b_P, t^e_P] = [t^b_P, t^b_A] \cup [t^b_A, t^e_P]\): the reference time span leads the auxiliary time span with a non-empty intersection. AUXDATA series values in \([t^b_P, t^b_A]\) are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in \([t^b_A, t^e_P]\) are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in \((t^e_P, t^e_A]\) are truncated.

**Case 3:**

\[
\text{DATA} \quad t^b_P \quad \quad \quad \quad \quad t^e_P
\]

\[
\text{AUXDATA} \quad t^b_A \quad \quad \quad \quad \quad t^e_A
\]

In this case, \([t^b_P, t^e_P] = [t^b_P, t^b_A] \cup (t^e_A, t^e_P]\): the reference time span lags the auxiliary time span with a non-empty intersection. AUXDATA series values in \([t^b_P, t^e_A]\) are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in \((t^b_A, t^e_P]\) are truncated.
in \((t^b_A, t^e_A]\) are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in \((t^b_b, t^e_b]\) are truncated.

**Case 4:**

\[\begin{array}{c}
\text{DATA} \\
\hline
\quad \quad t^b_P \\
\quad \quad t^e_P \\
\hline
\text{AUXDATA} \\
\hline
\quad \quad t^b_A \\
\quad \quad t^e_A \\
\end{array}\]

In this case, \([t^b_P, t^e_P]\) \subset [t^b_b, t^e_b]: the auxiliary time span is a subset of the reference time span. AUXDATA series values in \([t^b_b, t^b_a]\) and values in \((t^e_a, t^e_b]\) are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in \([t^b_A, t^e_A]\) are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

**Case 5:**

\[\begin{array}{c}
\text{DATA} \\
\hline
\quad \quad t^b_P \\
\quad \quad t^e_P \\
\hline
\text{AUXDATA} \\
\hline
\quad \quad t^b_A \\
\quad \quad t^e_A \\
\end{array}\]

In this case, \([t^b_P, t^e_P]\) \∩ [t^b_A, t^e_A] = \∅: the auxiliary time span does not intersect the reference time span at all. AUXDATA series values in \([t^b_P, t^e_P]\) 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 798. 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:

(ERRNO) a numeric variable that stores the _ERRNO_ variable for the BY group. The value of
(ERRNO) might be set by the user-defined program directly or might be set implicitly
by calling a function or method that sets the _ERRNO_ value.
(LOGLEN) a numeric variable that stores the length of the _LOG_ variable text (byte count).
(LOG) 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 PUTTOLOG=YES is specified in the PROC TSMODEL
statement, the _LOG_ 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.
Example 15.1: 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.

```latex
proc tsmodele 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 15.1.1.
Example 15.1: Accumulating Transactional Data into Time Series Data

Output 15.1.1 Summary of Accumulation Processing

The TSMODEL Procedure

<table>
<thead>
<tr>
<th>Summary of time series processing for RETAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of analysis variables</td>
</tr>
<tr>
<td>Number of rows read</td>
</tr>
<tr>
<td>Number of groups read</td>
</tr>
<tr>
<td>Memory for group packages (KB)</td>
</tr>
<tr>
<td>Time to load groups (seconds)</td>
</tr>
<tr>
<td>Minimum time ID</td>
</tr>
<tr>
<td>Maximum time ID</td>
</tr>
<tr>
<td>Minimum time periods</td>
</tr>
<tr>
<td>Maximum time periods</td>
</tr>
<tr>
<td>Number of nodes run</td>
</tr>
<tr>
<td>Number of nodes with data</td>
</tr>
<tr>
<td>Number of nodes with groups</td>
</tr>
<tr>
<td>Number of threads budgeted</td>
</tr>
<tr>
<td>Minimum thread group count</td>
</tr>
<tr>
<td>Maximum thread group count</td>
</tr>
<tr>
<td>Minimum threads active</td>
</tr>
<tr>
<td>Maximum threads active</td>
</tr>
<tr>
<td>Number of groups processed by submitted code</td>
</tr>
<tr>
<td>Number of groups failing</td>
</tr>
<tr>
<td>Elapsed time to process groups (seconds)</td>
</tr>
<tr>
<td>Number of output table rows produced</td>
</tr>
</tbody>
</table>

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:

```plaintext
proc tsmodel 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 15.2: 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.

```sas
%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.

```sas
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.
Example 15.3: 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 following PROC PRINT step displays the OUTSUM= table, as shown in Output 15.2.1.

proc print data=mycas.airsum; run;

Output 15.2.1 Summary Statistics for mycas.Air

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>NAME</em></th>
<th>NOBS</th>
<th>N</th>
<th>NMISS</th>
<th>MIN</th>
<th>MAX</th>
<th>MEAN</th>
<th>STDDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AIR</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>341</td>
<td>1736</td>
<td>840.8958333</td>
<td>356.43144067</td>
</tr>
<tr>
<td>2</td>
<td>logair</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>5.8318824773</td>
<td>7.4593388952</td>
<td>6.6430769906</td>
<td>0.4395319391</td>
</tr>
<tr>
<td>3</td>
<td>logairc</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>-835.0639509</td>
<td>-833.4364944</td>
<td>-834.2527563</td>
<td>0.4395319394</td>
</tr>
</tbody>
</table>

Example 15.3: Using Auxiliary Tables

The following PROC PRINT step displays the OUTSUM= table, as shown in Output 15.2.1.

proc print data=mycas.airsum; run;

Output 15.2.1 Summary Statistics for mycas.Air

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>NAME</em></th>
<th>NOBS</th>
<th>N</th>
<th>NMISS</th>
<th>MIN</th>
<th>MAX</th>
<th>MEAN</th>
<th>STDDEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AIR</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>341</td>
<td>1736</td>
<td>840.8958333</td>
<td>356.43144067</td>
</tr>
<tr>
<td>2</td>
<td>logair</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>5.8318824773</td>
<td>7.4593388952</td>
<td>6.6430769906</td>
<td>0.4395319391</td>
</tr>
<tr>
<td>3</td>
<td>logairc</td>
<td>48</td>
<td>48</td>
<td>0</td>
<td>-835.0639509</td>
<td>-833.4364944</td>
<td>-834.2527563</td>
<td>0.4395319394</td>
</tr>
</tbody>
</table>
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:

```sas
proc tsmodel 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:

```sas
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.

```sas
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 15.3.1.

```sas
proc tsmode 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 15.3.1** Validation of Oil and Gas Shares by Region

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
<td><em>STATUS</em></td>
<td>oilsum</td>
<td>gassum</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>123</td>
<td>123</td>
</tr>
</tbody>
</table>

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 15.3.2.

```sas
proc sgplot data=mycas.shares;
    series x=Date y=OilShare/group=RegionName;
run;
```
Output 15.3.2 Region Share of Oil Production

References


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: Each function in this chapter has a prefix of “TSA.”; however, the prefixes are omitted in descriptions for better readability. 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: Language Reference.
### Functional Summary

Table 16.1 summarizes the functions in the TSA package.

**Table 16.1 Functions in Time Series Analysis Package**

<table>
<thead>
<tr>
<th>TSA Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCUMULATE</td>
<td>Accumulates a univariate time series to a particular frequency</td>
</tr>
<tr>
<td>ACCUMULATE2</td>
<td>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</td>
</tr>
<tr>
<td>ACF</td>
<td>Computes autocorrelation and autocovariance for a time series array</td>
</tr>
<tr>
<td>ARMAORDERS</td>
<td>Performs tests to tentatively identify the autoregressive and moving average orders of mixed autoregressive moving average models</td>
</tr>
<tr>
<td>CCF</td>
<td>Computes the cross-correlation and cross-covariance for two time series arrays</td>
</tr>
<tr>
<td>DPF Class Object</td>
<td>Performs count distribution analysis for time series</td>
</tr>
<tr>
<td>FREQ Class Object</td>
<td>Performs frequency analysis of a time series</td>
</tr>
<tr>
<td>INTERMITTENCYTEST</td>
<td>Tests for intermittency of a univariate time series</td>
</tr>
<tr>
<td>IACF</td>
<td>Computes the inverse autocorrelation for a time series array</td>
</tr>
<tr>
<td>MOVINGSUMMARY</td>
<td>Computes statistics for a set of values within a moving time window</td>
</tr>
<tr>
<td>PACF</td>
<td>Computes the partial autocorrelation for a time series array</td>
</tr>
<tr>
<td>SCALE</td>
<td>Scales a time series between the minimum value and the maximum value of the original time series</td>
</tr>
<tr>
<td>SEASONALDECOMP</td>
<td>Computes the seasonal indices of a univariate time series using classical decomposition</td>
</tr>
<tr>
<td>SEASONALINDICES</td>
<td>Computes the seasonal indices of a univariate time series by using regression seasonal dummies</td>
</tr>
<tr>
<td>SEASONTEST</td>
<td>Tests for seasonality of a univariate time series</td>
</tr>
<tr>
<td>SIMILARITY</td>
<td>Performs similarity analysis for time series</td>
</tr>
<tr>
<td>STATIONARITYTEST</td>
<td>Tests for stationarity of a univariate time series</td>
</tr>
<tr>
<td>TRANSFORM</td>
<td>Transforms time series according to the specified transformation type</td>
</tr>
<tr>
<td>UNBIASEDNESS</td>
<td>Tests whether a univariate time series is unbiased</td>
</tr>
<tr>
<td>WHITENOISE</td>
<td>Tests for white noise of a time series array</td>
</tr>
</tbody>
</table>
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: Language Reference*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

ACCUMULATE Function

```
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:
ACCUMULATE Function

- **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:

  
<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

- **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:

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 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 Function

\[
rc = TSA.ACCUMULATE2 \ (time, y, \ 'interval', \ id, \ z, \ <'accumulate'>, \ <'setmiss'>, \ <'zeromiss'>) ;
\]

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:

- \( \text{DAY} \) specifies a seasonal cycle of length 7.
- \( \text{MONTH} \) specifies a seasonal cycle of length 12.
- \( \text{QTR} \) 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:
ACCUMULATE2 Function

**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:
  - $rc = 0$: Success.
  - $rc < 0$: 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:

```plaintext
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 Function

$$rc = TSA.ACF (y, nlag, lags, df, < mu>, < acov>, < acf>, < acfstd>, < acfnorm>, < acfprob>
                 < acf1prob>);$$

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:
  
<table>
<thead>
<tr>
<th>$rc$</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

- $lags$ 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:

```plaintext
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 Function

\[ rc = TSA.\text{ARMAORDERS}(y, <\text{dif}>, <'\text{method}'>, <\text{p}>, <\text{q}>, <\text{perror}>, \text{porders}, \text{qorders}); \]

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.

\[ \text{dif} \]

specifies either an array of positive integers or a positive integer that is used for differencing. The default value is 0.

\[ '\text{method}' \]

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.

\[ \text{p} \]

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].

\[ \text{q} \]

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].

\[ \text{perror} \]

specifies the autoregressive orders used to estimate the error series for the MINIC method, where \( \text{perror} \) is an array of two nonnegative integers that define the minimum and maximum values.

By default, \( \text{perror} \) 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:

<table>
<thead>
<tr>
<th>( rc )</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

- \( porders \) 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 \( \text{Air} \):

```plaintext
proc tsmode 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 Function

\[ rc = \text{TSA.CCF} (y, x, nlag, lags, df, <yimu>, <xmu>, <ccov>, <ccf>, <ccfstd>, <ccf2std>, <ccfnorm>, <ccfprob>, <ccfprob>) \]

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:
  - 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.
- \( ccfstd \) 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.
- \( ccfprob \) 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:

```r
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 ccfprob ccflprob;
  require tsa;
submit;
  declare object TSA(tsa);
  rc=TSA.CCF(price, sale, 20, lags, df, ymu, xmu, ccov, ccf,
            ccfstd, ccf2std, ccfnorm, ccfprob, ccflprob);
endsubmit;
run;
```

DPF Class 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.

Signature

```
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.SetY(claims);
rc = f.SetOption(<'name',value, 'name', value, ...>);
rc = f.Run();
rc = of.Collect(f);
rc = dpf.Initialize(f);
rc = dpf.SetOption(<'name',value, 'name', value, ...>);
rc = dpf.Run();
rc = odpe.Collect(dpf);
rc = odprob.Collect(dpf);
```
Chapter 16: Time Series Analysis Package

**Required Input**

You must specify a FREQ object through the Initialize() statement:

'freq' specifies a FREQ class object, which includes frequency analysis results.

---

**Optional Specifications**

You can optionally specify one or more of the following 'Name', Value pairs, separated by commas, in the SetOption() statement:

- '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 maxiter is an integer.
- 'select' specifies the distribution selection criterion.
  
  You can specify the following values within single quotation marks:
  
  AIC specifies Akaike’s information criterion.
  BIC specifies the Bayesian information criterion.
  LOGLIK specifies the log-likelihood.
  
  The default value is LOGLIK.

- 'method' specifies candidate distribution to use in the analysis.
  
  You can specify the following values within single quotation marks:
  
  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.
Returned Values

The Run() statement returns the following values:

\[ rc \] returns one of the following scalar return codes:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

Output Table Schema

Two collector objects, OUTDPE and OUTDPROB, collect the results of the count distribution analysis. Results can be output to tables using OUTOBJ=() statement.

**OUTDPE** collects parameter estimates information of selected count distribution. The output table contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Content</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>Name of target variable</td>
</tr>
<tr>
<td>Distribution</td>
<td>Distribution</td>
</tr>
<tr>
<td>Parameter</td>
<td>Name of parameter</td>
</tr>
<tr>
<td>Estimate</td>
<td>Parameter estimate</td>
</tr>
<tr>
<td>StdErr</td>
<td>Standard error</td>
</tr>
<tr>
<td>Tvalue</td>
<td>t value</td>
</tr>
<tr>
<td>Probt</td>
<td>Approximate probability &gt;</td>
</tr>
<tr>
<td>Lower</td>
<td>Lower 95% of parameter estimate</td>
</tr>
<tr>
<td>Upper</td>
<td>Upper 95% of parameter estimate</td>
</tr>
</tbody>
</table>

**OUTDPROB** collects predictions using the count distribution that has the best fit. The output table contains the following columns:
### 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:

```sas
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 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.

Signature

```c
declare object f(FREQ);
declare object of(OUTFREQ);
rc = f.Initialize();
rc = f.SetY(claims);
rc = f.SetOption(<'name',value, 'name', value, ...>);
rc = f.Run();
rc = f.GetResult('name',result);
nvalues = f.nvalues();
rc = of.Collect(f);
```

Required Input

You must specify the following input series by using the SetY() statement:

- `y` specifies the time series array to analyze.

When you retrieve results by using the GetResult() statement, you must specify one of the following 'Names' and its corresponding output numeric array `Result`:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUES</td>
<td>retrieves a monotonically increasing list of the unique values found in the input series that was set by using the SetY() statement.</td>
</tr>
<tr>
<td>COUNTS</td>
<td>retrieves the number of occurrences of each unique value.</td>
</tr>
<tr>
<td>PCTS</td>
<td>retrieves the percentage of total occurrences of each unique value.</td>
</tr>
</tbody>
</table>

All results series have the same length, which you can retrieve by using the nvalues() statement.

`Result` specifies a numeric array to receive the result series.

Optional Specification

You can optionally specify the following 'Name', Value pair, in the SetOption() statement:
seasonality specifies the seasonality of the time series, where seasonality is an integer. The default value is 0.

Returned Values

The Run() statement returns the following values:

\( rc \) returns one of the following scalar return codes:

<table>
<thead>
<tr>
<th>( rc )</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful</td>
</tr>
<tr>
<td>1</td>
<td>Binary series</td>
</tr>
<tr>
<td>2</td>
<td>Nonnegative integer series</td>
</tr>
<tr>
<td>3</td>
<td>Integer series</td>
</tr>
<tr>
<td>4</td>
<td>Noninteger series</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

The nvalues() statement returns the following values:

\( nvalues \) the number of unique values in the input series that was set by using the SetY() statement. It is also the length of all result series that are returned by the GetResult() statement. It is set to a missing value if the Run() statement did not complete successfully.

Output Table Schema

The collector object, OUTFREQ, collects the results of frequency function. Results can be output to a table using the OUTOBJ=() statement.

OUTFREQ collects the results of frequency function. The output table contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Content</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>Name of target variable</td>
</tr>
<tr>
<td>Values</td>
<td>Array of series values</td>
</tr>
<tr>
<td>Counts</td>
<td>Array of frequency series counts</td>
</tr>
<tr>
<td>Percent</td>
<td>Percentage of total frequency</td>
</tr>
</tbody>
</table>

Example

This example uses the TSMODEL procedure to analyze the frequency of the time series Air:

```plaintext
proc tsmode 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;

IACF Function

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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

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:

```plaintext
proc tsmodel 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 Function**

```plaintext
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:

<table>
<thead>
<tr>
<th>( rc )</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Time series is not intermittent</td>
</tr>
<tr>
<td>1</td>
<td>Time series is intermittent</td>
</tr>
<tr>
<td>(&lt; 0)</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

\[ med \]
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:

```plaintext
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 Function

\[ rc = TSA.MOVINGSUMMARY \ (y, \ 'method', k, \ < lead >, \ < w >, \ < 'setmiss' >, \ < 'abs' >, \ x, \ < p >, \ < nmiss >) \]

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:

\[ y \]
specifies the input time series array.
specifies the statistic to calculate for each output array, $x_t$, based on the elements of the $y$ input array in the $t$ 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 $t$.

$k$ specifies the window size, where $k$ is a positive integer. When the method is EWMA, $k$ 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.

- **lead** specifies the number of leading terms, where lead is a nonnegative integer less than $k$. You can specify the following values:
  - 0 specifies the backward moving summary.
  - $k/2$ specifies the centered moving summary.
  - $k-1$ 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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Warping limits relaxed</td>
</tr>
<tr>
<td>3</td>
<td>Expansion limits relaxed</td>
</tr>
<tr>
<td>2</td>
<td>Compression limits relaxed</td>
</tr>
<tr>
<td>1</td>
<td>Warping limits imposed</td>
</tr>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

- **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 Function

```
rc = TSA.PACF (y, nlag, lags, df, < mu> , < pacf> , < pacfstd>, < 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:
  - `rc` 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.
- **pacfstd**: 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:

```plaintext
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 Function**

\[ rc = TSA.SC 

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:

- `y` specifies the input time series array.
- `min` specifies the minimum value in the output array.
- `max` specifies the maximum value in the output array.
- `nomiss` 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:

- `rc` returns one of the following scalar return codes:
  - `rc`  
    - `5` The input series is nearly constant
    - `4` Missing values were found when the value 1 was specified for `nomiss`
    - `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:

\[ n\text{miss} \]
returns the number of missing values that are generated.

Example

This example uses the TSMODEL procedure to scale the time series array \[ \text{Air} \] between a minimum value of 0 and a maximum value of 100:

```sas
proc tsmodel 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 Function

\[ rc = \text{TSA.SEASONALDECOMP}(y, s, 'mode', <lambda>, <tcc>, <sic>, <sc>, <scstd>, <tcs>, <ic>, <sa>, <pcsA>, <tc>, <cc>); \]

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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

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.

tcc specifies the trend-cycle component.
sic specifies the seasonal-irregular component.
sc specifies the seasonal component.
scstd specifies the seasonal component standard errors.
tcs specifies the trend-cycle-seasonal component.
ic specifies the irregular component.
sa specifies the seasonally adjusted series.
pcsas specifies the percentage of change in seasonally adjusted series.
tc specifies the trend component.
cc specifies the cycle component.

Example

This example uses the TSMODEL procedure to compute the seasonal indices on the time series array Air:

```sas
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 Function

\( rc = \text{TSA.SEASONALINDICES}(y, s, '<mode>', '<term>', indices) \);

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:

- \( y \) specifies the times series array.
- \( s \) 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.

- \('mode'\) 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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

indices returns an array of seasonal indices.

Example

This example uses the TSMODEL procedure to compute the seasonal indices of the time series Air:

```r
proc tsmodel 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 Function

\[
rc = \text{TSA.SEASONTEST}(y, s, \langle \text{dif} \rangle, \langle p \rangle, \langle \alpha \rangle, \langle \text{aic} \rangle);
\]

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 \(\text{SEASONALITY}\), which is the length of the seasonal cycle as specified by the \(\text{SEASONALITY}=\) option in the PROC TSMODEL statement or implied by the \(\text{INTERVAL=}\) 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.

- \(\text{dif}\) 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:

<table>
<thead>
<tr>
<th>(rc)</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Time series is not seasonal</td>
</tr>
<tr>
<td>1</td>
<td>Time series is seasonal</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>
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:

```sas
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.SEASONTEST(air, _SEASONALITY_, 0, 1, , aic); /*- no detrending -*/
  if rc>0 then seasonal= 1;
  rc=TSA.SEASONTEST(air, _SEASONALITY_, 1, 1, 0.05, ); /*- detrending -*/
  if rc>0 then seasonal= 1;
  endsubmit;
run;
```

SIMILARITY Function

\[ rc = TSA.SIMILARITY (x, y, \langle 'type' \rangle, \langle 'scale' \rangle, \langle expandpct \rangle, \langle expandabs \rangle, \langle compresspct \rangle, \langle compressabs \rangle, measure) ; \]

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.

'\texttt{type}': specifies the similarity measure.

You can specify the following values within single quotation marks:

- \texttt{ABSDEV}: specifies the absolute deviation.
- \texttt{MABSDEV}: specifies the mean absolute deviation.
- \texttt{MABSDEVINP}: specifies the mean absolute deviation relative to the length of the input sequence.
- \texttt{MABSDEVMAX}: specifies the mean absolute deviation relative to the maximum valid path length.
- \texttt{MABSDEVMIN}: specifies the mean absolute deviation relative to the minimum valid path length.
- \texttt{MABSDEVTAR}: specifies the mean absolute deviation relative to the length of the target sequence.
- \texttt{MSQRDEV}: specifies the mean squared deviation.
- \texttt{MSQRDEVINP}: specifies the mean squared deviation relative to the length of the input sequence.
- \texttt{MSQRDEVMAX}: specifies the mean squared deviation relative to the maximum valid path length.
- \texttt{MSQRDEVMIN}: specifies the mean squared deviation relative to the minimum valid path length.
- \texttt{MSQRDEVTAR}: specifies the mean squared deviation relative to the length of the target sequence.
- \texttt{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:

- \texttt{ABS}: applies absolute scaling.
- \texttt{NONE}: applies no scaling.
- \texttt{STD}: applies standard scaling.

The default value is NONE.

\texttt{expandpct}: specifies the warping expansion as a percentage of the length of the target sequence, where \texttt{expandpct} ranges from 0 to 100, 0 implies no compression, and 100 implies maximum allowable compression. The default value is 100.
**Chapter 16: Time Series Analysis Package**

*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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Warping limits relaxed</td>
</tr>
<tr>
<td>3</td>
<td>Expansion limits relaxed</td>
</tr>
<tr>
<td>2</td>
<td>Compression limits relaxed</td>
</tr>
<tr>
<td>1</td>
<td>Warping limits imposed</td>
</tr>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

*measure* returns the similarity measure.

---

**Example**

This example uses the TSMODEL procedure to compute the similarity of two time series arrays: x and y.

```plaintext
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 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;
```
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.

\[ \text{dif}, \text{d}, \text{p}, \text{'type'} \]

- \( \text{dif} \) specifies an array of positive integers or a positive integer that is used for differencing. The default value is 0.
- \( \text{d} \) specifies the order of unit root \( (d = 1, \ldots, 12) \). If the \( \text{type} \) is SSM, then \( d = 1 \). The default value is 1.
- \( \text{p} \) specifies the autoregressive order, where \( p \) must be a nonnegative integer. The default value is 5.
- \( \text{'type'} \) specifies the type of test statistic used.

You can specify the following values within single quotation marks:

- \( \text{SSM} \) specifies the studentized test statistic for the single mean (intercept) case.
- \( \text{STR} \) specifies the studentized test statistic for the deterministic time trend case.
- \( \text{SZM} \) specifies the studentized test statistic for the zero mean (no intercept) case.

This value is allowed only when \( d=1 \).

The default value of \( \text{type} \) is \( \text{SZM} \).
Returned Values

The STATIONARITYTEST function returns the following values:

\[ rc \]

returns one of the following scalar return codes:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Time series is stationary with the default significance level of 0.05</td>
</tr>
<tr>
<td>1</td>
<td>Time series is not stationary with the default significance level of 0.05</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

\[ pvalue \]

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:

```plaintext
proc tsmodel 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 Function

\[ rc = TSA.TRANSFORM (y, '<type>', '<inverse>', '<c>', x); \]

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: \( x = \log(c 	imes y) / (1 - (c \times y)) \). The default value is 1.
- For the Box-Cox transformation, \( c \) is \( \lambda \): \( x = c^2 + (y^c - 1) / c \). 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:

<table>
<thead>
<tr>
<th>rc</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

x returns the transformed series.
**Example**

This example uses the TSMODEL procedure to take the log transform of the time series array *Air*:

```sql
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 Function**

```sql
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:
WHITENOISE Function

\[ rc = \text{TSA.WHITENOISE}(y, nlag, lags, df, wn, <wnprob>, <wnlprob>) \]

The WHITENOISE function tests for white noise in a time series array.

---

### Example

This example uses the TSMODEL procedure to test whether the series Actual is unbiased:

```plaintext
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;
```
Chapter 16: Time Series Analysis Package

Required Arguments

You must specify the following arguments, separated by a comma:

- \( y \) specifies the times series array to compute.
- \( nlag \) 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:

<table>
<thead>
<tr>
<th>( rc )</th>
<th>Termination Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>(&lt; 0)</td>
<td>Computational failure</td>
</tr>
</tbody>
</table>

- \( lags \) 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:

```plaintext
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;
```
# Chapter 17
## Time Series Model Package

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview: TSM Package</td>
<td>880</td>
</tr>
<tr>
<td>TSM Package Summary</td>
<td>880</td>
</tr>
<tr>
<td>Using the TSM Package</td>
<td>883</td>
</tr>
<tr>
<td>Return Codes</td>
<td>885</td>
</tr>
<tr>
<td>TSM Object</td>
<td>885</td>
</tr>
<tr>
<td>TSM Synopsis</td>
<td>886</td>
</tr>
<tr>
<td>TSM Methods</td>
<td>887</td>
</tr>
<tr>
<td>ESMSPEC Object</td>
<td>891</td>
</tr>
<tr>
<td>ESMSPEC Synopsis</td>
<td>892</td>
</tr>
<tr>
<td>ESMSPEC Methods</td>
<td>893</td>
</tr>
<tr>
<td>ARIMASPEC Object</td>
<td>895</td>
</tr>
<tr>
<td>ARIMASPEC Synopsis</td>
<td>898</td>
</tr>
<tr>
<td>ARIMASPEC Methods</td>
<td>898</td>
</tr>
<tr>
<td>EXMSPEC Object</td>
<td>904</td>
</tr>
<tr>
<td>EXMSPEC Synopsis</td>
<td>906</td>
</tr>
<tr>
<td>EXMSPEC Methods</td>
<td>906</td>
</tr>
<tr>
<td>IDMSPEC Object</td>
<td>908</td>
</tr>
<tr>
<td>IDMSPEC Synopsis</td>
<td>909</td>
</tr>
<tr>
<td>IDMSPEC Methods</td>
<td>909</td>
</tr>
<tr>
<td>UCMSPEC Object</td>
<td>913</td>
</tr>
<tr>
<td>UCMSPEC Synopsis</td>
<td>914</td>
</tr>
<tr>
<td>UCMSPEC Methods</td>
<td>915</td>
</tr>
<tr>
<td>TSMPEST Object</td>
<td>920</td>
</tr>
<tr>
<td>TSMPEST Synopsis</td>
<td>921</td>
</tr>
<tr>
<td>TSMPEST Methods</td>
<td>921</td>
</tr>
<tr>
<td>TSMSPEC Object</td>
<td>922</td>
</tr>
<tr>
<td>TSMSPEC Synopsis</td>
<td>923</td>
</tr>
<tr>
<td>TSMSPEC Methods</td>
<td>923</td>
</tr>
<tr>
<td>TSMFOR Object</td>
<td>924</td>
</tr>
<tr>
<td>TSMFOR Synopsis</td>
<td>925</td>
</tr>
<tr>
<td>TSMFOR Methods</td>
<td>926</td>
</tr>
<tr>
<td>TSMSTAT Object</td>
<td>928</td>
</tr>
<tr>
<td>TSMSTAT Synopsis</td>
<td>931</td>
</tr>
<tr>
<td>TSMSTAT Methods</td>
<td>931</td>
</tr>
<tr>
<td>TSMINEST Object</td>
<td>932</td>
</tr>
</tbody>
</table>
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 17.1 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 17.1 Objects in the Time Series Model Package |
|---------------------------------------------------|----------------------------------------------------------|
| **Object**                                        | **Description**                                          |
| Central Object (Note 1)                           |                                                          |
| TSM                                               | Time series model object                                 |
| Time Series Model Specification Objects (Note 2) |                                                          |
| ARIMASPEC                                         | Autoregressive integrated moving average model specification object |
| ESMSPEC                                           | Exponential smoothing model specification object          |
| EXMSPEC                                           | External model specification object                       |
| IDMSPEC                                           | Intermittent demand model specification object            |
| UCMSPEC                                           | Unobserved component model (UCM) specification object     |
Table 17.1 continued

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time Series Model Collector Objects (Note 3)</strong></td>
<td></td>
</tr>
<tr>
<td>TSMFOR</td>
<td>Time series model forecast collector object</td>
</tr>
<tr>
<td>TSMPEST</td>
<td>Time series model parameter estimates collector object</td>
</tr>
<tr>
<td>TSMSPEC</td>
<td>Time series model specification collector object</td>
</tr>
<tr>
<td>TSMSTAT</td>
<td>Time series model fit statistics collector object</td>
</tr>
<tr>
<td><strong>Time Series Model Repeater Objects (Note 4)</strong></td>
<td></td>
</tr>
<tr>
<td>TSMINEST</td>
<td>Time series model input parameter estimates repeater object</td>
</tr>
<tr>
<td>TSMINSPEC</td>
<td>Time series model input specification repeater object</td>
</tr>
</tbody>
</table>

**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 time series model specification objects enable you to specify the characteristics of a time series model to be executed by a TSM object.

3. 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).

4. 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 17.1 diagrams the relationships among the objects in the TSM package.
Figure 17.1 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 Analysis and Forecasting 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.

**Step 3: 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 collects 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 17.2 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 17.2  Return Codes**

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>An unrecoverable error occurred. No result was produced.</td>
</tr>
<tr>
<td>= 0</td>
<td>Unconditional success. The requested action was completed, and a normal result was produced.</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>Conditional success or warning. A result was produced subject to conditions.</td>
</tr>
</tbody>
</table>

TSM Object

The TSM object generates forecasts of univariate time series.

Table 17.3 lists the time series model families that are supported.

**Table 17.3  Model Families for the TSM Object**

<table>
<thead>
<tr>
<th>Family</th>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARIMA</td>
<td>ARIMASPEC</td>
<td>ARIMAX models</td>
</tr>
<tr>
<td>ESM</td>
<td>ESMSPEC</td>
<td>Exponential smoothing models</td>
</tr>
<tr>
<td>EXM</td>
<td>EXMSPEC</td>
<td>External model (external forecast)</td>
</tr>
<tr>
<td>IDM</td>
<td>IDMSPEC</td>
<td>Intermittent demand models (Croston’s/average demand (ADEM))</td>
</tr>
<tr>
<td>UCM</td>
<td>UCMSPEC</td>
<td>Unobserved component models</td>
</tr>
</tbody>
</table>

Table 17.4 summarizes the methods that are associated with the TSM object.
Table 17.4  Methods of the TSM Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddX</td>
<td>Add an independent time series array (XSeries) for the TSM object</td>
</tr>
<tr>
<td>AddExternal</td>
<td>Add external forecast component series for the TSM object</td>
</tr>
<tr>
<td>criterion</td>
<td>Return the final forecast fit statistic</td>
</tr>
<tr>
<td>GetForecast</td>
<td>Get the forecast series</td>
</tr>
<tr>
<td>Initialize</td>
<td>Initialize the TSM object</td>
</tr>
<tr>
<td>nfor</td>
<td>Return the forecast series length</td>
</tr>
<tr>
<td>Replay</td>
<td>Replay the restored model and parameter estimates</td>
</tr>
<tr>
<td>Run</td>
<td>Run the TSM object</td>
</tr>
<tr>
<td>SetOption</td>
<td>Specify the named option for the TSM object</td>
</tr>
<tr>
<td>SetY</td>
<td>Specify the dependent time series array (YSeries) for the TSM object</td>
</tr>
</tbody>
</table>

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();

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 904.

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:

Series specifies a numeric array that contains an external forecast series for the TSM object.

Role 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

\[ rc = \text{obj}.\text{AddX} \left( XSeries \text{ < .Required, NoDiff, ModelSymbol > } \right); \]

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, \( \text{ModelSymbol} \), 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:

\( XSeries \) specifies a numeric array that contains an independent series for the TSM object.

You can also specify the following input arguments:

- \( \text{Required} \) takes a Boolean value (0 or 1) that, when set to 1, specifies that the \( XSeries \) variable is required to be in the model. This might cause the model estimation to fail if the \( XSeries \) array variable is deemed inadmissible for inclusion into the underlying ARIMA model. The default value is 0.
- \( \text{NoDiff} \) takes a Boolean value (0 or 1) that, when set to 1, specifies that the \( XSeries \) variable does not automatically follow the \( XSeries \) variable differencing. The default value is 0.
- \( \text{ModelSymbol} \) takes a character variable that specifies the name of an input symbol in the model specification to be associated with the \( XSeries \) variable when the model is run. By default, the TSM.AddX method adds \( XSeries \) variables into an ARIMA model specification if they are not already referenced as an input symbol. Specification of \( \text{ModelSymbol} \) defines a way to include the \( XSeries \) variable if the ARIMASPEC object includes a symbol that matches the \( \text{ModelSymbol} \) argument.

TSM.criterion Method

\[ \text{criterion} = \text{obj}.\text{criterion}(); \]

Returns the fit statistic value for the final forecast for the TSM object. The criterion is set via the 'CRITERION' argument in the TSM.SetOption method. A missing value indicates that the FORENG instance has not produced a successful forecast.

**Arguments**
There are no arguments associated with this method.
TSM.GetForecast Method

```
rc = obj.GetForecast (Which, Result);
```

Gets the specified forecast series \( \text{(Which)} \) from the TSM object and stores it in the specified numeric array \( \text{(Result)} \).

**Input Arguments**
You must specify the following input argument:

- \( \text{Which} \) is a case-insensitive character string that specifies the type of forecast series to return. You can specify one of the following values:
  - \( \text{ERROR} \): returns prediction errors.
  - \( \text{LOWER} \): returns a lower confidence limit series.
  - \( \text{STDERR} \): returns a prediction standard error series.
  - \( \text{PREDICT} \): returns a prediction series.
  - \( \text{UPPER} \): returns an upper confidence limit series.

**Output Arguments**
You must specify the following output argument:

- \( \text{Result} \) specifies a numeric array to receive the forecast series.

TSM.Initialize Method

```
rc = obj.Initialize (< ModelSpec >);
```

Initializes a TSM object to use the specified \( \text{ModelSpec} \). This method must be called before the time series arrays \( \text{(XSeries and YSeries)} \) and other attributes for the TSM object are specified. If no \( \text{ModelSpec} \) object is specified, the TSM object is initialized to use the default ESM specification. This is equivalent to initializing the TSM object with an ESMSPEC object that has default option values.

**Input Arguments**
You can specify the following input argument:

- \( \text{ModelSpec} \) 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**

```r
cr = 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**

```r
cr = 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**

```r
cr = obj.SetOption ('Name', Value < , 'Name', Value ,...>);
```

Specifies the named options for the TSM object.

**Input Arguments**

You must specify at least one of the following 'Names' and its associated Value:

- `'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.

- `'BACK'` takes a nonnegative integer `Value` that specifies the back region for model performance. If `'BACK'=n` and the number of observations is `T`, then the first `T-n` observations are used to diagnose a series. The default is value 0.

- `'CRITERION'` takes a string `Value` 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 *SAS Forecast Server Procedures: User’s Guide*. The default is RMSE.
'HOLDOUT' takes a nonnegative integer Value 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 Value 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 Value that specifies the forecast lead. The default value is 12.

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.

**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 17.5 summarizes the methods that are associated with the ESMSPEC object.
Table 17.5 Methods of the ESMSPEC Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>Close ESM model specification</td>
</tr>
<tr>
<td>Open</td>
<td>Open ESM model specification</td>
</tr>
<tr>
<td>SetOption</td>
<td>Set option for ESM model</td>
</tr>
<tr>
<td>SetParm</td>
<td>Set parameters for ESM model</td>
</tr>
<tr>
<td>SetTransform</td>
<td>Set transform for ESM model</td>
</tr>
</tbody>
</table>

You can also store the XML representation of the ESMSPEC object in a CAS table. For more information, see the TSMSPEC object.

Figure 17.2 illustrates the data flow through the ESMSPEC object.

**Figure 17.2 ESMSPEC Object Data Flow**

ESMSPEC Synopsis

```plaintext
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

crc=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

crc=obj.Open () ;
Opens the ESMSPEC object.

Arguments
There are no arguments associated with this method.

ESMSPEC.SetOption Method

crc=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:

'CRIERION' takes a string Value that specifies the model selection criterion (statistic of fit) to be used
select from several candidate models. For a list of valid values, see the CRITERION=
option in the HPDIAGNOSE procedure in SAS Forecast Server Procedures: User’s
Guide. The default is RMSE.

'METHOD' takes a string Value 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 WIN-
TERS methods.
BESTN requests the best candidate nonseasonal smoothing model among
the SIMPLE, LINEAR, or DAMPTREND methods.
Chapter 17: Time Series Model Package

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

```c
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 Parm is the initial value of a damping weight parameter.
- **LEVEL** specifies that Parm is the initial value of a level weight parameter.
- **SEASON** specifies that Parm is the initial value of a season weight parameter.
- **TREND** specifies that Parm 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:

'Type' takes one of the following string values:

**AUTO** automatically chooses between NONE and LOG based on model selection criteria.
**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.

You can also specify the following input arguments:

'Option' 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.

Parm 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 17.6 summarizes the methods that are associated with the TSM.ARIMASPEC method.
### Table 17.6 Methods of the ARIMASPEC Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddARPoly</td>
<td>Add an autoregressive polynomial factor to the ARIMA model</td>
</tr>
<tr>
<td>AddMAPoly</td>
<td>Add a moving average polynomial factor to the ARIMA model</td>
</tr>
<tr>
<td>AddTF</td>
<td>Add a transfer function to the ARIMA model</td>
</tr>
<tr>
<td>AddTFDenPoly</td>
<td>Add a transfer function denominator factors to the ARIMA model</td>
</tr>
<tr>
<td>AddTFNumPoly</td>
<td>Add a transfer function numerator factors to the ARIMA model</td>
</tr>
<tr>
<td>Close</td>
<td>Close the ARIMA model specification</td>
</tr>
<tr>
<td>Open</td>
<td>Open the ARIMA model specification</td>
</tr>
<tr>
<td>SetDiff</td>
<td>Add differencing to the ARIMA model</td>
</tr>
<tr>
<td>SetOption</td>
<td>Specify the ARIMA option</td>
</tr>
<tr>
<td>SetTFTransform</td>
<td>Specify the transfer function transform</td>
</tr>
<tr>
<td>SetTransform</td>
<td>Specify transform</td>
</tr>
</tbody>
</table>

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 TSMSPEC object. Figure 17.3 illustrates the data flow through the ARIMASPEC object.
Figure 17.3 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**

```plaintext
DECLARE OBJECT obj (ARIMASPEC) ;
```

Method syntax, in order of typical usage:

```plaintext
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, DiffArray, 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**

```plaintext
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.

**DiffArray** 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 **DiffArray** 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 **DiffArray** values to use. If not specified, then all the values in **DiffArray** 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

\[
rc=\text{obj}.\text{AddTFNumPoly} ( \text{XName}, \text{NumArray} <,\text{NNum},\text{Seasonal}, \text{CoeffArray}>) ;
\]

Adds a transfer function numerator polynomial factor for the specified \textit{XName} variable. More polynomials can be added by subsequent calls to the ARIMASPEC.AddTFNumPoly method.

\textit{Input Arguments}
You must specify the following input argument:

\textit{XName} is a character string that specifies the name of the X variable.

\textit{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:

\textit{NNum} takes a numeric value that specifies the number of \textit{NumArray} values to use. By default, all \textit{NumArray} values are used.

\textit{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.

\textit{CoeffArray} is a numeric array that specifies numerator polynomial coefficients.

ARIMASPEC.Close Method

\[
rc=\text{obj}.\text{Close} () ;
\]

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.

\textit{Arguments}
There are no arguments associated with this method.

ARIMASPEC.Open Method

\[
rc=\text{obj}.\text{Open} () ;
\]

Initializes an empty ARIMASPEC object for configuration.

\textit{Arguments}
There are no arguments associated with this method.

ARIMASPEC.SetDiff Method

\[
rc=\text{obj}.\text{SetDiff} ( \text{DiffArray} <,\text{NDiff}>) ;
\]

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

```matlab
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 **Value** 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 **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. The default value is 0.001.
- **'DELTA'** takes a numeric **Value** 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 **Value** that specifies the maximum number of iterations allowed. The default is 50.
- **'METHOD'** takes a string **Value** that specifies the estimation method to use. You can specify one of the following **Values**:
  - **CLS** specifies the conditional least squares method.
  - **ML** specifies the maximum likelihood method.
  - **ULS** specifies the unconditional least squares method.
  The default is CLS.
- **'MU'** takes a numeric **Value** that specifies a constant term for the ARIMA model. The default is 0.
'NOEST' takes a Boolean Value (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 Value (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 Value (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 Value between 0 and 1, exclusive, that specifies the criterion for checking singularity. 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.

**ARIMASPEC.SetTFTransform Method**

```c
rc = obj.SetTFTransform (XName, 'Type', <Parm>);
```

Specifies a functional transform for specified XName variable.

**Input Arguments**

You must specify the following input arguments:

- **XName** 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(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** specifies square-root transformation.

  The default is NONE.

You can also specify the following input argument:

- **Parm** 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**

```c
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(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` specifies square-root transformation.  

  The default is `NONE`.

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.

---

**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 17.7* summarizes the methods that are associated with the EXMSPEC object.
### Table 17.7  Methods of the EXMSPEC Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Open EXM model specification</td>
</tr>
<tr>
<td>Close</td>
<td>Close EXM model specification</td>
</tr>
<tr>
<td>SetTransform</td>
<td>Set transform for EXM model specification</td>
</tr>
<tr>
<td>SetOption</td>
<td>Set options for EXM model specification</td>
</tr>
</tbody>
</table>

Figure 17.4 illustrates the data flow through the EXMSPEC object.

#### Figure 17.4  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 ESMSPEC object, see the section “TSM
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 922.

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 887.

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 Analysis and Forecasting Package” (SAS Visual Forecasting: Time Series Packages).

---

**EXMSPEC Synopsis**

```ruby
DECLARE OBJECT obj (EXMSPEC) ;
```

Method syntax, in order of typical usage:

```ruby
rc=obj.Open () ;
rc=obj.SetTransform ("Type" ,"Option","Parm">) ;
rc=obj.SetOption ("Name", Value < ,"Name",Value,...>) ;
rc=obj.Close () ;
```

---

**EXMSPEC Methods**

**EXMSPEC.Close Method**

```ruby
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**

```ruby
rc=obj.Open () ;
```

Initializes an EXMSPEC object for configuration.
EXMSPEC Methods

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 Value that specifies the variable name for ERROR series.
- **'LOWER'** takes a string Value that specifies the variable name for LOWER series.
- **'METHOD'** takes a string Value that specifies the method to approximate prediction STDERR series.
- **'NLAGPCT'** takes a numeric Value that specifies the percentage of error series count for ACF computations.
- **'NPARMS'** takes a numeric Value that specifies the number of parameters for external forecast.
- **'PREDICT'** takes a string Value that specifies the variable name for PREDICT series.
- **'SIGMA'** takes a string Value that specifies the prediction error standard deviation.
- **'STDERR'** takes a string Value that specifies the variable name for STDERR series.
- **'UPPER'** takes a string Value 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:
  - **AUTO** automatically chooses between NONE and LOG based on model selection criteria.
  - **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.
You can also specify the following input arguments:

'Option' 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**.

**Parm** 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 889 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 922.

Table 17.8 the methods that are associated with the IDMSPEC object.

**Table 17.8** Methods of the IDMSPEC Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>Close the IDM model specification</td>
</tr>
<tr>
<td>Open</td>
<td>Open the IDM model specification</td>
</tr>
<tr>
<td>SetMethod</td>
<td>Set component smoothing method</td>
</tr>
<tr>
<td>SetOption</td>
<td>Set options for IDM model specification</td>
</tr>
<tr>
<td>SetParm</td>
<td>Set component parameters</td>
</tr>
<tr>
<td>SetTransform</td>
<td>Set component transform</td>
</tr>
</tbody>
</table>

Figure 17.5 illustrates the data flow through the IDMSPEC object.
**Figure 17.5 IDMSPEC Object Data Flow**

![Diagram showing the data flow of IDMSPEC object]

---

**IDMSPEC Synopsis**

```plaintext
DECLARE OBJECT obj (IDMSPEC) ;

Method syntax, in order of typical usage:

```rc=objc.Open () ;
rc=objc.SetOption ('Name', Value < , 'Name', Value,...>) ;
rc=objc.SetTransform ('IDMComp', 'Type', 'Option', Parm>) ;
rc=objc.SetMethod ('IDMComp', 'Method') ;
rc=objc.SetParm ('IDMComp', 'ESMComp', Parm < , Noest, LRest, URest>) ;
rc=objc.Close () ;
```

**IDMSPEC Methods**

**IDMSPEC.Close Method**

```plaintext
rc=objc.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**

```plaintext
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 disabled (see the value NONE for the 'Type' argument in the SetTransform method).

**Arguments**
There are no arguments associated with this method.

**IDMSPEC.SetMethod Method**

```plaintext
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**

```plaintext
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 Value 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 Value 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 *SAS Forecast Server Procedures: User’s Guide*. The default is RMSE.

- **'MODEL'** is a character string that specifies a mnemonic for the IDM model to use. You can specify the following Values:

  - **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**

```plaintext
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:
  - **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 Parm is the initial value of a damping weight parameter.
  - **LEVEL** specifies that Parm is the initial value of a level weight parameter.
  - **SEASON** specifies that Parm is the initial value of a season weight parameter.
  - **TREND** specifies that Parm 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:
Noest takes a Boolean value (0 or 1) that, when set to 1, specifies that the Parm argument is fixed. The default value is 0 (initial).

LRest takes a numeric value between –1 and 2 that specifies a lower bound restriction on the ESM weight.

URest 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 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.

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 889. You can also store the XML representation of the UCM in a CAS table. For more information, see the section “TSMSPEC Object” on page 922. Note that the UCMSPEC object is equivalent to the HPFUCMSPEC procedure in SAS Forecast Server Procedures: User’s Guide.

Table 17.9 lists the methods that are associated with the UCMSPEC object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddAutoreg</td>
<td>Add an autoregressive component to a UCM model</td>
</tr>
<tr>
<td>AddBlockSeason</td>
<td>Add a block-season component to a UCM model</td>
</tr>
<tr>
<td>AddComponent</td>
<td>Add a fundamental component to a UCM model</td>
</tr>
<tr>
<td>AddCycle</td>
<td>Add a cycle component to a UCM model</td>
</tr>
<tr>
<td>AddDeplag</td>
<td>Add a deplag component to a UCM model</td>
</tr>
<tr>
<td>AddInput</td>
<td>Add an input component to a UCM model</td>
</tr>
<tr>
<td>AddSeason</td>
<td>Add a seasonal component to a UCM model</td>
</tr>
<tr>
<td>Close</td>
<td>Close the UCM model specification</td>
</tr>
<tr>
<td>Open</td>
<td>Open the UCM model specification</td>
</tr>
<tr>
<td>SetTransform</td>
<td>Specify the transform</td>
</tr>
</tbody>
</table>

Figure 17.6 illustrates the data flow through the UCMSPEC object.
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, CoefArray, 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

\[ rc = \text{obj.AddAutoreg}(<Rho, NoestRho, Variance, Noest>) \];

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:

- **Rho** takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of Rho must be in the interval \((-1, 1]\). The default is a missing value.
- **NoestRho** 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 NoestRho is 1, a valid nonmissing Rho value must be specified. The default value is 0.
- **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 variance of the AR(1) noise process 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.AddBlockSeason Method

\[ rc = \text{obj.AddBlockSeason}(<\text{BlockSize, NBlocks, 'Type', Offset, Variance, Noest}) \];

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:

- **BlockSize** takes a numeric value that specifies the block size, where BlockSize 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).
- **NBlocks** takes a numeric value that specifies the number of blocks, where the NBlocks can be any integer greater than or equal to 2.

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:
Chapter 17: Time Series Model Package

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**

```matlab
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:

- `Period`: specifies a numeric value as a starting value for the cycle period during the parameter estimation process, where `Period` can be any number larger than or equal to 2. The default is a missing value.
- `NoestPeriod`: takes a Boolean value that specifies whether the cycle period is to be estimated (0) or fixed (1) at the specified starting value. If `NoestPeriod` is 1, a valid nonmissing `Period` value must be specified. The default value is 0.
- `Rho`: takes a numeric value that specifies the starting value for the AR(1) coefficient during the parameter estimation process. The value of `Rho` must be in the interval \((-1, 1]\). The default is a missing value.
- `NoestRho`: 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 `NoestRho` is 1, a valid nonmissing `Rho` value must be specified. The default value is 0.
- `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.AddDeplag Method**

```matlab
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:

- `LagArray`: 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:

- `NLag`: takes an integer that specifies the number of `LagArray` values to use. The default is all `LagArray` values.
- `CoeffArray`: 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**

```csharp
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. If not specified, no transform is used.

**TransParm** takes a numeric value that specifies a transform parameter. This argument is used only for Box-Cox transformations.

**UCMSPEC.AddSeason Method**

```csharp
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:
**UCMSPEC Methods**

**UCMSPEC.Close Method**

```plaintext
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**

```plaintext
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**

```plaintext
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:

- **AUTO** automatically chooses between NONE and LOG based on model selection criteria.
- **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.

'value' 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 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 17.10 summarizes the methods that are associated with the TSMPEST object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Collect parameter estimates from TSM objects</td>
</tr>
<tr>
<td>nrows</td>
<td>Return the number of rows that are collected</td>
</tr>
</tbody>
</table>

Figure 17.7 illustrates the data flow through the TSMPEST object.
**TSMPEST Synopsis**

```verbatim
DECLARE OBJECT obj (TSMPEST) ;
```

Method syntax, in order of typical usage:

```verbatim
rc = obj.Collect (TSMObj) ;
nrows = obj.nrows () ;
```

**TSMPEST Methods**

**TSMPEST.Collect Method**

```verbatim
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**

\[ \text{nrows}=\text{obj.nrows}() \]

Returns the number of rows that have been collected and stored in the CAS table.

**Arguments**

There are no arguments associated with this method.

---

**TSMSPEC Object**

The time series model specification (TSMSPEC) collector object collects the model specification from a TSM object, \( \text{TSMObj} \), and stores it in a CAS table for printing or archiving or for use by a repeater object on another TSM object.

Table 17.11 summarizes the methods that are associated with the TSMSPEC object.

**Table 17.11 Methods of the TSMSPEC Object**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Collect model specification from TSM objects</td>
</tr>
<tr>
<td>nrows</td>
<td>Return the number of rows that are collected</td>
</tr>
</tbody>
</table>

Figure 17.8 illustrates the data flow through the TSMSPEC object.
**TSMSPEC Synopsis**

```plaintext
DECLARE OBJECT obj (TSMSPEC) ;
```

Method syntax, in order of typical usage:

```plaintext
rc = obj.Collect (TSMObj) ;
nrows = obj.nrows () ;
```

---

**TSMSPEC Methods**

**TSMSPEC.Collect Method**

```plaintext
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**

```plaintext
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 17.12 shows the contents of the TSMFOR object.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>String</td>
<td>Name of the dependent variable</td>
</tr>
<tr>
<td><em>MODEL</em></td>
<td>String</td>
<td>(Optional) Model name</td>
</tr>
<tr>
<td><em>TIMEID</em></td>
<td>Numeric</td>
<td>Uniform time ID values for series</td>
</tr>
<tr>
<td>ACTUAL</td>
<td>Numeric</td>
<td>Accumulated values of dependent variable</td>
</tr>
<tr>
<td>ERROR</td>
<td>Numeric</td>
<td>Residuals</td>
</tr>
<tr>
<td>LOWER</td>
<td>Numeric</td>
<td>Lower confidence limit</td>
</tr>
<tr>
<td>PREDICT</td>
<td>Numeric</td>
<td>Forecasts of the dependent variable</td>
</tr>
<tr>
<td>STD</td>
<td>Numeric</td>
<td>Prediction standard error</td>
</tr>
<tr>
<td>UPPER</td>
<td>Numeric</td>
<td>Upper confidence limit</td>
</tr>
</tbody>
</table>

Table 17.13 summarizes the methods that are associated with the TSMFOR object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Collect forecasts estimates from the TSM object</td>
</tr>
<tr>
<td>nrows</td>
<td>Return the number of rows collected</td>
</tr>
<tr>
<td>SetOption</td>
<td>Specifies options for the TSMFOR instance</td>
</tr>
</tbody>
</table>

Figure 17.9 illustrates the data flow through the TSMFOR object.
**Figure 17.9** TSMFOR Object Data Flow

![Diagram of TSMFOR Object Data Flow]

**TSMFOR Synopsis**

DECLARE OBJECT obj (TSMFOR <('Name', Value)>) ;

Method syntax, in order of typical usage:

\[
rc = \text{obj}.\text{SetOption} (\text{Name}, \text{Value}) ;
\]

\[
rc = \text{obj}.\text{Collect} (\text{ModelObj < Region} >) ;
\]

\[
rc = \text{obj}.\text{nrows} () ;
\]

**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**

```r
crc = 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 17.14 shows the contents of the TSMSTAT object.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>String</td>
<td>Name of the dependent variable</td>
</tr>
<tr>
<td><em>REGION</em></td>
<td>String</td>
<td>Region in which the fit statistics are calculated. You can specify the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>FIT</strong> indicates that fit statistics were calculated over the fit region.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>FORECAST</strong> indicates that fit statistics were calculated over the forecast (back) region.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>HOLDOUT</strong> indicates that fit statistics were calculated over the holdout region.</td>
</tr>
<tr>
<td><em>MODEL</em></td>
<td>String</td>
<td>Model name</td>
</tr>
<tr>
<td>AADJRSQ</td>
<td>Numeric</td>
<td>Amemiya’s adjusted R-square</td>
</tr>
<tr>
<td>ADJRSQ</td>
<td>Numeric</td>
<td>Adjusted R-square</td>
</tr>
<tr>
<td>AIC</td>
<td>Numeric</td>
<td>Akaike’s information criterion</td>
</tr>
<tr>
<td>AICC</td>
<td>Numeric</td>
<td>Finite sample corrected AIC</td>
</tr>
<tr>
<td>APC</td>
<td>Numeric</td>
<td>Amemiya’s prediction criterion</td>
</tr>
<tr>
<td>DFE</td>
<td>Numeric</td>
<td>Degrees of freedom error</td>
</tr>
<tr>
<td>GMAPE</td>
<td>Numeric</td>
<td>Geometric mean absolute percentage error</td>
</tr>
<tr>
<td>GMAPES</td>
<td>Numeric</td>
<td>Geometric mean absolute error as a percentage of standard deviation</td>
</tr>
<tr>
<td>GMAPPE</td>
<td>Numeric</td>
<td>Geometric mean absolute predictive percentage error</td>
</tr>
<tr>
<td>GMRAE</td>
<td>Numeric</td>
<td>Geometric mean relative absolute error</td>
</tr>
<tr>
<td>GMASPE</td>
<td>Numeric</td>
<td>Geometric mean absolute symmetric percentage error</td>
</tr>
<tr>
<td>MAE</td>
<td>Numeric</td>
<td>Mean absolute error</td>
</tr>
<tr>
<td>MAPE</td>
<td>Numeric</td>
<td>Mean absolute percentage error</td>
</tr>
</tbody>
</table>
Table 17.14  continued

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPES</td>
<td>Numeric</td>
<td>Mean absolute error as a percentage of standard deviation</td>
</tr>
<tr>
<td>MAPPE</td>
<td>Numeric</td>
<td>Mean absolute predictive percentage error</td>
</tr>
<tr>
<td>MASE</td>
<td>Numeric</td>
<td>Mean absolute scaled error</td>
</tr>
<tr>
<td>MAXAPES</td>
<td>Numeric</td>
<td>Maximum absolute error as a percentage of standard deviation</td>
</tr>
<tr>
<td>MAXERR</td>
<td>Numeric</td>
<td>Maximum error</td>
</tr>
<tr>
<td>MAXPE</td>
<td>Numeric</td>
<td>Maximum percentage error</td>
</tr>
<tr>
<td>MAXPPE</td>
<td>Numeric</td>
<td>Maximum predictive percentage error</td>
</tr>
<tr>
<td>MAXRE</td>
<td>Numeric</td>
<td>Maximum relative error</td>
</tr>
<tr>
<td>MAXSPE</td>
<td>Numeric</td>
<td>Maximum symmetric percentage error</td>
</tr>
<tr>
<td>MDAPE</td>
<td>Numeric</td>
<td>Median absolute percentage error</td>
</tr>
<tr>
<td>MDAPES</td>
<td>Numeric</td>
<td>Median absolute error as a percentage of standard deviation</td>
</tr>
<tr>
<td>MDAPPE</td>
<td>Numeric</td>
<td>Median absolute predictive percentage error</td>
</tr>
<tr>
<td>MDASPE</td>
<td>Numeric</td>
<td>Median absolute symmetric percentage error</td>
</tr>
<tr>
<td>MDRAE</td>
<td>Numeric</td>
<td>Median relative absolute error</td>
</tr>
<tr>
<td>ME</td>
<td>Numeric</td>
<td>Mean error</td>
</tr>
<tr>
<td>MINAPES</td>
<td>Numeric</td>
<td>Minimum absolute error as a percentage of standard deviation</td>
</tr>
<tr>
<td>MINERR</td>
<td>Numeric</td>
<td>Minimum error</td>
</tr>
<tr>
<td>MINPE</td>
<td>Numeric</td>
<td>Minimum percentage error</td>
</tr>
<tr>
<td>MINPPE</td>
<td>Numeric</td>
<td>Minimum predictive percentage error</td>
</tr>
<tr>
<td>MINRE</td>
<td>Numeric</td>
<td>Minimum relative error</td>
</tr>
<tr>
<td>MINSPE</td>
<td>Numeric</td>
<td>Minimum symmetric percentage error</td>
</tr>
<tr>
<td>MPE</td>
<td>Numeric</td>
<td>Mean percentage error</td>
</tr>
<tr>
<td>MPPE</td>
<td>Numeric</td>
<td>Mean predictive percentage error</td>
</tr>
<tr>
<td>MRAE</td>
<td>Numeric</td>
<td>Mean relative absolute error</td>
</tr>
<tr>
<td>MRE</td>
<td>Numeric</td>
<td>Mean relative error</td>
</tr>
<tr>
<td>MSE</td>
<td>Numeric</td>
<td>Mean square error</td>
</tr>
<tr>
<td>MSPE</td>
<td>Numeric</td>
<td>Mean symmetric percentage error</td>
</tr>
<tr>
<td>N</td>
<td>Numeric</td>
<td>Number of observations that were used</td>
</tr>
<tr>
<td>NMISSA</td>
<td>Numeric</td>
<td>Number of missing actual values</td>
</tr>
</tbody>
</table>
Table 17.14  continued

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMISSP</td>
<td>Numeric</td>
<td>Number of missing predicted values</td>
</tr>
<tr>
<td>NOBS</td>
<td>Numeric</td>
<td>Number of observations</td>
</tr>
<tr>
<td>NPARMS</td>
<td>Numeric</td>
<td>Number of parameters</td>
</tr>
<tr>
<td>RMSE</td>
<td>Numeric</td>
<td>Root mean square error</td>
</tr>
<tr>
<td>RSQUARE</td>
<td>Numeric</td>
<td>R-square</td>
</tr>
<tr>
<td>RWRSQ</td>
<td>Numeric</td>
<td>Random walk R-square</td>
</tr>
<tr>
<td>SBC</td>
<td>Numeric</td>
<td>Schwarz Bayesian information criterion</td>
</tr>
<tr>
<td>SMAPE</td>
<td>Numeric</td>
<td>Symmetric mean absolute percentage error</td>
</tr>
<tr>
<td>SSE</td>
<td>Numeric</td>
<td>Sum of square error</td>
</tr>
<tr>
<td>SST</td>
<td>Numeric</td>
<td>Corrected total sum of squares</td>
</tr>
<tr>
<td>TSS</td>
<td>Numeric</td>
<td>Total sum of squares</td>
</tr>
<tr>
<td>UMSE</td>
<td>Numeric</td>
<td>Unbiased mean square error</td>
</tr>
<tr>
<td>URMSE</td>
<td>Numeric</td>
<td>Unbiased root mean square error</td>
</tr>
</tbody>
</table>

Table 17.15 summarizes the methods that are associated with the TSMSTAT object.

Table 17.15  Methods of the TSMSTAT Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Collect model statistics of fit from the TSM object</td>
</tr>
<tr>
<td>nrows</td>
<td>Return the number of rows collected</td>
</tr>
</tbody>
</table>

Figure 17.10 illustrates the data flow through the TSMSTAT object.
TSMSTAT Synopsis

DECLARE OBJECT obj (TSMSTAT) ;

Method syntax, in order of typical usage:

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

\[ \text{nrows} = \text{obj.nrows}(); \]

Returns the number of rows that have been collected and stored in the CAS table.

**Arguments**
There are no arguments associated with this method.

---

**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 17.3).

Table 17.16 summarizes the methods of the TSMINEST object.

**Table 17.16** Methods of the TSMINEST Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrows</td>
<td>Return number of rows in the TSMINEST object</td>
</tr>
</tbody>
</table>

**TSMINEST Synopsis**

```c
DECLARE OBJECT obj (TSMINEST);
```

Method syntax:

\[ rc = \text{obj.nrows}(); \]

**TSMINEST Methods**

**TSMINEST.nrows Method**

\[ rc = \text{obj.nrows}(); \]

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 17.17 summarizes the methods of the TSMINSPEC object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrows</td>
<td>Return number of rows in TSMINSPEC object</td>
</tr>
</tbody>
</table>

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 9, “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:

```sas
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:

```sas
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:

```sas
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: Language Reference*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Example 17.1: 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) × (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 tsmode 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;
ma[1] = 12;
rc = airSpec.AddMAPoly(ma);
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 17.1.1 shows the predictions, and Output 17.1.2 shows the parameter estimates for the airline model.

**Output 17.1.1** Airline Model Predictions (Partial Output)

<table>
<thead>
<tr>
<th>DATE</th>
<th>PREDICT</th>
<th>STD</th>
<th>UPPER</th>
<th>LOWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN1961</td>
<td>450.4</td>
<td>16.9215</td>
<td>484.5</td>
<td>418.2</td>
</tr>
<tr>
<td>FEB1961</td>
<td>426.1</td>
<td>18.8590</td>
<td>464.2</td>
<td>390.3</td>
</tr>
<tr>
<td>MAR1961</td>
<td>480.1</td>
<td>24.0408</td>
<td>528.9</td>
<td>434.7</td>
</tr>
<tr>
<td>APR1961</td>
<td>492.8</td>
<td>27.2405</td>
<td>548.3</td>
<td>441.6</td>
</tr>
<tr>
<td>MAY1961</td>
<td>509.5</td>
<td>30.5863</td>
<td>572.0</td>
<td>452.1</td>
</tr>
<tr>
<td>JUN1961</td>
<td>584.2</td>
<td>37.6514</td>
<td>661.4</td>
<td>513.9</td>
</tr>
<tr>
<td>JUL1961</td>
<td>670.7</td>
<td>45.9957</td>
<td>765.3</td>
<td>585.1</td>
</tr>
<tr>
<td>AUG1961</td>
<td>668.2</td>
<td>48.4237</td>
<td>768.0</td>
<td>578.3</td>
</tr>
<tr>
<td>SEP1961</td>
<td>559.6</td>
<td>42.6271</td>
<td>647.7</td>
<td>480.7</td>
</tr>
<tr>
<td>OCT1961</td>
<td>498.3</td>
<td>39.7181</td>
<td>580.6</td>
<td>425.0</td>
</tr>
<tr>
<td>NOV1961</td>
<td>431.2</td>
<td>35.8240</td>
<td>505.5</td>
<td>365.2</td>
</tr>
<tr>
<td>DEC1961</td>
<td>478.9</td>
<td>41.3484</td>
<td>565.0</td>
<td>403.0</td>
</tr>
</tbody>
</table>

**Output 17.1.2** Airline Model Parameter Estimates (Partial Output)

<table>
<thead>
<tr>
<th>EST</th>
<th>STDERR</th>
<th>TVALE</th>
<th>PVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3773</td>
<td>0.0820</td>
<td>4.6033</td>
<td>9.828E-6</td>
</tr>
<tr>
<td>0.5724</td>
<td>0.0780</td>
<td>7.3361</td>
<td>2.17E-11</td>
</tr>
</tbody>
</table>

Output 17.1.3 shows the number of times the airline model residuals are smaller than the Winters model residuals.
Chapter 17: Time Series Model Package

Example 17.2: 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 be $y = TFinput(x) + AR(2)$, where 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:

```plaintext
proc tsm model 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 ***;
```

---

**Output 17.1.3** Number of Times the Airline Model Residuals Are Smaller Than the Winters Model Residuals

<table>
<thead>
<tr>
<th>nbetter</th>
</tr>
</thead>
<tbody>
<tr>
<td>71.0000</td>
</tr>
</tbody>
</table>
Example 17.3: 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 17.1 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.

```plaintext
proc tsmodel data=mycas.air
   outobj=(airEst=mycas.airEst airOSpec=mycas.airOSpec) ;
   id date interval=month;
   var air;
   require tsm;
```

Output 17.2.1 shows the parameter estimates for the transfer function model.

**Output 17.2.1 Parameter Estimates for the Transfer Function Model (Partial Output)**

<table>
<thead>
<tr>
<th>EST</th>
<th>STDERR</th>
<th>TVALUE</th>
<th>PVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>53.2630</td>
<td>0.1193</td>
<td>446.5</td>
<td>0</td>
</tr>
<tr>
<td>1.5329</td>
<td>0.0475</td>
<td>32.2472</td>
<td>6.25E-97</td>
</tr>
<tr>
<td>-0.6330</td>
<td>0.0501</td>
<td>-12.6434</td>
<td>2.27E-29</td>
</tr>
<tr>
<td>-0.5352</td>
<td>0.0748</td>
<td>-7.1534</td>
<td>7.21E-12</td>
</tr>
<tr>
<td>0.3760</td>
<td>0.1029</td>
<td>3.6553</td>
<td>0.000306</td>
</tr>
<tr>
<td>0.5189</td>
<td>0.1078</td>
<td>4.8124</td>
<td>2.425E-6</td>
</tr>
<tr>
<td>0.5484</td>
<td>0.0382</td>
<td>14.3499</td>
<td>1.72E-35</td>
</tr>
</tbody>
</table>

Example 17.3: Replaying a Previously Fitted 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 17.2.1 shows the parameter estimates for the transfer function model.
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 17.3.1 shows the parameter estimates that are saved in mycas.airEst.

**Output 17.3.1** Parameter Estimates for the Airline Model (Partial Output)

<table>
<thead>
<tr>
<th>EST</th>
<th>STDERR</th>
<th>TVALUE</th>
<th>PVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3773</td>
<td>0.0820</td>
<td>4.6033</td>
<td>9.828E-6</td>
</tr>
<tr>
<td>0.5724</td>
<td>0.0780</td>
<td>7.3361</td>
<td>2.17E-11</td>
</tr>
</tbody>
</table>
The following statements show how to forecast the airline series by using the previously saved model specification (mycas.airOSpec) and parameter estimates (mycas.airEst).

```plaintext
proc tsm model 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(tsminspec);
declare object airEst(tsmest);
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 17.3.2 shows the forecasts that are produced according to the fitted model.

**Output 17.3.2** Replayed Forecasts (Partial Output)

**Airline Model Predictions**

<table>
<thead>
<tr>
<th>DATE</th>
<th>PREDICT</th>
<th>STD</th>
<th>UPPER</th>
<th>LOWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN1961</td>
<td>450.4</td>
<td>16.9215</td>
<td>484.5</td>
<td>418.2</td>
</tr>
<tr>
<td>FEB1961</td>
<td>426.1</td>
<td>18.8590</td>
<td>464.2</td>
<td>390.3</td>
</tr>
<tr>
<td>MAR1961</td>
<td>480.1</td>
<td>24.0408</td>
<td>528.9</td>
<td>434.7</td>
</tr>
<tr>
<td>APR1961</td>
<td>492.8</td>
<td>27.2405</td>
<td>548.3</td>
<td>441.6</td>
</tr>
<tr>
<td>MAY1961</td>
<td>509.5</td>
<td>30.5863</td>
<td>572.0</td>
<td>452.1</td>
</tr>
<tr>
<td>JUN1961</td>
<td>584.2</td>
<td>37.6514</td>
<td>661.4</td>
<td>513.9</td>
</tr>
<tr>
<td>JUL1961</td>
<td>670.7</td>
<td>45.9957</td>
<td>765.3</td>
<td>585.1</td>
</tr>
<tr>
<td>AUG1961</td>
<td>668.2</td>
<td>48.4237</td>
<td>768.0</td>
<td>578.3</td>
</tr>
<tr>
<td>SEP1961</td>
<td>559.6</td>
<td>42.6271</td>
<td>647.7</td>
<td>480.7</td>
</tr>
<tr>
<td>OCT1961</td>
<td>498.3</td>
<td>39.7181</td>
<td>580.6</td>
<td>425.0</td>
</tr>
<tr>
<td>NOV1961</td>
<td>431.2</td>
<td>35.8240</td>
<td>505.5</td>
<td>365.2</td>
</tr>
<tr>
<td>DEC1961</td>
<td>478.9</td>
<td>41.3484</td>
<td>565.0</td>
<td>403.0</td>
</tr>
</tbody>
</table>
Example 17.4: 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 17.1 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.

```sas
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 17.4.1.

```sas
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 17.4.1, 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.
Example 17.4: Performing Time Series Imputation Using an ARIMA Model

**Output 17.4.1** 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:

```plaintext
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) ***;
```

```plaintext
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 17.4.2.

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 17.4.2, 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 17.4.2 Imputed Airline Passenger Time Series

![Graph showing imputed airline passenger time series]

References

Chapter 18
Utility Package

Contents

Overview: UTL Package .................................................. 947
UTL Package Summary .................................................. 948
Using the UTL Package .................................................. 948
  Common Argument Types .............................................. 949
  Return Codes .......................................................... 949
CLIMITS Object ............................................................ 950
  CLIMITS Synopsis .................................................... 950
  CLIMITS Methods ..................................................... 951
OUTNVP Object ............................................................ 952
  OUTNVP Synopsis .................................................... 953
  OUTNVP Methods ..................................................... 954
UTLSTAT Object ............................................................ 955
  UTLSTAT Synopsis ................................................... 957
  UTLSTAT Methods .................................................... 958
Examples: UTL Package .................................................. 959
  Using CAS Sessions and CAS Engine Librefs ..................... 959
  Example 18.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into CAS Tables .................................................. 960
  Example 18.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc External Forecast .................................................. 963

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 18.1 summarizes the single object class in the SFS package.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIMITS</td>
<td>Compute prediction standard errors and confidence limits for specified actual and predicted time series.</td>
</tr>
<tr>
<td>OUTNVP</td>
<td>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.</td>
</tr>
<tr>
<td>UTLSTAT</td>
<td>Collector object for computing forecast fit statistics for specified actual and predicted time series and storing those statistics in a CAS table.</td>
</tr>
</tbody>
</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
The 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 18.2 defines the common argument types that are used in this chapter. The symbol \( x \) corresponds to the variable name.

**Table 18.2** Common Argument Types

<table>
<thead>
<tr>
<th>SAS Data Type</th>
<th>Declaration Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>LENGTH ( x $n );</td>
</tr>
<tr>
<td>Numeric</td>
<td>( x ) or LENGTH ( x &amp;8 );</td>
</tr>
<tr>
<td>Numeric array</td>
<td>ARRAY ( x[n]/)NOSYMBOLS;</td>
</tr>
<tr>
<td>Status</td>
<td>( x ) or LENGTH ( x &amp;8 );</td>
</tr>
</tbody>
</table>

### Return Codes

Table 18.3 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 18.3** Return Codes

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; 0)</td>
<td>An unrecoverable error occurred. No result was produced.</td>
</tr>
<tr>
<td>(= 0)</td>
<td>Unconditional success. The requested action completed and a normal result was produced.</td>
</tr>
<tr>
<td>(&gt; 0)</td>
<td>Conditional success or warning. A result was produced subject to conditions.</td>
</tr>
</tbody>
</table>

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 9, “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 18.4 summarizes the methods that are associated with the UTL object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>Compute the prediction standard errors and confidence limits for specified actual and predicted time series</td>
</tr>
<tr>
<td>GetForecast</td>
<td>Retrieve a computed forecast series by name</td>
</tr>
</tbody>
</table>

**CLIMITS Synopsis**

```plaintext
DECLARE OBJECT obj (CLIMITS) ;
```

Method syntax, in order of typical usage:

```plaintext
rc=obj.Compute (Actual,Predicted,<Alpha>) ;
rc=obj.GetForecast (Which,Result) ;
```

Figure 18.1 outlines the programmatic data flow through the CLIMITS object; each arrow represents a different object method.
**CLIMITS Methods**

**CLIMITS.Compute Method**

```plaintext
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:
Chapter 18: Utility Package

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 18.5 shows the contents of the OUTNVP object.
Table 18.5  Contents of the OUTNVP Object

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>String</td>
<td>Name of the dependent variable</td>
</tr>
<tr>
<td><em>CALL</em></td>
<td>Numeric</td>
<td>Call count within the BY group</td>
</tr>
<tr>
<td><em>UTAG</em></td>
<td>Numeric</td>
<td>User-defined numeric tag</td>
</tr>
<tr>
<td><em>VIX</em></td>
<td>Numeric</td>
<td>Value index (1-based) for the row</td>
</tr>
<tr>
<td><em>VALUE</em></td>
<td>Numeric</td>
<td>Actual value for variable’s row</td>
</tr>
</tbody>
</table>

Table 18.6 summarizes the methods that are associated with the OUTNVP object.

Table 18.6  Methods of the OUTNVP Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Collect value for numeric data type</td>
</tr>
<tr>
<td>nrows</td>
<td>Get the OUTNVP instance row count</td>
</tr>
</tbody>
</table>

OUTNVP Synopsis

DECLARE OBJECT obj (OUTNVP) ;

Method syntax, in order of typical usage:

rc=obj.Collect (Variable,< Utag>) ;  
nrows=obj.nrows () ;

Figure 18.2 outlines the programmatic data flow through the OUTNVP object.
OUTNVP Methods

OUTNVP.Collect Method

\[ rc=obj.Collect\left(\text{Variable},\text{Utag}\right); \]

Stores a numeric type, either scalar literal or variable or array variable in the OUTNVP table. When the \text{Variable} parameter is a numeric scalar literal or variable, this method collects a single row into the OUTNVP table. When the \text{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 \text{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.

\textit{Input Arguments}

You must specify the following input arguments:

\textit{Variable} specifies a numeric array to be collected into the OUTNVP table.

You can also specify the following input argument:

\textit{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

\[ \text{nrows} = \text{obj.nrows}(); \]

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 18.7 shows the contents of the UTLSTAT object. For more information about the HPFENGINE procedure, see *SAS Forecast Server Procedures: User’s Guide*.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NAME</em></td>
<td>String</td>
<td>Variable name of actual series</td>
</tr>
<tr>
<td><em>MODEL</em></td>
<td>String</td>
<td>Variable name of predicted series</td>
</tr>
<tr>
<td>DFE</td>
<td>Numeric</td>
<td>Degrees of freedom error</td>
</tr>
<tr>
<td>N</td>
<td>Numeric</td>
<td>Number of observations</td>
</tr>
<tr>
<td>NOBS</td>
<td>Numeric</td>
<td>Number of observations used</td>
</tr>
<tr>
<td>NMISSA</td>
<td>Numeric</td>
<td>Number of missing actuals</td>
</tr>
<tr>
<td>NMISSP</td>
<td>Numeric</td>
<td>Number of missing predicted values</td>
</tr>
<tr>
<td>NPARMS</td>
<td>Numeric</td>
<td>Number of model parameters</td>
</tr>
<tr>
<td>TSS</td>
<td>Numeric</td>
<td>Total sum of squares</td>
</tr>
<tr>
<td>SST</td>
<td>Numeric</td>
<td>Corrected total sum of squares</td>
</tr>
<tr>
<td>SSE</td>
<td>Numeric</td>
<td>Sum of square error</td>
</tr>
<tr>
<td>MSE</td>
<td>Numeric</td>
<td>Mean square error</td>
</tr>
<tr>
<td>RMSE</td>
<td>Numeric</td>
<td>Root mean square error</td>
</tr>
<tr>
<td>UMSE</td>
<td>Numeric</td>
<td>Unbiased mean square error</td>
</tr>
<tr>
<td>URMSE</td>
<td>Numeric</td>
<td>Unbiased root mean square error</td>
</tr>
</tbody>
</table>
Table 18.7  continued

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPE</td>
<td>Numeric</td>
<td>Mean absolute percentage of error</td>
</tr>
<tr>
<td>MAE</td>
<td>Numeric</td>
<td>Mean absolute error</td>
</tr>
<tr>
<td>RSQUARE</td>
<td>Numeric</td>
<td>R-square</td>
</tr>
<tr>
<td>ADJRSQ</td>
<td>Numeric</td>
<td>Adjusted R-square</td>
</tr>
<tr>
<td>AADJRSQ</td>
<td>Numeric</td>
<td>Amemiya’s adjusted R-square</td>
</tr>
<tr>
<td>RWRSQ</td>
<td>Numeric</td>
<td>Random walk R-square</td>
</tr>
<tr>
<td>AIC</td>
<td>Numeric</td>
<td>Akaike’s information criterion</td>
</tr>
<tr>
<td>AICC</td>
<td>Numeric</td>
<td>Finite sample corrected Akaike’s information criterion</td>
</tr>
<tr>
<td>SBC</td>
<td>Numeric</td>
<td>Schwarz Bayesian information criterion</td>
</tr>
<tr>
<td>APC</td>
<td>Numeric</td>
<td>Amemiya’s prediction criterion</td>
</tr>
<tr>
<td>MAXERR</td>
<td>Numeric</td>
<td>Maximum error</td>
</tr>
<tr>
<td>MINERR</td>
<td>Numeric</td>
<td>Minimum error</td>
</tr>
<tr>
<td>MAXPE</td>
<td>Numeric</td>
<td>Maximum percentage of error</td>
</tr>
<tr>
<td>MINPE</td>
<td>Numeric</td>
<td>Minimum percentage of error</td>
</tr>
<tr>
<td>ME</td>
<td>Numeric</td>
<td>Mean error</td>
</tr>
<tr>
<td>MPE</td>
<td>Numeric</td>
<td>Mean percentage of error</td>
</tr>
<tr>
<td>MDAPE</td>
<td>Numeric</td>
<td>Median absolute percentage of error</td>
</tr>
<tr>
<td>GMAPE</td>
<td>Numeric</td>
<td>Geometric mean absolute percentage of error</td>
</tr>
<tr>
<td>MINPPE</td>
<td>Numeric</td>
<td>Minimum predicted percentage of error</td>
</tr>
<tr>
<td>MAXPPE</td>
<td>Numeric</td>
<td>Maximum predicted percentage of error</td>
</tr>
<tr>
<td>MPPE</td>
<td>Numeric</td>
<td>Mean predicted percentage of error</td>
</tr>
<tr>
<td>MAPPE</td>
<td>Numeric</td>
<td>Mean absolute predicted percentage of error</td>
</tr>
<tr>
<td>MDAPPE</td>
<td>Numeric</td>
<td>Median absolute predicted percentage of error</td>
</tr>
<tr>
<td>GMAPPE</td>
<td>Numeric</td>
<td>Geometric mean absolute predicted percentage of error</td>
</tr>
<tr>
<td>MINSPE</td>
<td>Numeric</td>
<td>Minimum symmetric percentage of error</td>
</tr>
<tr>
<td>MAXSPE</td>
<td>Numeric</td>
<td>Maximum symmetric percentage of error</td>
</tr>
<tr>
<td>MSPE</td>
<td>Numeric</td>
<td>Mean symmetric percentage of error</td>
</tr>
<tr>
<td>SMAPE</td>
<td>Numeric</td>
<td>Mean absolute symmetric percentage of error</td>
</tr>
<tr>
<td>MDASPE</td>
<td>Numeric</td>
<td>Median absolute symmetric percentage of error</td>
</tr>
</tbody>
</table>
Table 18.7  continued

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMASPE</td>
<td>Numeric</td>
<td>Geometric mean absolute symmetric percentage of error</td>
</tr>
<tr>
<td>MINRE</td>
<td>Numeric</td>
<td>Minimum relative error</td>
</tr>
<tr>
<td>MAXRE</td>
<td>Numeric</td>
<td>Maximum relative error</td>
</tr>
<tr>
<td>MRE</td>
<td>Numeric</td>
<td>Mean relative error</td>
</tr>
<tr>
<td>MRAE</td>
<td>Numeric</td>
<td>Mean relative absolute error</td>
</tr>
<tr>
<td>MDRAE</td>
<td>Numeric</td>
<td>Median relative absolute error</td>
</tr>
<tr>
<td>GMRAE</td>
<td>Numeric</td>
<td>Geometric mean relative absolute error</td>
</tr>
<tr>
<td>MASE</td>
<td>Numeric</td>
<td>Mean absolute scaled error</td>
</tr>
<tr>
<td>MINAPES</td>
<td>Numeric</td>
<td>Minimum absolute error percentage of standard deviation</td>
</tr>
<tr>
<td>MAXAPES</td>
<td>Numeric</td>
<td>Maximum absolute error percentage of standard deviation</td>
</tr>
<tr>
<td>MAPES</td>
<td>Numeric</td>
<td>Mean absolute error percentage of standard deviation</td>
</tr>
<tr>
<td>MDAPES</td>
<td>Numeric</td>
<td>Median absolute error percentage of standard deviation</td>
</tr>
<tr>
<td>GMAPES</td>
<td>Numeric</td>
<td>Geometric mean absolute error percentage of standard deviation</td>
</tr>
</tbody>
</table>

Table 18.8 summarizes the methods that are associated with the UTLSTAT object.

Table 18.8   Methods of the UTLSTAT Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collect</td>
<td>Compute and collect forecast fit statistics for a specified ad hoc pair of actual and predicted time series.</td>
</tr>
<tr>
<td>nrows</td>
<td>Get the UTLSTAT instance row count</td>
</tr>
</tbody>
</table>

UTLSTAT Synopsis

DECLARE OBJECT obj (UTLSTAT) ;

Method syntax, in order of typical usage:

\[ rc = obj.Collect(Actual, Predicted, <Nparms>) ; \]
\[ nrows = obj.nrows() ; \]

Figure 18.3 outlines the programmatic data flow through the UTLSTAT object.
**UTLSTAT Methods**

**UTLSTAT.Collect Method**

`utilstatCollect 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 18.7. 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**

```plaintext
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 9, “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:

```plaintext
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:

```plaintext
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: Language Reference*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

### Example 18.1: 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 18.7) 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 tsmode data=mycas.air
          outarray = mycas.outarray (replace=yes)
          outscalar = mycas.outscalar (replace=yes)
          outobj=(
              utlstatobj = mycas.utlstat (replace = YES)
              outnvpmodelobj = mycas.outnvpmodel (replace = YES)
          );
```
Example 18.1: Collecting Forecast Fit Statistics and Ad Hoc Numeric Variables into CAS Tables

```plaintext
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;

/* 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 18.1.1 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 18.1.1** Sample of the Forecast Fit Statistics Computed and Collected by the UTLSTAT Object

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>NAME</em></th>
<th><em>MODEL</em></th>
<th>NOBS</th>
<th>RMSE</th>
<th>MAPE</th>
<th>MAE</th>
<th>RSQUARE</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AIR</td>
<td>predict</td>
<td>144</td>
<td>10.579085435</td>
<td>3.0845016398</td>
<td>8.0064787209</td>
<td>0.9921692375</td>
<td>679.35714621</td>
</tr>
</tbody>
</table>

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;```
Example 18.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc Forecast

Output 18.1.2 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>VAR</em></th>
<th><em>CALL</em></th>
<th><em>UTAG</em></th>
<th><em>VIX</em></th>
<th><em>VALUE</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>111.44700541</td>
</tr>
<tr>
<td>3</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0.5529945856</td>
</tr>
<tr>
<td>4</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>10.691036547</td>
</tr>
<tr>
<td>5</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>90.492958825</td>
</tr>
<tr>
<td>6</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>132.401052</td>
</tr>
<tr>
<td>7</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>118</td>
</tr>
<tr>
<td>8</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>119.94581467</td>
</tr>
<tr>
<td>9</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>-1.945814668</td>
</tr>
<tr>
<td>10</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>10.691036547</td>
</tr>
<tr>
<td>11</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>98.991768079</td>
</tr>
<tr>
<td>12</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>140.89986126</td>
</tr>
<tr>
<td>13</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>132</td>
</tr>
<tr>
<td>14</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>135.17351431</td>
</tr>
<tr>
<td>15</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>-3.173514311</td>
</tr>
<tr>
<td>16</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>10.691036547</td>
</tr>
<tr>
<td>17</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>114.21946772</td>
</tr>
<tr>
<td>18</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>156.1275609</td>
</tr>
</tbody>
</table>

The sequence that is displayed in Output 18.1.2 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 18.2: 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
Chapter 18: Utility Package

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 actualAir 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 18.1 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:

```plaintext
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.

```plaintext
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:

```plaintext
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 rcl 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;
```
Example 18.2: Computing Prediction Standard Errors and Confidence Limits for an Ad Hoc Forecast

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 18.1, 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;
Chapter 18: Utility Package

/* 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 18.2.1 Sample of the Six Ad Hoc Series Collected by the OUTNVP Object

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>VAR</em></th>
<th><em>CALL</em></th>
<th><em>UTAG</em></th>
<th><em>VIX</em></th>
<th><em>VALUE</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>3</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>5.0408057116</td>
</tr>
<tr>
<td>5</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>107.12020235</td>
</tr>
<tr>
<td>6</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>126.87979765</td>
</tr>
<tr>
<td>7</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>118</td>
</tr>
<tr>
<td>8</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>124</td>
</tr>
<tr>
<td>9</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>5.0408057116</td>
</tr>
<tr>
<td>11</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>114.12020235</td>
</tr>
<tr>
<td>12</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>133.87979765</td>
</tr>
<tr>
<td>13</td>
<td>AIR</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>132</td>
</tr>
<tr>
<td>14</td>
<td>predict</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>135</td>
</tr>
<tr>
<td>15</td>
<td>error</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>stderr</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>5.0408057116</td>
</tr>
<tr>
<td>17</td>
<td>lcl</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>125.12020235</td>
</tr>
<tr>
<td>18</td>
<td>ucl</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>144.87979765</td>
</tr>
</tbody>
</table>

The sequence displayed in Output 18.2.1 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 18.2.1 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-MacCurdy estimation, 323
CPANEL procedure, 323
ANCOVA effects
Shared Concepts, 51
ANOVA effects
Shared Concepts, 51
AR tests
CPANEL procedure, 340
ARIMA model
Box-Jenkins model, 69
at sign (@) operator
Shared Concepts, 49, 52
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, 60
bar (|) operator
Shared Concepts, 48, 51, 52
between-groups regression, 313
CPANEL procedure, 313
instrumental variables, 325
bounds on parameter estimates, 373
BOUNDS statement, 373
Box-Jenkins model, see ARIMA model
BY groups
CARIMA procedure, 74
CCDM procedure, 115
CESM procedure, 204
CNTSELECT procedure, 247
CSPATIALREG procedure, 413
SEVSELECT procedure, 658
TSMODEL procedure, 810
CAL, 191
calibration estimation, 191
canonical maximum likelihood estimation, 190
CARIMA procedure
B-spline basis, 32
BY groups, 74
computational method, 63, 64
convergence criterion, 41–45
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
input data tables, 73
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
parameter-based convergence criteria, 41, 44, 45
splines and spline bases, 30
truncated power function (TPF) basis, 31
CCDM procedure
B-spline basis, 32
BY groups, 115
computational method, 63, 64
convergence criterion, 41–45
descriptive statistics, 141
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
input data tables, 111
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34, 146
optimization techniques, 44, 64
parameter perturbation analysis, 140
parameter-based convergence criteria, 41, 44, 45
scenario analysis, 124
simulating aggregate adjusted loss distribution, 131
simulating aggregate loss distribution, 125
splines and spline bases, 30
truncated power function (TPF) basis, 31
CCOPULA procedure
B-spline basis, 32
computational method, 63, 64
convergence criterion, 41–45
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
output table names, 194
overview, 166
parameter-based convergence criteria, 41, 44, 45
splines and spline bases, 30
syntax, 170
truncated power function (TPF) basis, 31
censored regression models
CQLIM procedure, 385
censoring and truncation
SEVSELECT procedure, 688
CESM procedure
B-spline basis, 32
BY groups, 204
computational method, 63, 64
convergence criterion, 41–45
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
input data tables, 203
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
parameter-based convergence criteria, 41, 44, 45
splines and spline bases, 30
truncated power function (TPF) basis, 31
class level
CPANEL procedure, 339
CLASS statement
Shared Concepts, 45
syntax (Shared Concepts), 14, 45
classification variables
Shared Concepts, 45
Clayton copula, 185
CMLE, 190
CNTSELECT procedure
B-spline basis, 32
bounds on parameter estimates, 246
BY groups, 247
computational method, 63, 64
convergence criterion, 41–45
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
output table names, 280
parameter-based convergence criteria, 41, 44, 45
restrictions on parameter estimates, 255
splines and spline bases, 30
syntax, 241
truncated power function (TPF) basis, 31
colon (:) operator
Shared Concepts, 50
comparison tables
CPANEL procedure, 296, 341
compound distribution modeling
CCDM procedure, 90
computational method
CARIMA procedure, 63, 64
CCDM procedure, 63, 64
CCOPULA procedure, 63, 64
CESM procedure, 63, 64
CNTSELECT procedure, 63, 64
CPANEL procedure, 63, 64
CQLIM procedure, 63, 64
CSPATIALREG procedure, 63, 64
HMM procedure, 63, 64
SEVSELECT procedure, 63, 64
conjugate-gradient optimization method, 246
continuous-by-class effects
Shared Concepts, 53
continuous-nesting-class effects
Shared Concepts, 53
convergence criterion
CARIMA procedure, 41–45
CCDM procedure, 41–45
CCOPULA procedure, 41–45
CESM procedure, 41–45
CNTSELECT procedure, 41–45
CPANEL procedure, 41–45
CQLIM procedure, 41–45
CSPATIALREG procedure, 41–45
HMM procedure, 41–45
SEVSELECT procedure, 41–45
Shared Concepts, 41–45
copulas
Clayton, 185
Frank, 185
Gumbel, 185
normal, 181
Student’s \( t \), 182
correlated variables
CPANEL procedure, 298
correlation matrix
CPANEL procedure, 341
covariance matrix
CPANEL procedure, 340
covariates
heteroscedasticity models, 378
CPANEL procedure, 286
Amemiya-MacCurdy estimation, 323
AR tests, 340
B-spline basis, 32
between-groups regression, 313
class level, 339
comparison tables, 296, 341
computational method, 63, 64
convergence criterion, 41–45
correlated variables, 298
correlation matrix, 341
covariance matrix, 340
displayed output, 339
dynamic panel estimation, 326
endogenous variables, 300
fit statistics, 339
fixed-effects test, 339
Fuller-Battese variance components, 317, 321
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
Hausman test, 336, 339
Hausman-Taylor estimation, 322
ID variables, 301
input data tables, 295
instrumental variables, 302, 324
instruments table, 340
linear hypothesis testing, 337
model comparisons, 296
multithreading, 63
natural cubic spline basis, 34
Nerlove variance components, 318, 321
ODS table names, 34, 341
one-way fixed-effects by first differencing, 314
one-way fixed-effects model, 313
one-way random-effects model, 315
optimization techniques, 44, 64
output data tables, 308
parameter estimates, 340
parameter-based convergence criteria, 41, 44, 45
partitioning data, 295
poolability test, 339
poolability test, fixed effects, 336
pooled regression, 313
predetermined variables, 308
predicted values, 308
printing options, 295
R-square measure, 338
residuals, 308
Sargan test, 340
selecting variance component methods, 318
splines and spline bases, 30
structural summary, 340
Swamy-Arora variance components, 316, 319
tests of linear hypotheses, 341
truncated power function (TPF) basis, 31
two-way fixed-effects model, 314
two-way random-effects model, 318
variance components, 339
Wallace-Hussain variance components, 316, 320
Wansbeek-Kapteyn variance components, 317, 320
CQLIM procedure, 366
B-spline basis, 32
BY groups, 374
censored regression models, 385
computational method, 63, 64
convergence criterion, 41–45
frontier, 386
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
heteroscedasticity, 388
input data tables, 372
limited dependent variable models, 385
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
ordinal discrete choice modeling, 384
output, 389
output data tables, 381
output table names, 394
parameter-based convergence criteria, 41, 44, 45
splines and spline bases, 30
syntax, 370
tests on parameters, 389
truncated power function (TPF) basis, 31
truncated regression models, 386
crossed effects
Shared Concepts, 52
CSPATIALREG procedure
B-spline basis, 32
bounds on parameter estimates, 412
BY groups, 413
computational method, 63, 64
convergence criterion, 41–45
function-based convergence criteria, 41, 42
gradient-based convergence criteria, 41, 43
input data tables, 409
multithreading, 63
natural cubic spline basis, 34
ODS table names, 34
optimization techniques, 44, 64
output data tables, 417
output table names, 442
parameter-based convergence criteria, 41, 44, 45
restrictions on parameter estimates, 418
splines and spline bases, 30
syntax, 406
truncated power function (TPF) basis, 31
dash (-) operator
Shared Concepts, 50
data
fit, 181
data processing and fitting, 181
data table output
HMM procedure, 599
defining a custom objective function
SEVSELECT procedure, 733
defining a custom probability distribution
   SEVSELECT procedure, 709
dependence measures, 180
descriptive statistics
   CCDM procedure, 141
DIAG function
   creating a diagonal matrix from a vector, 597
   extracting the diagonal elements of a matrix, 597
DISPLAY statement
   syntax (Shared Concepts), 18
displayed output
   CPANEL procedure, 339
DISPLAYOUT statement
   syntax (Shared Concepts), 19
double dash (- -) operator
   Shared Concepts, 50
double-dogleg optimization method, 246
dummy parameterization
   Shared Concepts, 50
dynamic panel estimation, 326
   CPANEL procedure, 326
ECM procedure
   input data tables, 474
   ODS table names, 491
economic capital modeling
   ECM procedure, 465
effect parameterization
   Shared Concepts, 55
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, 698
endogenous variables
   CPANEL procedure, 300
exact maximum likelihood estimation, 191
example of Gaussian hidden Markov model
   HMM procedure, 514
example of Gaussian hidden Markov model for cross-sectional time series data
   HMM procedure, 526
example of Gaussian mixture hidden Markov model
   HMM procedure, 529
example of Gaussian mixture hidden Markov model for cross-sectional time series data
   HMM procedure, 529
example of regime-switching autoregression model
   HMM procedure, 555
example of regime-switching regression model
   HMM procedure, 549
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, 442
   multimember effect, 23
examples, HMM
   multimember effect, 23
examples, SEVSELECT
   multimember effect, 23
fit statistics
   CPANEL procedure, 339
fitting custom probability distributions
   SEVSELECT procedure, 661
fixed-effects model
   poolability test, 336
fixed-effects test
   CPANEL procedure, 339
forward selection
   Shared Concepts, 58
forward swap selection
   Shared Concepts, 61
Frank copula, 185
Fuller-Battese variance components
   CPANEL procedure, 317, 321
function-based convergence criteria
   CARIMA procedure, 41, 42
   CCDM procedure, 41, 42
   CCOPULA procedure, 41, 42
   CESM procedure, 41, 42
   CNTSELECT procedure, 41, 42
   CPANEL procedure, 41, 42
   CQLIM procedure, 41, 42
   CSPATIALREG procedure, 41, 42
   HMM procedure, 41, 42
   SEVSELECT procedure, 41, 42
   Shared Concepts, 41, 42
Gaussian hidden Markov model
   HMM procedure, 578
Gaussian mixture hidden Markov model
   HMM procedure, 578
general effects
 Shared Concepts, 54
generalized method of moments (GMM), 326
GLM parameterization
 Shared Concepts, 50
gradient-based convergence criteria
 CARIMA procedure, 41, 43
CCDM procedure, 41, 43
CCOPULA procedure, 41, 43
CESM procedure, 41, 43
CNTSELECT procedure, 41, 43
CPANEL procedure, 41, 43
CQLIM procedure, 41, 43
CSPATIALREG procedure, 41, 43
HMM procedure, 41, 43
SEVSELECT procedure, 41, 43
 Shared Concepts, 41, 43
Gumbel copula, 185
Hausman specification test, 336
 random-effects model, 336
Hausman test
 CPANEL procedure, 336, 339
Hausman-Taylor estimation, 322
 CPANEL procedure, 322
heteroscedasticity models
covariates, 378
hidden Markov model
 HMM procedure, 575
HMM procedure
 B-spline basis, 32
 computational method, 63, 64
 convergence criterion, 41–45
data table output, 599
 example of Gaussian hidden Markov model, 514
 example of Gaussian hidden Markov model for
cross-sectional time series data, 526
 example of Gaussian mixture hidden Markov
model, 529
 example of Gaussian mixture hidden Markov
model for cross-sectional time series data,
529
 example of regime-switching autoregression
model, 555
 example of regime-switching regression model,
549
 function-based convergence criteria, 41, 42
Gaussian hidden Markov model, 578
Gaussian mixture hidden Markov model, 579
gradient-based convergence criteria, 41, 43
 hidden Markov model, 575
 information criteria, 598
 input data tables, 562
 matrix expression, 588
 missing values, 598
 multithreading, 63
 natural cubic spline basis, 34
 ODS table names, 34, 602
 optimization algorithms, 587
 optimization techniques, 44, 64
 parameter estimation methods, 583
 parameter-based convergence criteria, 41, 44, 45
 regime-switching autoregression model, 581
 regime-switching regression model, 579
 splines and spline bases, 30
 truncated power function (TPF) basis, 31
I function
 creating an identity matrix, 597
ID variables
 CPANEL procedure, 301
information criteria
 HMM procedure, 598
instrumental variables
 between-groups regression, 325
 CPANEL procedure, 302, 324
 one-way fixed effects, 325
 one-way random effects, 326
 pooled regression, 324
 regression, 324
instruments table
 CPANEL procedure, 340
interaction effects
 Shared Concepts, 52
intercept
 Shared Concepts, 51
J function
 creating a matrix of identical values, 598
Kaplan-Meier’s EDF estimator
 SEVSELECT procedure, 700
Lagrange multiplier test
 nonlinear hypotheses, 389
levelization
 Shared Concepts, 45
limited dependent variable models
 CQLIM procedure, 385
linear hypothesis testing, 337
 CPANEL procedure, 337
loss distribution modeling
 SEVSELECT procedure, 628
main effects
 Shared Concepts, 51
matrix expression
 HMM procedure, 588
measure
dependence, 180
missing values
  HMM procedure, 598
MLE, 191
model comparisons
  CPANEL procedure, 296
modified Kaplan-Meier’s EDF estimator
  SEVSELECT procedure, 700
multithreading
  CARIMA procedure, 63
  CCDM procedure, 63
  CCOPULA procedure, 63
  CESM procedure, 63
  CNTSELECT procedure, 63
  CPANEL procedure, 63
  CQLIM procedure, 63
  CSPATIALREG procedure, 63
  HMM procedure, 63
  SEVSELECT procedure, 63
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
Nerlove variance components
  CPANEL procedure, 318, 321
nested effects
  Shared Concepts, 52
nested versus crossed effects
  Shared Concepts, 52
Newton-Raphson optimization method, 246, 412
Newton-Raphson Ridge optimization method, 246
nonlinear hypotheses
  Lagrange multiplier test, 389
nonsingular parameterization
  Shared Concepts, 55
normal copula, 181
ODS graph names
  SEVSELECT procedure, 744
ODS table names
  CARIMA procedure, 34
  CCDM procedure, 34, 146
  CCOPULA procedure, 34
  CESM procedure, 34
  CNTSELECT procedure, 34
  CPANEL procedure, 34
  CQLIM procedure, 34
  CSPATIALREG procedure, 34
  ECM procedure, 491
  HMM procedure, 34, 602
  SEVSELECT procedure, 34, 742
optimization algorithms
  HMM procedure, 587
optimization methods
  conjugate-gradient, 246
  double-dogleg, 246
  Newton-Raphson, 246
  Newton-Raphson Ridge, 246
  none, 246
  quasi-Newton, 246
  trust region, 246
optimization techniques
  CARIMA procedure, 44, 64
  CCDM procedure, 44, 64
  CCOPULA procedure, 44, 64
  CESM procedure, 44, 64
  CNTSELECT procedure, 44, 64
  CPANEL procedure, 44, 64
  CQLIM procedure, 44, 64
  CSPATIALREG procedure, 44, 64
  HMM procedure, 44, 64
  SEVSELECT procedure, 44, 64
  Shared Concepts, 44
options summary
  CLASS statement, 15
  EFFECT statement, 21, 250, 664
  MODEL statement (CPANEL), 303
ordering
  of class levels (Shared Concepts), 46
ordinal discrete choice modeling
  CQLIM procedure, 384
ordinal parameterization
  Shared Concepts, 55
ORTHEFFECT parameterization
  Shared Concepts, 56
ORTHORDINAL parameterization
  Shared Concepts, 56
ORTHOTHERM parameterization
  Shared Concepts, 56
ORTHPOLY parameterization
Subject Index

Shared Concepts, 57
ORTHREF parameterization
  Shared Concepts, 57
output table names
  CCOPULA procedure, 194
  CNTSELECT procedure, 280
  CQLIM procedure, 394
  CSPATIALREG procedure, 442

parameter estimates
  CPANEL procedure, 340
parameter estimation methods
  HMM procedure, 583
parameter perturbation analysis
  CCDM procedure, 140
parameter-based convergence criteria
  CARIMA procedure, 41, 44, 45
  CCDM procedure, 41, 44, 45
  CCOPULA procedure, 41, 44, 45
  CESM procedure, 41, 44, 45
  CNTSELECT procedure, 41, 44, 45
  CPANEL procedure, 41, 44, 45
  CQLIM procedure, 41, 44, 45
  CSPATIALREG procedure, 41, 44, 45
  HMM procedure, 41, 44, 45
  SEVSELECT procedure, 41, 44, 45
  Shared Concepts, 41, 44, 45
parameterization
  dummy (Shared Concepts), 50
  effect (Shared Concepts), 55
  GLM (Shared Concepts), 50
  nonsingular (Shared Concepts), 55
  ordinal (Shared Concepts), 55
  ORTHEFFECT (Shared Concepts), 56
  ORTHORDINAL (Shared Concepts), 56
  ORTHOTHERM (Shared Concepts), 56
  ORTHPOLY (Shared Concepts), 57
  ORTHREF (Shared Concepts), 57
  polynomial (Shared Concepts), 57
  reference (Shared Concepts), 55
  Shared Concepts, 47
  thermometer (Shared Concepts), 55
partitioning data
  CPANEL procedure, 295
polynomial effects
  Shared Concepts, 51
polynomial parameterization
  Shared Concepts, 56
poolability test, 336
  CPANEL procedure, 339
  fixed-effects model, 336
poolability test, fixed effects
  CPANEL procedure, 336
pooled regression, 313
CPANEL procedure, 313
  instrumental variables, 324
predicted variables
  CPANEL procedure, 308
predicted values
  CPANEL procedure, 308
printing options
  CPANEL procedure, 295
probability of observability
  SEVSELECT procedure, 689
quasi-Newton optimization method, 246, 412
R-square measure
  CPANEL procedure, 338
random-effects model
  Hausman specification test, 336
  reference parameterization
    Shared Concepts, 55
regime-switching autoregression model
  HMM procedure, 581
regime-switching regression model
  HMM procedure, 579
regression effects
  Shared Concepts, 51
residuals
  CPANEL procedure, 308
Sargan test
  CPANEL procedure, 340
scale regression model
  SEVSELECT procedure, 693
scenario analysis
  CCDM procedure, 124
scoring functions
  SEVSELECT procedure, 726
selecting variance component methods
  CPANEL procedure, 318
SELECTION statement
  syntax (Shared Concepts), 35
SEVSELECT procedure
  B-spline basis, 32
  BY groups, 658
  censoring and truncation, 688
  computational method, 63, 64
  convergence criterion, 41–45
  defining a custom objective function, 733
  defining a custom probability distribution, 709
  empirical distribution function, 698
  function-based convergence criteria, 41, 42
  gradient-based convergence criteria, 41, 43
  input data tables, 650
  multithreading, 63
  natural cubic spline basis, 34
  ODS graph names, 744
ODS table names, 34, 742
optimization techniques, 44, 64
output data tables, 668
parameter-based convergence criteria, 41, 44, 45
predefined distributions, 678
predefined utility functions, 721
probability of observability, 689
scale regression model, 693
scoring functions, 726
spline and spline bases, 30
statistics of fit, 704
truncated power function (TPF) basis, 31
Tweedie distribution, 680
SHAPE function
creating a matrix with different dimension, 598
Shared Concepts
ABSCONV option, 41
ABSFCONV option, 41
ABSFITOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXTOL option, 41
ANCOVA effects, 51
ANOVA effects, 51
at sign (@) operator, 49, 52
backward elimination, 60
bar (|) operator, 48, 51, 52
CLASS statement, 14, 45
classification variables, 45
collection effect (EFFECT statement), 22
colon (: ) operator, 50
continuous-by-class effects, 53
continuous-nesting-class effects, 53
convergence criterion, 41–45
crossed effects, 52
dash (-) operator, 50
DISPLAY statement, 18
DISPLAYOUT statement, 19
double dash (- -) operator, 50
dummy parameterization, 50
effect parameterization, 55
EFFECT statement, 20
FCONV option, 42
FCONV2 option, 42
forward selection, 58
forward swap selection, 61
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
function-based convergence criteria, 41, 42
GCONV option, 43
GCONV2 option, 43
general effects, 54
GLM parameterization, 50
gradient-based convergence criteria, 41, 43
GTOL option, 43
GTOL2 option, 43
interaction effects, 52
intercept, 51
levelization, 45
main effects, 51
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 43
MINITER= option, 44
missing values, CLASS variables, 46
multimember effect (EFFECT statement), 22
nested effects, 52
nested versus crossed effects, 52
nonsingular parameterization, 55
optimization techniques, 44
ORDER= option, 46
ordering of class levels, 46
ordinal parameterization, 55
ORTHEFFECT parameterization, 56
ORTHORDINAL parameterization, 56
ORTHOTHERM parameterization, 56
ORTHPOLY parameterization, 57
ORTHREF parameterization, 57
parameter-based convergence criteria, 41, 44, 45
parameterization, 47
polynomial effect (EFFECT statement), 24
polynomial effects, 51
polynomial parameterization, 56
reference parameterization, 55
regression effects, 51
SELECTION statement, 35
singular parameterization, 52
sort order of class levels, 46
spline basis, B-spline, 32
spline basis, natural cubic spline, 34
spline basis, truncated power function, 31
spline effect (EFFECT statement), 27
stepwise selection, 61
TECHNIQUE= option, 44
thermometer parameterization, 55
XCONV option, 44
XSIZE option, 45
XTOL option, 44
simulating aggregate adjusted loss distribution
CCDM procedure, 131
simulating aggregate loss distribution
CCDM procedure, 125
singular parameterization
Shared Concepts, 52
Sklar’s theorem, 179
sort order
  of class levels (Shared Concepts), 46

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

statistics of fit
  SEVSELECT procedure, 704

stepwise selection
  Shared Concepts, 61

structural summary
  CPANEL procedure, 340

Student’s t copula, 182

Swamy-Arora variance components
  CPANEL procedure, 316, 320

tests of linear hypotheses
  CPANEL procedure, 341

thermometer parameterization
  Shared Concepts, 55

TPF basis, see truncated power function (TPF) basis

t 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, 386

trust region optimization method, 246, 412

TSMODEL procedure
  BY groups, 810
  input data tables, 806
  time series analysis, 797

Turnbull’s EDF estimator
  SEVSELECT procedure, 701

Tweedie distribution
  SEVSELECT procedure, 680

two-stage least squares (2SLS)
  regression, 322–324

two-way fixed-effects model, 314

two-way random-effects model, 318
  CPANEL procedure, 318

variance components
  CPANEL procedure, 339

Wallace-Hussain variance components
  CPANEL procedure, 316, 320

Wansbeek-Kapteyn variance components
  CPANEL procedure, 317, 320
Syntax Index

ABSCONV option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ABSFCONV option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ABSFTOL option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ABSFXTOL option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ABSGCONV option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ABSGXTOL option
PROC CARIMA statement, 41
PROC CCDM statement, 41
PROC CCOPULA statement, 41
PROC CESM statement, 41
PROC CNTSELECT statement, 41
PROC CPANEL statement, 41
PROC CQLIM statement, 41
PROC CSPATIALREG statement, 41
PROC HMM statement, 41
PROC SEVSELECT statement, 41

ACCUMULATE function
TSA package, 841

ACCUMULATE2 function
TSA package, 844

ACCUMULATE= option
FORECAST statement (CESM), 205
ID statement (CARIMA), 77
ID statement (CESIM), 207
<table>
<thead>
<tr>
<th>Syntax Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID statement (TSMODEL), 814</td>
</tr>
<tr>
<td>VAR statement (TSMODEL), 819</td>
</tr>
<tr>
<td>ACF function</td>
</tr>
<tr>
<td>TSA package, 846</td>
</tr>
<tr>
<td>AddARPol method</td>
</tr>
<tr>
<td>TSM Package, 898</td>
</tr>
<tr>
<td>AddBlockSeason method</td>
</tr>
<tr>
<td>TSM Package, 915</td>
</tr>
<tr>
<td>AddComponent method</td>
</tr>
<tr>
<td>TSM Package, 916</td>
</tr>
<tr>
<td>AddCycle method</td>
</tr>
<tr>
<td>TSM Package, 917</td>
</tr>
<tr>
<td>AddDeplag method</td>
</tr>
<tr>
<td>TSM Package, 917</td>
</tr>
<tr>
<td>AddExternal method</td>
</tr>
<tr>
<td>TSM Package, 887</td>
</tr>
<tr>
<td>AddInput method</td>
</tr>
<tr>
<td>TSM Package, 918</td>
</tr>
<tr>
<td>AddMAPoly method</td>
</tr>
<tr>
<td>TSM Package, 899</td>
</tr>
<tr>
<td>AddSeason method</td>
</tr>
<tr>
<td>TSM Package, 918</td>
</tr>
<tr>
<td>AddTF method</td>
</tr>
<tr>
<td>TSM Package, 899</td>
</tr>
<tr>
<td>AddTFDenPoly method</td>
</tr>
<tr>
<td>TSM Package, 900</td>
</tr>
<tr>
<td>AddTFNumPoly method</td>
</tr>
<tr>
<td>TSM Package, 901</td>
</tr>
<tr>
<td>AddX method</td>
</tr>
<tr>
<td>TSM Package, 888</td>
</tr>
<tr>
<td>ADJSAMPLEVAR= option</td>
</tr>
<tr>
<td>OUTPUT statement (CCDM), 119</td>
</tr>
<tr>
<td>ADJUSTEDSEVERITY= option</td>
</tr>
<tr>
<td>PROC CCDM statement, 111</td>
</tr>
<tr>
<td>ALGORITHM= option</td>
</tr>
<tr>
<td>OPTIMIZE statement (HMM), 570</td>
</tr>
<tr>
<td>ALIGN= option</td>
</tr>
<tr>
<td>ID statement (CARIMA), 78</td>
</tr>
<tr>
<td>ID statement (CESM), 208</td>
</tr>
<tr>
<td>ID statement (TSMODEL), 815</td>
</tr>
<tr>
<td>ALL option</td>
</tr>
<tr>
<td>TEST statement (CNTSELECT), 259</td>
</tr>
<tr>
<td>TEST statement (CPANEL), 310</td>
</tr>
<tr>
<td>TEST statement (CQLIM), 383</td>
</tr>
<tr>
<td>TEST statement (CSPATIALREG), 420</td>
</tr>
<tr>
<td>ALPHA= number</td>
</tr>
<tr>
<td>FORECAST statement (CARIMA), 76</td>
</tr>
<tr>
<td>ALPHA= option</td>
</tr>
<tr>
<td>FORECAST statement (CESM), 205</td>
</tr>
<tr>
<td>FORECAST statement (HMM), 566</td>
</tr>
<tr>
<td>AMACURDY option</td>
</tr>
<tr>
<td>MODEL statement (CPANEL), 305</td>
</tr>
<tr>
<td>APPROXIMATION= option</td>
</tr>
<tr>
<td>PROC CSPATIALREG statement, 410</td>
</tr>
<tr>
<td>ARIMASPEC object</td>
</tr>
<tr>
<td>TSM Package, 895</td>
</tr>
<tr>
<td>ARMAORDERS function</td>
</tr>
<tr>
<td>TSA package, 848</td>
</tr>
<tr>
<td>Autoreg method</td>
</tr>
<tr>
<td>TSM Package, 915</td>
</tr>
<tr>
<td>AUXDATA= option</td>
</tr>
<tr>
<td>PROC TSMODEL statement, 806</td>
</tr>
<tr>
<td>BACK= option</td>
</tr>
<tr>
<td>FORECAST statement (CESM), 205</td>
</tr>
<tr>
<td>FORECAST statement (HMM), 566</td>
</tr>
<tr>
<td>BASIS option</td>
</tr>
<tr>
<td>EFFECT statement, spline (CARIMA), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CCDM), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CCOPULA), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CESM), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CNTSELECT), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CPANEL), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CQLIM), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (CSPATIALREG), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (HMM), 27</td>
</tr>
<tr>
<td>EFFECT statement, spline (SEVSELECT), 27</td>
</tr>
<tr>
<td>BOUNDS statement</td>
</tr>
<tr>
<td>CNTSELECT procedure, 246</td>
</tr>
<tr>
<td>CQLIM procedure, 373</td>
</tr>
<tr>
<td>CSPATIALREG procedure, 412</td>
</tr>
<tr>
<td>BTWNG option</td>
</tr>
<tr>
<td>MODEL statement (CPANEL), 305</td>
</tr>
<tr>
<td>BY statement</td>
</tr>
<tr>
<td>CARIMA procedure, 74</td>
</tr>
<tr>
<td>CCDM procedure, 115</td>
</tr>
<tr>
<td>CCOPULA procedure, 172</td>
</tr>
<tr>
<td>CESM procedure, 204</td>
</tr>
<tr>
<td>CNTSELECT procedure, 247</td>
</tr>
<tr>
<td>CPANEL procedure, 295</td>
</tr>
<tr>
<td>CQLIM procedure, 374</td>
</tr>
<tr>
<td>CSPATIALREG procedure, 413</td>
</tr>
<tr>
<td>SEVSELECT procedure, 658</td>
</tr>
<tr>
<td>TSMODEL procedure, 810</td>
</tr>
<tr>
<td>CARIMA procedure, 72</td>
</tr>
<tr>
<td>syntax, 72</td>
</tr>
<tr>
<td>CARIMA procedure, CLASS statement</td>
</tr>
<tr>
<td>DESCENDING option, 15</td>
</tr>
<tr>
<td>MISSING option, 15</td>
</tr>
<tr>
<td>ORDER= option, 15</td>
</tr>
<tr>
<td>PARAM= option, 15</td>
</tr>
<tr>
<td>REF= option, 16</td>
</tr>
<tr>
<td>SPLIT option, 17</td>
</tr>
<tr>
<td>CARIMA procedure, EFFECT statement</td>
</tr>
<tr>
<td>BASIS option (spline), 27</td>
</tr>
<tr>
<td>DATABOUNDARY option (spline), 27</td>
</tr>
<tr>
<td>DEGREE option (polynomial), 24</td>
</tr>
</tbody>
</table>
DEGREE option (spline), 27  
DETAILS option (multimember), 23  
DETAILS option (polynomial), 24  
DETAILS option (spline), 27  
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), 29  
SPLIT option (spline), 30  
STANDARDIZE option (polynomial), 25  
CARIMA procedure, PROC CARIMA statement  
ABSCONV option, 41  
ABSFCONV option, 41  
ABSFFTOL option, 41  
ABSGCONV option, 41  
ABSGTOL option, 41  
ABSTOL option, 41  
ABSXCONV option, 41  
ABSXTOL option, 41  
DATA= option, 73  
FCONV option, 42  
FCONV2 option, 42  
FSIZE option, 42  
FTOL option, 42  
FTOL2 option, 42  
GCONV option, 43  
GCONV2 option, 43  
GTOL option, 43  
GTOL2 option, 43  
MAXFUNC= option, 43  
MAXITER= option, 43  
MAXTIME= option, 44  
MINITER= option, 44  
TECHNIQUE= option, 44  
XCONV option, 44  
XSIZE option, 45  
XTOL option, 44  
CARIMA procedure, SELECTION statement  
CHOOSE= option, 36  
COMPETITIVE option, 36  
CRITERION= option, 36  
DETAILS= option, 37, 38  
HIERARCHY= option, 37  
MAXEFFECTS= option, 36  
MAXSTEPS= option, 36  
METHOD= option, 35  
MINEFFECTS= option, 36  
ORDERSELECT option, 38  
SELECT= option, 36  
SELECTION= option, 40  
STOP= option, 36  
STOPHORIZON= option, 40  
CASESENSITIVE option  
DISPLAY statement (CCDM), 19, 116  
DISPLAY statement (CCOPULA), 19, 174  
DISPLAY statement (CNTSELECT), 19, 248  
DISPLAY statement (CPANEL), 19, 299  
DISPLAY statement (CQLIM), 19, 375  
DISPLAY statement (CSpatialReg), 19, 414  
DISPLAY statement (ECM), 19, 480  
DISPLAY statement (HMM), 19, 564  
DISPLAY statement (SEVSELECT), 19, 660  
CCDM procedure, 109  
EXTERNALCOUNTS statement, 117  
OUTPUT statement, 118  
OUTSUM statement, 119  
SEVERITYMODEL statement, 123  
syntax, 109  
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, 19, 116  
EXCLUDE option, 19, 116  
EXCLUDEALL option, 19, 116  
TRACE option, 19, 116  
CCDM procedure, DISPLAYOUT statement  
INCLUDEALL option, 19, 117  
NOREPLACE option, 19, 117  
REPEATED option, 20, 117  
CCDM procedure, EFFECT statement  
BASIS option (spline), 27  
DATABOUNDARY option (spline), 27  
DEGREE option (polynomial), 24  
DEGREE option (spline), 27  
DETAILS option (multimember), 23  
DETAILS option (polynomial), 24  
DETAILS option (spline), 27  
KNOTMAX= option (spline), 28  
KNOTMETHOD option (spline), 28  
KNOTMIN= option (spline), 29  
MDEGREE option (polynomial), 25  
NATURALCUBIC option (spline), 29  
NOEFFECT option (multimember), 23  
NOEFFECT option (polynomial), 25  
NOEFFECT option (spline), 29  
SEPARATE option (spline), 30  
STANDARDIZE option (polynomial), 25  
CCDM procedure, EXTERNALCOUNTS statement  
COUNT= option, 118
ID= option, 118

CCDM procedure, OUTPUT statement
ADJSAMPLEVAR= option, 119
OUT= option, 113, 118
PERTURBOUT option, 119
SAMPLEVAR= option, 119

CCDM procedure, OUTSUM statement
OUT= option, 120
PCTLNAME= option, 121
PCTLNDEC= option, 122
PCTLPTS= option, 121

CCDM procedure, PROC CCDM statement, 111
ABS_CONV option, 41
ABSF_CONV option, 41
ABSF_TOL option, 41
ABSG_CONV option, 41
ABSG_TOL option, 41
ABSTOL option, 41
ABSTOL option, 41
ABS_X_CONV option, 41
ABS_X_TOL option, 41
ADJUSTED_SEVERITY= option, 111
COUNTSTORE= option, 111
DATA= option, 111
FC_CONV option, 42
FC_CONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
G_CONV option, 43
G_CONV2 option, 43
GTOL option, 43
GTOL2 option, 43
IGNORECOV option, 112
MAXCOUNTDRAW= option, 112
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
NOPRINT option, 112
NPERTURBEDSAMPLES= option, 112
NREPLICATES= option, 113
PCTLDEF= option, 113
PRINT= option, 113
SEED= option, 114
SEVERITYEST= option, 114
SEVERITYSTORE= option, 115
TECHNIQUE= option, 44
TRUNCATEZEROS option, 115
VARDEF= option, 115
X_CONV option, 44
X_SIZE option, 45
XTOL option, 44

CCDM procedure, SELECTION statement
CHOOSE= option, 36

COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40

CCDM procedure, SEVERITYMODEL statement
DEFINITIONSTABLE= option, 123

CCF function
TSA package, 849

CCOPULA procedure, 170
BY statement, 172
DEFINE statement, 172
FIT statement, 175
PROC CCOPULA statement, 172
SIMULATE statement, 177
syntax, 170
VAR statement, 178
VIEWSTORE statement, 178

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, 19, 174
EXCLUDE option, 19, 174
EXCLUDEALL option, 19, 174
TRACE option, 19, 174

CCOPULA procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 175
NOPLACE option, 19, 175
REPEATED option, 20, 175

CCOPULA procedure, EFFECT statement
BASIS option (spline), 27
DATABOUNDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25

CCOPULA procedure, PROC CCOPULA statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFTOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXSTOL option, 41
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
XSIZE option, 45
XTOL option, 44

CCOPULA procedure, SELECTION statement
CHOOSE= option, 36
COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40

CENSORED option
ENDOGENOUS statement (CQLIM), 376, 380

CESM procedure, 201
syntax, 201

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
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25

CESM procedure, PROC CESM statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFTOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXSTOL option, 41
DATA= option, 203
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
XSIZE option, 45
XTOL option, 44

CESM procedure, SELECTION statement
CHOOSE= option, 36
COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40
CHOOSE= option
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36, 258
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIMIT), 36
SELECTION statement (CSpatialReg), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36, 676
CLASS statement
CARIMA procedure, 14, 45
CCDM procedure, 14, 45
CCOPULA procedure, 14, 45
CESM procedure, 14, 45
CNTSELECT procedure, 14, 45, 247
CPANEL procedure, 14, 45
CQLIMIT procedure, 14, 45
CSpatialReg procedure, 14, 45, 413
HMM procedure, 14, 45
SEVSELECT procedure, 14, 45, 659
CLIMITS object, 950
Close method
TSM Package, 893, 901, 909, 919
CNTSELECT procedure, 241
syntax, 241
VIEWSTORE statement, 260
CNTSELECT procedure, CLASS statement, 247
DESCENDING option, 15
MISSING option, 15
ORDER= option, 15
PARAM= option, 15
REF= option, 16
SPLIT option, 17
CNTSELECT procedure, DISPLAY statement
CASESENSITIVE option, 19, 248
EXCLUDE option, 19, 248
EXCLUDEALL option, 19, 248
TRACE option, 19, 248
CNTSELECT procedure, DISPLAY statement
INCLUDEALL option, 19, 249
NOREPLACE option, 19, 249
REPEATED option, 20, 249
CNTSELECT procedure, EFFECT statement
BASIS option (spline), 27
DATABASEBOUNDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
KNOTMAX= option (spline), 28
KNOTMETHOD option (spline), 28
KNOTMIN= option (spline), 29
MEDEGREE option (polynomial), 25
NATURALCUBIC option (spline), 29
NOEFFECT option (multimember), 23
NOSEPARATE option (polynomial), 25
SEPARATE option (spline), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25
CNTSELECT procedure, PROC CNTSELECT statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFTOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXTOL option, 41
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
XSIZE option, 45
XTOL option, 44
CNTSELECT procedure, SELECTION statement, 257
CHOOSE= option, 36, 258
COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MEDEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35, 257
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36, 258
SELECTION= option, 36, 258
STOP= option, 36, 258
STOPHORIZON= option, 40
CNTSELECT procedure, TEST statement, 259
CNTSELECT procedure, WEIGHT statement, 261
COMMONPACKAGE option
OUTSCORELIB statement (SEVSELECT), 672
COMPARE statement
CPANEL procedure, 296
COMPETITIVE option
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIM), 36
SELECTION statement (CSPATIALREG), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36
Compute method, 951
CONDITIONAL
OUTPUT statement (CQLIM), 381
CONGRA option
METHOD= option, PROC CSPATIALREG statement, 412
CONVERGE= value
ESTIMATE statement (CARIMA), 75
COPULASIM= option
PROC ECM statement, 474
COPYVAR= option
OUTPUT statement (CNTSELECT), 254
OUTPUT statement (CPANEL), 308
OUTPUT statement (CQLIM), 381
OUTPUT statement (CSPATIALREG), 417
COPYVARS= option
OUTPUT statement (SEVSELECT), 668
CORRB option
CQLIM procedure, 372
MODEL statement, 253, 416
MODEL statement (CPANEL), 307
PROC CNTSELECT statement, 245
PROC CSPATIALREG statement, 410
CORRELATED statement
CPANEL procedure, 298
COST option
ENDOGENOUS statement (CQLIM), 377, 381
COUNT= option
EXTERNALCOUNTS statement (CCDM), 118
COUNTSTORE= option
PROC CCDM statement, 111
COVB option
CQLIM procedure, 372
MODEL statement, 253, 416
MODEL statement (CPANEL), 307
PROC CNTSELECT statement, 245
PROC CSPATIALREG statement, 410
COVEST= option
CQLIM procedure, 373
PROC CNTSELECT statement, 245
PROC CSPATIALREG statement, 411
COVOUT option
PROC SEVSELECT statement, 649
CPANEL procedure, 292
details, 311
syntax, 292
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, 296
MSTAT option, 296
OUTPARM= option, 297
OUTSTAT= option, 297
PSTAT option, 298
CPANEL procedure, CORRELATED statement, 298
CPANEL procedure, DISPLAY statement
CASESENSITIVE option, 19, 299
EXCLUDE option, 19, 299
EXCLUDEALL option, 19, 299
TRACE option, 19, 299
CPANEL procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 300
NOREPLACE option, 19, 300
REPEATED option, 20, 300
CPANEL procedure, EFFECT statement
BASIS option (spline), 27
DATABOUNDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25
CPANEL procedure, ENDOGENOUS statement, 300
CPANEL procedure, ID statement, 301
CPANEL procedure, INSTRUMENTS statement, 302
CPANEL procedure, MODEL statement, 303
AMACURDY option, 305
BTWNG option, 305
CORRB option, 307
COVB option, 307
DYNDIFF option, 305
DYNSYS option, 305
FDONE option, 305
FIXONE option, 305
FIXTWO option, 305
HTAYLOR option, 305
IVBTWNG option, 305
IVFIXONE option, 305
IVPOOLED option, 305
IVRANONE option, 305
NOINT option, 306
NOLABEL option, 307
POOLED option, 305
PRINTFIXED option, 307
RANONE option, 306
RANTWO option, 306
VCOMP= option, 306
CPANEL procedure, OUTPUT statement, 307
COPYVRAR= option, 308
OUT= option, 308
P= option, 308
PREDICTED= option, 308
R= option, 308
RESIDUAL= option, 308
CPANEL procedure, PREDETERMINED statement, 308
CPANEL procedure, PROC CPANEL statement, 294
ABSCONV option, 41
ABSFCONV option, 41
ABSFOTOL option, 41
ABSGCONV option, 41
ABSTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXTOL option, 41
DATA= option, 295
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
CPANEL procedure, RESTRICT statement, 309
CPANEL procedure, SELECTION statement
CHOOSE= option, 36
COMPEITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40
CPANEL procedure, TEST statement, 310
ALL option, 310
LM option, 310
LR option, 310
WALD option, 310
CQLIM procedure, 370
syntax, 370
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, 19, 375
EXCLUDE option, 19, 375
EXCLUDEALL option, 19, 375
TRACE option, 19, 375
CQLIM procedure, DISPLAYOUT statement
INCLUDALL option, 19, 376
NOREPLACE option, 19, 376
REPEATED option, 20, 376
CQLIM procedure, EFFECT statement
BASIS option (spline), 27
DATABOUNDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25
CQLIM procedure, FREQ statement, 377
CQLIM procedure, OUTPUT statement
OUT= option, 381
CQLIM procedure, PROC CQLIM statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFTOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSXCONV option, 41
ABSXTOL option, 41
DATA= option, 372
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
XSIZE option, 45
XTOL option, 44
CQLIM procedure, SELECTION statement
CHOOSE= option, 36
COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 36
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40
CQLIM procedure, TEST statement, 383
CQLIM procedure, WEIGHT statement, 384
Criterion method
TSM Package, 888
CRITERION= option
FORECAST statement (CESM), 205
PROC SEVSELECT statement, 650
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIM), 36
SELECTION statement (CSPATIALREG), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36
CSPATIALREG procedure, syntax, 406
CSPATIALREG procedure, CLASS statement, 413
DESCENDING option, 15
MISSING option, 15
ORDER= option, 15
PARAM= option, 15
REF= option, 16
SPLIT option, 17
CSPATIALREG procedure, DISPLAY statement
CASESENSITIVE option, 19, 414
EXCLUDE option, 19, 414
EXCLUDEALL option, 19, 414
TRACE option, 19, 414
CSPATIALREG procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 415
NOREPLACE option, 19, 415
REPEATED option, 20, 415
CSPATIALREG procedure, EFFECT statement
BASIS option (spline), 27
DATABASEBOUNDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25

CSPATIALREG procedure, OUTPUT statement
OUT= option, 417

CSPATIALREG procedure, PROC CSPATIALREG statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFSTOL option, 41
ABSGCONV option, 41
ABSGTOL option, 41
ABSTOL option, 41
DATA= option, 409
FCONV option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
FSIZE option, 45
XTOL option, 44

CSPATIALREG procedure, SELECTION statement
CHOOSE= option, 36
COMPETITIVE option, 36
CRITERION= option, 36
DETAILS= option, 37, 38
HIERARCHY= option, 37
MAXEFFECTS= option, 36
MAXSTEPS= option, 36
METHOD= option, 35
MINEFFECTS= option, 36
ORDERSELECT option, 38
SELECT= option, 36
SELECTION= option, 40
STOP= option, 36
STOPHORIZON= option, 40

CSPATIALREG procedure, TEST statement, 419

DATA= option
MARGINAL statement (ECM), 481
EFFECT statement, spline (SEVSELECT), 27
DELTA= value
ESTIMATE statement (CARIMA), 75
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), 24
EFFECT statement, polynomial (CCDM), 24
EFFECT statement, polynomial (CCOPULA), 24
EFFECT statement, polynomial (CESM), 24
EFFECT statement, polynomial (CNTSELECT), 24
EFFECT statement, polynomial (CPANEL), 24
EFFECT statement, polynomial (CQLIM), 24
EFFECT statement, polynomial (CSPATIALREG), 24
EFFECT statement, polynomial (HMM), 24
EFFECT statement, polynomial (SEVSELECT), 24
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
DETAILS= option
SELECTION statement (CARIMA), 37, 38
SELECTION statement (CCDM), 37, 38
SELECTION statement (CCOPULA), 37, 38
SELECTION statement (CESM), 37, 38
SELECTION statement (CNTSELECT), 37, 38
SELECTION statement (CPANEL), 37, 38
SELECTION statement (CQLIM), 37, 38
SELECTION statement (CSPATIALREG), 37, 38
SELECTION statement (HMM), 37, 38
SELECTION statement (SEVSELECT), 37, 38
DFMIXTURE= option
SCALEMODEL statement (SEVSELECT), 674
DIAG function, 597
DIFF= value-list
ESTIMATE statement (CARIMA), 75
DISCRETE option
ENDOGENOUS statement (CQLIM), 376, 379
DISPERSION= option
OUTPUT statement (CNTSELECT), 255
DISPLAY statement
CCDM procedure, 18, 116
CCOPULA procedure, 18, 173
CNTSELECT procedure, 18, 248
CPANEL procedure, 18, 298
CQLIM procedure, 18, 374
CSPATIALREG procedure, 18, 414
ECM procedure, 18, 479
HMM procedure, 18, 563
SEVSELECT procedure, 18, 659
DISPLAYOUT statement
CCDM procedure, 19, 117
CCOPULA procedure, 19, 174
CNTSELECT procedure, 19, 249
CPANEL procedure, 19, 299
CQLIM procedure, 19, 375
CSPATIALREG procedure, 19, 415
ECM procedure, 19, 480
HMM procedure, 19, 564
SEVSELECT procedure, 19, 660
DISPMODEL statement
CNTSELECT procedure, 249
DIST statement
SEVSELECT procedure, 661
DIST= option
CNTSELECT statement (CNTSELECT), 252
MODEL statement (CNTSELECT), 252
DISTRIBUTION= option
ENDOGENOUS statement (CQLIM), 376, 380
DPF Class Object
TSA package, 851
DRAWID= option
MARGINAL statement (ECM), 481
PROC ECM statement, 475
DYNDIFF option
MODEL statement (CPANEL), 305
DYNSYS option
MODEL statement (CPANEL), 305

ECM procedure, 472
MARGINAL statement, 481
OUTPUT statement, 482
OUTSUM statement, 483
syntax, 472
VAR statement, 485

ECM procedure, DISPLAY statement
CASESENSITIVE option, 19, 480
EXCLUDE option, 19, 480
EXCLUDEALL option, 19, 480
TRACE option, 19, 480

ECM procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 480
NOREPLACE option, 19, 480
REPEATED option, 20, 480

ECM procedure, MARGINAL statement
DATA= option, 481
DRAWID= option, 481
PCTLDEF= option, 482
SAMPLEVAR= option, 481

ECM procedure, OUTPUT statement
MARGINALSOUT option, 482
OUT= option, 482
TOTALLOSSVAR= option, 482

ECM procedure, OUTSUM statement
OUT= option, 483
PCTLNDEC= option, 483
PCTLPRE= option, 483
PCTLPTS= option, 483
TVARPRE= option, 484
TVARPTS= option, 484

ECM procedure, PROC ECM statement, 474
COPULASIM= option, 474
DATA= option, 474
DRAWID= option, 475
EDFAccuracy= option, 475
EXACTFINALCOUNT option, 475
MAXITER= option, 476
NOPRINT option, 476
NOSHUFFLE option, 476
PCTLDEF= option, 477
PRINT= option, 477
SAMPLEFRACTION= option, 478
SEED= option, 478
TAILEDACCURACY= option, 478
TAILSTARTEDF= option, 478
TOLERANCE= option, 478
VARDEF= option, 479
EDFAccuracy= option
PROC ECM statement, 475
EDFAlpha= option
PROC SEVSELECT statement, 651

PROC SEVSELECT statement, 651

EFFECT statement
CARIMA procedure, 20
CCDM procedure, 20
CCOPULA procedure, 20
CESM procedure, 20
CNTSELECT procedure, 20, 250
collection effect, 22
collection effect, 22
polynomial effect, 24
SEVSELECT procedure, 20, 663

END= option
ID statement (CARIMA), 78
ID statement (CESM), 208
ID statement (TSMODEL), 815

ENDogenous statement
CPANEL procedure, 300

ENDSUBMIT statement
TSMODEL procedure, 821

ERRORCOMP= option
CNTSELECT statement (CNTSELECT), 253
MODEL statement (CNTSELECT), 253

ERRSTD
OUTPUT statement (CQLIM), 382

ESMSPEC object
TSM Package, 891

ESTIMATE statement
CARIMA procedure, 74
HMM procedure, 565

ESTISPv option
MODEL statement (HMM), 568

EVALUATE statement, 565

EXACTFINALCOUNT option
PROC ECM statement, 475

EXCLUDE option
DISPLAY statement (CCDM), 19, 116
DISPLAY statement (CCOPULA), 19, 174
DISPLAY statement (CNTSELECT), 19, 248
DISPLAY statement (CPANEL), 19, 299
DISPLAY statement (CQLIM), 19, 375
DISPLAY statement (CSPATIALREG), 19, 414
DISPLAY statement (ECM), 19, 480
DISPLAY statement (HMM), 19, 564
DISPLAY statement (SEVSELECT), 19, 660

EXCLUDEALL option
DISPLAY statement (CCDM), 19, 116
DISPLAY statement (CCOPULA), 19, 174
DISPLAY statement (CNTSELECT), 19, 248
DISPLAY statement (CPANEL), 19, 299
DISPLAY statement (CQLIM), 19, 375
DISPLAY statement (CSPATIALREG), 19, 414
DISPLAY statement (ECM), 19, 480
DISPLAY statement (HMM), 19, 564
DISPLAY statement (SEVSELECT), 19, 660

EXMSPEC object
TSM Package, 904

EXPECTED
OUTPUT statement (CQLIM), 382
EXTERNALCOUNTS statement
CCDM procedure, 117

FCONV 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

FCONV2 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

FDONE option
MODEL statement (CPANEL), 305

FEASTOL= option
OPTIMIZE statement (HMM), 570

FILTER statement, 566
FIT statement
CCOPULA procedure, 175

FIXONE option
MODEL statement (CPANEL), 305

FIXTWO option
MODEL statement (CPANEL), 305

FORECAST statement, 566
CARIMA procedure, 76
CESM procedure, 204

FORMAT= option
ID statement (CARIMA), 78
ID statement (CESM), 208
ID statement (TSMODEL), 815

FREQ class object
TSA package, 855
FREQ statement
CNTSELECT procedure, 251

FRONTIER option
ENDOGENOUS statement (CQLIM), 377, 380

FSIZE 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

FTOL 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

FTOL2 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

FUNCTIONS= option
OUTPUT statement (SEVSELECT), 668

GCONV 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

GCONV2 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
GDELTA= option
OUTPUT statement (CNTSELECT), 255
GetForecast method, 952
TSM Package, 889
GROUPID= option
PROC CNTSELECT statement, 245
GTOL 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
GTOL2 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
HIERARCHY= 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
HMM procedure, 560
syntax, 560
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, 19, 564
EXCLUDE option, 19, 564
EXCLUDEALL option, 19, 564
TRACE option, 19, 564
HMM procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 565
NOREPLACE option, 19, 565
REPEATED option, 20, 565
HMM procedure, EFFECT statement
BASIS option (spline), 27
DATABASENDARY option (spline), 27
DEGREE option (polynomial), 24
DEGREE option (spline), 27
DETAILS option (multimember), 23
DETAILS option (polynomial), 24
DETAILS option (spline), 27
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), 29
SPLIT option (spline), 30
STANDARDIZE option (polynomial), 25
HMM procedure, PROC HMM statement
ABSCONV option, 41
ABSFCONV option, 41
ABSFSTOL option, 41
ABSFGCOV option, 41
ABSGTOL option, 41
ABSTOL option, 41
ABSSXCONV option, 41
ABSXTOL option, 41
DATA= option, 562
FCONV option, 42
FCONV2 option, 42
FSIZE option, 42
FTOL option, 42
FTOL2 option, 42
GCONV option, 43
GCONV2 option, 43
GTOL option, 43
GTOL2 option, 43
MAXFUNC= option, 43
MAXITER= option, 43
MAXTIME= option, 44
MINITER= option, 44
TECHNIQUE= option, 44
XCONV option, 44
XSIZE option, 45
XTOL option, 44
HMM procedure, SELECTION statement
   CHOOSE= option, 36
   COMPETITIVE option, 36
   CRITERION= option, 36
   DETAILS= option, 37, 38
   HIERARCHY= option, 37
   MAXEFFECTS= option, 36
   MAXSTEPS= option, 36
   METHOD= option, 35
   MINEFFECTS= option, 36
   ORDERSELECT option, 38
   SELECT= option, 36
   SELECTION= option, 40
   STOP= option, 36
   STOPHORIZON= option, 40
HTAYLOR option
   MODEL statement (CPANEL), 305
I function, 597
IACF function
   TSA package, 857
ID statement
   CARIMA procedure, 77
   CESM procedure, 206
   CPANEL procedure, 301
   HMM procedure, 567
   TSMODEL procedure, 811
ID= option
   EXTERNALCOUNTS statement (CCDM), 118
IDENTIFY statement
   CARIMA procedure, 80
IDMSPEC object
   TSM Package, 908
IGNORECOV option
   PROC CCDM statement, 112
IN= option
   ESTIMATE statement (HMM), 565
   LEARN statement (HMM), 565
INCLUDE option
   SCALMODEL statement (SEVSELECT), 675
INCLUDEALL option
   DISPLAYOUT statement (CCDM), 19, 117
   DISPLAYOUT statement (CCOPULA), 19, 175
   DISPLAYOUT statement (CNTSELECT), 19, 249
   DISPLAYOUT statement (CPANEL), 19, 300
   DISPLAYOUT statement (CQLIM), 19, 376
   DISPLAYOUT statement (CSPATIALREG), 19, 415
   DISPLAYOUT statement (ECM), 19, 480
   DISPLAYOUT statement (HMM), 19, 565
   DISPLAYOUT statement (SEVSELECT), 19, 660
INEST= option
   PROC SEVSELECT statement, 651
INFORMATIVE option
   SCALMODEL statement (SEVSELECT), 675
INFUNCDEF= option
   DIST statement (SEVSELECT), 662
INIT statement
   CNTSELECT procedure, 252
   CQLIM procedure, 379
   CSPATIALREG procedure, 415
INIT= option
   DIST statement (SEVSELECT), 662
INITIAL statement
   HMM procedure, 567
   Initialize method
   TSM Package, 889
INITSAMPLE option
   PROC SEVSELECT statement, 657
INMODEL= option
   SCORE statement (HMM), 574
INOBJ= option
   PROC TSMODEL statement, 807
INSCALEAR= option
   PROC TSMODEL statement, 807
INSCALARS statement
   TSMODEL procedure, 817
INSTRUMENTS statement
   CPANEL procedure, 302
INTERMITTENCYTEST function
   TSA package, 858
INTERVAL= option
   ID statement (CARIMA), 77
   ID statement (CESM), 207
   ID statement (TSMODEL), 811
ITPRINT option
   CQLIM procedure, 372
   PROC CCOPULA statement, 177
IVBTWNG option
   MODEL statement (CPANEL), 305
IVFIXONE option
   MODEL statement (CPANEL), 305
IVPOOLED option
   MODEL statement (CPANEL), 305
IVRANONE option
   MODEL statement (CPANEL), 305
J function, 598
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

LAMBDA= option
OUTPUT statement (CNTSELECT), 255
LEAD= number
FORECAST statement (CARIMA), 77
LEAD= option
FORECAST statement (CESM), 205
FORECAST statement (HMM), 566
PROC TSMODEL statement, 808
LEARN statement
HMM procedure, 565
LEFTCENSORED= option
LOSS statement (SEVSELECT), 665
LEFTTRUNCATED= option
LOSS statement (SEVSELECT), 666
LISTONLY option
DIST statement (SEVSELECT), 662
LM option
TEST statement (CNTSELECT), 259
TEST statement (CPANEL), 310
TEST statement (CQLIM), 383
TEST statement (CSPATIALREG), 420
LOGCONTROL= option
PROC TSMODEL statement, 808
LOSS statement
SEVSELECT procedure, 665
LOWERBOUND= option
ENDOGENOUS statement (CQLIM), 376, 377, 380
LR option
TEST statement (CNTSELECT), 259
TEST statement (CPANEL), 310
TEST statement (CQLIM), 383
TEST statement (CSPATIALREG), 420
MARGINAL
OUTPUT statement (CQLIM), 382
MARGINAL statement
ECM procedure, 481
MARGINALSOUT option
OUTPUT statement (ECM), 482
MAXCOUNTDRAW= option
PROC CCDM statement, 112
MAXEFFECTS= option
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIM), 36
SELECTION statement (CSPATIALREG), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36
MAXFUN= 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
MAXITER= option
OPTIMIZE statement (HMM), 570
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 ECM statement, 476
PROC HMM statement, 43
PROC SEVSELECT statement, 43
MAXITER= value
ESTIMATE statement (CARIMA), 75
MAXSTEPS= option
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIM), 36
SELECTION statement (CSPATIALREG), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36
MAXTIME= option
OPTIMIZE statement (HMM), 570
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
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), 205
METHOD= option
ESTIMATE statement (CARIMA), 75
FORECAST statement (CESM), 205
MODEL statement (HMM), 568
PROC CNTSELECT statement, 246
PROC CSPATIALREG statement, 412
SELECTION statement (CARIMA), 35
SELECTION statement (CCDM), 35
SELECTION statement (CCOPULA), 35
SELECTION statement (CESM), 35
SELECTION statement (CNTSELECT), 35, 257
SELECTION statement (CPANEL), 35
SELECTION statement (CQLIM), 35
SELECTION statement (CSPATIALREG), 35
SELECTION statement (HMM), 35
SELECTION statement (SEVSELECT), 35, 676
MILLS
OUTPUT statement (CQLIM), 382
MINEFFECTS= option
SELECTION statement (CARIMA), 36
SELECTION statement (CCDM), 36
SELECTION statement (CCOPULA), 36
SELECTION statement (CESM), 36
SELECTION statement (CNTSELECT), 36
SELECTION statement (CPANEL), 36
SELECTION statement (CQLIM), 36
SELECTION statement (CSPATIALREG), 36
SELECTION statement (HMM), 36
SELECTION statement (SEVSELECT), 36
MINITER= 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
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
MODE= option
OUTPUT statement (CNTSELECT), 255
MODEL statement
CNTSELECT procedure, 252
CPANEL procedure, 303
CQLIM procedure, 379
CSPATIALREG procedure, 416
HMM procedure, 568
MOVINGSUMMARY function
TSA package, 859
MSRANGE= option
OPTIMIZE statement (HMM), 571
MSTAT option
COMPARE statement (CPANEL), 296
MU= option
OUTPUT statement (CNTSELECT), 255
MU= value
ESTIMATE statement (CARIMA), 75
MULTISTART= option
OPTIMIZE statement (HMM), 571

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), 569

NEWRAP option
METHOD= option, PROC CSPATIALREG statement, 412
nfor method
TSM Package, 889
NLOPTIONS statement
SEVSELECT procedure, 667

NMSIMP option
METHOD= option, PROC CSPATIALREG statement, 412

NOCORR option
PROC CCOPULA statement, 177
PROC CNTSELECT statement, 245, 253
PROC CPANEL statement, 295
PROC CSPATIALREG statement, 410, 416
PROC ECM statement, 476
PROC SEVSELECT statement, 651

NOREPLACE option
DISPLAYOUT statement (CCDM), 19, 117
DISPLAYOUT statement (CCOPULA), 19, 175
DISPLAYOUT statement (CNTSELECT), 19, 249
DISPLAYOUT statement (CPANEL), 19, 300
DISPLAYOUT statement (CQLIM), 19, 376
DISPLAYOUT statement (CSPATIALREG), 19, 415
DISPLAYOUT statement (ECM), 19, 480
DISPLAYOUT statement (HMM), 19, 565
DISPLAYOUT statement (SEVSELECT), 19, 661

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, 476
NPERTURBESAMPLES= option
PROC CCDM statement, 112
NREPLICATES= option
PROC CCDM statement, 113
nrows method, 955, 959
NRRIDG option
  METHOD= option, PROC CSPATIALREG statement, 412
NSEASON= option
  MODEL statement (HMM), 569
NSIM= option
  FORECAST statement (HMM), 566
NSTATE= option
  MODEL statement (HMM), 569
NU= option
  OUTPUT statement (CNTSELECT), 255
OBJECTIVE= option
  PROC SEVSELECT statement, 658
OBJLIMIT= option
  OPTIMIZE statement (HMM), 571
OFFSET= option
  MODEL statement (CNTSELECT), 253
SCALEMODEL statement (SEVSELECT), 675
ONLINE option
  FORECAST statement (HMM), 567
Open method
  TSM Package, 893, 901, 910, 919
OPTIMIZE statement
  HMM procedure, 570
OPTTOL= option
  OPTIMIZE statement (HMM), 571
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), 379
ORDERSELECT option
  SELECTION statement (CARIMA), 38
  SELECTION statement (CCDM), 38
  SELECTION statement (CCOPULA), 38
  SELECTION statement (CESM), 38
  SELECTION statement (CNTSELECT), 38
  SELECTION statement (CPANEL), 38
  SELECTION statement (CQLIM), 38
  SELECTION statement (CSPATIALREG), 38
  SELECTION statement (HMM), 38
  SELECTION statement (SEVSELECT), 38
OUT= option
  DECODE statement (HMM), 563
  ESTIMATE statement (HMM), 565
  EVALUATE statement (HMM), 566
  FILTER statement (HMM), 566
  FORECAST statement (HMM), 567
  LEARN statement (HMM), 565
  OUTPUT statement (CCDM), 113, 118
  OUTPUT statement (CNTSELECT), 254
  OUTPUT statement (CPANEL), 308
  OUTPUT statement (CQLIM), 381
  OUTPUT statement (CSPATIALREG), 417
  OUTPUT statement (ECM), 482
  OUTPUT statement (SEVSELECT), 668
  OUTSUM statement (CCDM), 120
  OUTSUM statement (ECM), 483
  PROC CARIMA statement, 74
  PROC CESM statement, 203
  PROC TSMODEL statement, 808
  SMOOTH statement (HMM), 574
OUTARRAY option
  TSMODEL procedure, 818
OUTARRAY= option
  PROC TSMODEL statement, 809
OUTARRAYS statement
  TSMODEL procedure, 818
OUTBYID= option
  PROC CARIMA statement, 74
  PROC CESM statement, 203
OUTFUNCDEF= option
  DIST statement (SEVSELECT), 662
OUTLIB= option
  OUTSCORELIB statement (SEVSELECT), 671
OUTLOG= option
  TSMODEL procedure, 818
OUTMODEL= option
  SCORE statement (HMM), 574
OUTMODELINFO= option
  PROC SEVSELECT statement, 651
OUTNVP object, 952
OUTOBJ= option
  PROC TSMODEL statement, 809
OUTPARM= option
  COMPARE statement (CPANEL), 297
OUTPUT statement
  CCDM procedure, 118
  CNTSELECT procedure, 254
  CPANEL procedure, 307
  CQLIM procedure, 381
  CSPATIALREG procedure, 417
  ECM procedure, 482
SEVSELECT procedure, 668
OUTSCALAR statement
TSMODEL procedure, 818
OUTSCALAR= option
PROC TSMODEL statement, 809
OUTSCALARS statement
TSMODEL procedure, 818
OUTSCORELIB statement
SEVSELECT procedure, 671
OUTSTAT= option
COMPARE statement (CPANEL), 297
PROC CESM statement, 204
PROC HMM statement, 562
PROC SEVSELECT statement, 652
OUTSTORE= option
PROC SEVSELECT statement, 652
OUTSUM statement
CCDM procedure, 119
ECM procedure, 483
OUTSUM= option
PROC CARIMA statement, 74
PROC CESM statement, 204
PROC TSMODEL statement, 810
P= option
OUTPUT statement (CPANEL), 308
P= value
ESTIMATE statement (CARIMA), 76
PACF function
TSA package, 862
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), 253
MODEL statement (CNTSELECT), 253
PCTLDEF= option
MARGINAL statement (ECM), 482
PROC CCDM statement, 113
PROC ECM statement, 476
PCTLNAME= option
OUTSUM statement (CCDM), 121
PCTLNDEC= option
OUTSUM statement (CCDM), 122
OUTSUM statement (ECM), 483
PCTLPRE= option
OUTSUM statement (ECM), 483
PCTLPCTS= option
OUTSUM statement (CCDM), 121
OUTSUM statement (ECM), 483
PERTURBOUT option
OUTPUT statement (CCDM), 119
PLOTS= option
PROC SEVSELECT statement, 652
POOLED option
MODEL statement (CPANEL), 305
PRED= option
OUTPUT statement (CPANEL), 308
PREDETERMINED statement
CPANEL procedure, 308
PREDICTED
OUTPUT statement (CQLIM), 382
PREDICTED= option
OUTPUT statement (CPANEL), 308
PRINT statement
TSMODEL procedure, 818
PRINT= option
PROC CCDM statement, 113
PROC ECM statement, 477
PROC SEVSELECT statement, 653
PRINTALL option
MODEL statement, 254, 417
PROC CCOPULA statement, 177
PROC CNTSELECT statement, 245
PROC CPANEL statement, 295
PROC CSPATIALREG statement, 411
PRINTFIXED option
MODEL statement (CPANEL), 307
PRINTINTERNALNAMES option
PROC CNTSELECT statement, 245
PROC CSPATIALREG statement, 411
PRINTITERFREQ= option
OPTIMIZE statement (HMM), 571
PRINTLEVEL= option
OPTIMIZE statement (HMM), 571
PRINTTIMING option
PROC CCOPULA statement, 172
PROC CNTSELECT statement, 245
PROC CSPATIALREG statement, 411
PRIOR statement
HMM procedure, 572
PROB
OUTPUT statement (CQLIM), 382
PROB= option
OUTPUT statement (CNTSELECT), 254
PROBALL
OUTPUT statement (CQLIM), 382
PROBCOUNT option
OUTPUT statement (CNTSELECT), 254
PROBOBSERVED= option
  LOSS statement (SEVSELECT), 666
PROBZERO= option
  OUTPUT statement (CNTSELECT), 255
PROC CARIMA statement, 73
PROC CCDSM statement, 111, see CCDM procedure
PROC CCOPULA statement
  CCOPULA procedure, 172
PROC CESM statement, 203
PROC CPANEL statement, see CPANEL procedure
PROC ECM statement, 474, see ECM procedure
PROC HMM statement, 562
PROC SEVSELECT statement, 649
PROC TSMODEL statement, 806
PRODUCTION option
  ENDOGENOUS statement (CQLIM), 377, 381
  programming statements
    TSMODEL procedure, 821
PSTAT option
  COMPARE statement (CPANEL), 298
PUTTOLOG= option
  PROC TSMODEL statement, 810
Q= value
  ESTIMATE statement (CARIMA), 76
QUANEW option
  METHOD= option, PROC CSPATIALREG statement, 412
QUANTILES= option
  OUTPUT statement (SEVSELECT), 670
R= option
  OUTPUT statement (CPANEL), 308
RANONE option
  MODEL statement (CPANEL), 306
RANTWO option
  MODEL statement (CPANEL), 306
REF= option
  CLASS statement (CARIMA), 16
  CLASS statement (CCDSM), 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
REPEATED option
  DISPLAYOUT statement (CCDSM), 20, 117
  DISPLAYOUT statement (CCOPULA), 20, 175
  DISPLAYOUT statement (CNTSELECT), 20, 249
  DISPLAYOUT statement (CPANEL), 20, 300
  DISPLAYOUT statement (CQLIM), 20, 376
  DISPLAYOUT statement (CSpatialREG), 20, 415
  DISPLAYOUT statement (ECM), 20, 480
  DISPLAYOUT statement (HMM), 20, 565
  DISPLAYOUT statement (SEVSELECT), 20, 661
Replay method
  TSM Package, 890
REQUIRE statement
  TSMODEL procedure, 819
RESID= option
  OUTPUT statement (CSpatialREG), 417
RESIDUAL
  OUTPUT statement (CQLIM), 382
RESIDUAL= option
  OUTPUT statement (CPANEL), 308
RESTRICT statement
  CNTSELECT procedure, 255
  CPANEL procedure, 309
  CQLIM procedure, 382
  CSpatialREG procedure, 418
RIGHTCENSORED= option
  LOSS statement (SEVSELECT), 666
RIGHTTRUNCATED= option
  LOSS statement (SEVSELECT), 667
Run method
  TSM Package, 890
SAMPLEFRACTION= option
  PROC ECM statement, 478
SAMPLEVAR= option
  MARGINAL statement (ECM), 481
  OUTPUT statement (CCDSM), 119
SCALE function
  TSA package, 864
SCALEMODEL statement
  SEVSELECT procedure, 673
SCENTER option
  MODEL statement (HMM), 569
SCORE statement
  HMM procedure, 574
SEASONALDECOMP function
  TSA package, 865
SEASONALINDICES function
  TSA package, 867
SEASONALITY= option
  PROC CESM statement, 204
  PROC TSMODEL statement, 810
SEASONTEST function
  TSA package, 869
SECTION= option
  ID statement (HMM), 567
SEED= option
PROC CCDM statement, 114
PROC ECM statement, 478
PROC HMM statement, 563

SELECT= option
PROC SELECTION statement (CARIMA), 36
PROC SELECTION statement (CCDM), 36
PROC SELECTION statement (CCOPULA), 36
PROC SELECTION statement (CESM), 36
PROC SELECTION statement (CNTSELECT), 36, 258
PROC SELECTION statement (CPANEL), 36
PROC SELECTION statement (CQLIM), 36
PROC SELECTION statement (CSPATIALREG), 36
PROC SELECTION statement (HMM), 36
PROC SELECTION statement (SEVSELECT), 36, 676

SELECTNLOTECH= option
PROC SEVSELECT statement, 658

SELECTOUT option
PROC SEVSELECT statement, 654

SEPARATE option
PROC SEVSELECT statement, 654

SEVERITYEST= option
PROC SEVSELECT procedure, 645

SEVERITYMODEL statement
PROC SEVSELECT procedure, 645

SEVERITYSTORE= option
PROC SEVSELECT procedure, 645

DIST statement, 661
LOSS statement, 665
NLOPTIONS statement, 667
OUTPUT statement, 668
OUTSCORELIB statement, 671
SCALEMODEL statement, 673
syntax, 645

WEIGHT statement, 677

SEVSELECT procedure, CLASS statement, 659
DESCENDING option, 15
MISSING option, 15
ORDER= option, 15
PARAM= option, 15
REF= option, 16
SPLIT option, 17

SEVSELECT procedure, DISPLAY statement
CASESENSITIVE option, 19, 660
EXCLUDE option, 19, 660
EXCLUDEALL option, 19, 660
TRACE option, 19, 660

SEVSELECT procedure, DISPLAYOUT statement
INCLUDEALL option, 19, 660
NOREPLACE option, 19, 661
REPEATED option, 20, 661

SEVSELECT procedure, DIST statement
INFUNCDEF= option, 662
INIT= option, 662
LISTONLY option, 662
OUTFUNCDEF= option, 662
VALIDATEONLY option, 663

SetDiff method
TSM Package, 901

SetMethod method
SEVSELECT procedure, EFFECT statement
  BASIS option (spline), 27
  DATABOUNDARY option (spline), 27
  DEGREE option (polynomial), 24
  DEGREE option (spline), 27
  DETAILS option (multimember), 23
  DETAILS option (polynomial), 24
  DETAILS option (spline), 27
  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), 29
  SPLIT option (spline), 30
  STANDARDIZE option (polynomial), 25

SEVSELECT procedure, LOSS statement
  LEFTCENSORED= option, 665
  LEFTTRUNCATED= option, 666
  PROBOBSERVED= option, 666
  RIGHTCENSORED= option, 666
  RIGHTTRUNCATED= option, 667

SEVSELECT procedure, OUTPUT statement
  COPYVARS= option, 668
  FUNCTIONS= option, 668
  OUT= option, 668
  QUANTILES= option, 670

SEVSELECT procedure, OUTSCORELIB statement
  COMMONPACKAGE option, 672
  OUTBYID= option, 672
  OUTLIB= option, 671

SEVSELECT procedure, PROC SEVSELECT statement, 649
  ABSCONV option, 41
  ABSFCNV option, 41
  ABSFTOL option, 41
  ABSGCONV option, 41
  ABSGTOL option, 41
  ABSTOL option, 41
  ABSXCONV option, 41
  ABSXTOL option, 41
  COVOUT option, 649
  CRITERION= option, 650
  DATA= option, 650
  EDF= AUTO option, 655
  EDF= KAPLANMEIER option, 655
  EDF= MODIFIEDKM option, 655
  EDF= NOTURNBULL option, 656
  EDF= STANDARD option, 656
  EDF= TURNBULL option, 656
  EMPIRICALCDF= option, 655
  FCONV option, 42
  FCONV2 option, 42
  FSIZE option, 42
  FTOL option, 42
  FTOL2 option, 42
  GCONV option, 43
  GCONV2 option, 43
  GTOL option, 43
  GTOL2 option, 43
  INIT= option, 651
  INITSAMPLE option, 657
  MAXFUNCTION option, 43
  MAXITER= option, 43
  MAXTIME= option, 44
  MINTER= option, 44
  NCLPRINT option, 651
  NOPRINT option, 651
  OBJECTIVE= option, 658
  OUTTEST= option, 651
  OUTFMODEINFO= option, 651
  OUTSTAT= option, 652
  OUTSTORE= option, 652
  PLOTS= option, 652
  PRINT= option, 653
  SELECTNLOTECH= option, 658
  SELECTOUT option, 654
  TECHNIQUE= option, 44
  VARDEF= option, 655
  XCONV option, 44
  XSIZE option, 45
  XTOL option, 44
  ZEROEST option, 655

SEVSELECT procedure, SCALEMODEL statement
  DFMIXTURE= option, 674
  INCLUDE option, 675
  INFORMATIVE option, 675
  OFFSET= option, 675
  START option, 675

SEVSELECT procedure, SELECTION statement, 675
  CHOOSE= option, 36, 676
  COMPETITIVE option, 36
  CRITERION= option, 36
  DETAILS= option, 37, 38
  HIERARCHY= option, 37
  MAXEFFECTS= option, 36
  MAXSTEPS= option, 36
  METHOD= option, 35, 676
  MINEFFECTS= option, 36
  ORDERSELECT option, 38
  SELECT= option, 36, 676
  SELECTION= option, 40
  STOP= option, 36, 677
  STOPHORIZON= option, 40

SHAPE function, 598
SIMILARITY function
TSA package, 870
SIMULATE statement
   CCOPULA procedure, 177
SINGULAR= value
   ESTIMATE statement (CARIMA), 76
SMOOTH statement
   HMM procedure, 574
SOLTYPE= option
   OPTIMIZE statement (HMM), 572
SPATIALEFFECTS statement
   CSPATIALREG procedure, 418
SPATIALID statement
   CSPATIALREG procedure, 419
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), 675
START= option
   ID statement (CARIMA), 79
   ID statement (CESM), 209
   ID statement (TSMODEL), 816
STATIONARITYTEST function
   TSA package, 873
STOP= option
   SELECTION statement (CARIMA), 36
   SELECTION statement (CCDM), 36
   SELECTION statement (CCOPULA), 36
   SELECTION statement (CESM), 36
   SELECTION statement (CNTSELECT), 36, 258
   SELECTION statement (CPANEL), 36
   SELECTION statement (CQLIM), 36
   SELECTION statement (CSPATIALREG), 36
   SELECTION statement (HMM), 36
   SELECTION statement (SEVSELECT), 36, 677
STOPHORIZON= option
   SELECTION statement (CARIMA), 40
   SELECTION statement (CCDM), 40
   SELECTION statement (CCOPULA), 40
   SELECTION statement (CESM), 40
   SELECTION statement (CNTSELECT), 40
   SELECTION statement (CPANEL), 40
   SELECTION statement (CQLIM), 40
   SELECTION statement (CSPATIALREG), 40
   SELECTION statement (HMM), 40
   SELECTION statement (SEVSELECT), 40
STORE= option
   PROC CNTSELECT statement, 245
SUBMIT statement
   TSMODEL procedure, 820
TAILEDACCURACY= option
   PROC ECM statement, 478
TAILSTARTEDF= option
   PROC ECM statement, 478
TE1
   OUTPUT statement (CQLIM), 382
TE2
   OUTPUT statement (CQLIM), 382
TECHNIQUE= 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
TEST statement
   CPANEL procedure, 310
TIME= option
   ID statement (HMM), 567
TOLERANCE= option
   PROC ECM statement, 478
TOTALLOSSV AR= option
   OUTPUT statement (ECM), 482

TRACE option
   DISPLAY statement (CCDM), 19, 116
   DISPLAY statement (CCOPULA), 19, 174
   DISPLAY statement (CNTSELECT), 19, 248
   DISPLAY statement (CPANEL), 19, 299
   DISPLAY statement (CQLIM), 19, 375
   DISPLAY statement (CSPATIALREG), 19, 414
   DISPLAY statement (ECM), 19, 480
   DISPLAY statement (HMM), 19, 564
   DISPLAY statement (SEVSELECT), 19, 660

TRANSFORM function
   TSA package, 874
   TRANSFORM= option
      FORECAST statement (CESM), 206
   TRANSFORM= transform-option
      ESTIMATE statement (CARIMA), 76
   TREND= option
      MODEL statement (HMM), 569
   TRIMID= option
      ID statement (CARIMA), 79
      ID statement (CESM), 209
      ID statement (TSMODEL), 816
   TRUNCATED option
      ENDOGENOUS statement (CQLIM), 377, 380
   TRUNCATEZEROS option
      PROC CCDM statement, 115
   TRUREG option
      METHOD= option, PROC CSPATIALREG statement, 412

TSA package
   ACCUMULATE function, 841
   ACCUMULATE2 function, 844
   ACF function, 846
   ARMAORDERS function, 848
   CCF function, 849
   DPF Class Object, 851
   FREQ class object, 855
   IACF function, 857
   INTERMITTENCYTEST function, 858
   MOVINGSUMMARY function, 859
   PACF function, 862
   SCALE function, 864
   SEASONALDECOMP function, 865
   SEASONALINDICES function, 867
   SEASONTEST function, 869
   SIMILARITY function, 870
   STATIONARITYTEST function, 873
   TRANSFORM function, 874
   UNBIASEDNESS function, 876
   WHITENOISE function, 877

TSM object
   TSM Package, 885

TSM Package
   AddARPoly method, 898
   AddBlockSeason method, 915
   AddComponent method, 916
   AddCycle method, 917
   AddDeplag method, 917
   AddExternal method, 887
   AddInput method, 918
   AddMAPoly method, 899
   AddSeason method, 918
   AddTF method, 899
   AddTFDenPoly method, 900
   AddTFNumPoly method, 901
   AddX method, 888
   ARIMASPEC object, 895
   Autoreg method, 915
   Close method, 893, 901, 909, 919
   Criterion method, 888
   ESMSPEC object, 891
   EXMSPEC object, 904
   GetForecast method, 889
   IDMSPEC object, 908
   Initialize method, 889
   nfor method, 889
   Open method, 893, 901, 910, 919
   Replay method, 890
   Run method, 890
   SetDiff method, 901
   SetMethod method, 910
   SetOption method, 890, 893, 902, 910
   SetParm method, 894, 911
   SetTFTransform method, 903
   SetTransform method, 895, 904, 912, 919
   SetY method, 891
   TSM object, 885
   TSMFOR object, 924
   TSMINEST object, 932
   TSMINSPEC object, 933
   TSMPEST object, 920
   TSMSTAT object, 928
   TSMFOR object
      TSM Package, 924
   TSMINEST object
      TSM Package, 932
   TSMINSPEC object
      TSM Package, 933
   TSMODEL procedure, 804, 806
      syntax, 804, 806
   TSMODEL procedure, PROC TSMODEL statement
      DATA= option, 806
   TSMPEST object
      TSM Package, 920
   TSMSPEC object
Syntax Index

TSM Package, 922
TSMSTAT object
  TSM Package, 928
TVARPRE= option
  OUTSUM statement (ECM), 484
TVARPTS= option
  OUTSUM statement (ECM), 484
TYPE= option
  CSPATIALREG procedure, 411
  MODEL statement (HMM), 569
  MODEL statement, CSPATIALREG procedure, 416
  PROC CSPATIALREG statement, 416
UNBIASEDNESS function
  TSA package, 876
UPPERBOUND= option
  ENDONGENOUS statement (CQLIM), 377, 380
UTLSTAT object, 955
VALIDATEONLY option
  DIST statement (SEVSELECT), 663
VAR statement
  CCOPULA procedure, 178
  ECM procedure, 485
  TSMODEL procedure, 819
VARDEF= option
  PROC CCDM statement, 115
  PROC ECM statement, 479
  PROC SEVSELECT statement, 45
VARIANCE= option
  OUTPUT statement (CNTSELECT), 255
VCOMP= option
  MODEL statement (CPANEL), 306
VIEWSTORE statement
  CCOPULA procedure, 178
  CNTSELECT procedure, 260
WALD option
  TEST statement (CNTSELECT), 259
  TEST statement (CPANEL), 310
  TEST statement (CQLIM), 383
  TEST statement (CSPATIALREG), 420
WEIGHT statement
  SEVSELECT procedure, 677
WHITENOISE function
  TSA package, 877
WMAT=option
  PROC CSPATIALREG statement, 409
XBETA
  OUTPUT statement (CQLIM), 382
XBETA= option
  OUTPUT statement (CNTSELECT), 254
  OUTPUT statement (CSPATIALREG), 417
XCONV option
  PROC CARIMA statement, 44
  PROC CCDM statement, 44
  PROC CCOPULA statement, 44
  PROC CESM statement, 44
  PROC CNTSELECT statement, 45
  PROC CPANEL statement, 44
  PROC CQLIM statement, 44
  PROC CSPATIALREG statement, 45
  PROC HMM statement, 44
  PROC SEVSELECT statement, 45
XLAG= option
  MODEL statement (HMM), 570
XSIZE 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, 44
  PROC CSPATIALREG statement, 45
  PROC HMM statement, 44
  PROC SEVSELECT statement, 45
XTOL option
  PROC CARIMA statement, 44
  PROC CCDM statement, 44
  PROC CCOPULA statement, 44
  PROC CESM statement, 44
  PROC CNTSELECT statement, 45
  PROC CPANEL statement, 45
  PROC CQLIM statement, 44
  PROC CSPATIALREG statement, 45
  PROC HMM statement, 44
  PROC SEVSELECT statement, 45
YLAG= option
  MODEL statement (HMM), 570
ZEROEST option
  PROC SEVSELECT statement, 655
ZEROMODEL statement
  CNTSELECT procedure, 261
ZGAMMA= option
  OUTPUT statement (CNTSELECT), 254
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.