SAS® Visual Forecasting 8.5 Forecasting Procedures

SAS® Documentation
August 18, 2020

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

SAS® Visual Forecasting 8.5: Forecasting Procedures

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

August 2020

8.5-P1:casforecast

Contents

Chapter 1
What's New in SAS Visual Forecasting 8.5 Procedures

Contents

Overview

SAS Visual Forecasting 8.5 adds support for the following:

- a new DFIL procedure for digital filter design and filtering of data
- enhancements to the SMSPEC stability monitoring (SM) procedure
- two new packages for the TSMODEL procedure (PROC TSMODEL is also provided in SAS Econometrics 8.5)

New DFIL Procedure

The DFIL procedure designs a digital filter that meets input design specifications. After the filter is designed, PROC DFIL displays the frequency response and zero-pole plot of the designed filter, so that you can evaluate whether the filter has met the design requirements. The DFIL procedure provides the capability to estimate the minimum required order for a digital filter that satisfies the design specifications and then use that order in the design process. The DFIL procedure also enables you to apply the designed digital filter to any input data. To summarize, the DFIL procedure enables you to do the following:

- design a digital infinite impulse response (IIR) filter of any type (lowpass, highpass, bandpass, or bandstop). The supported IIR filters are the Butterworth filter and the Chebyshev types I and II filters.
- visualize the frequency response of the designed filter
- visualize the zero-pole plot of the designed filter for stability analysis
- filter any input data by using the designed digital filter

Enhancements to the SMSPEC Procedure

The SMSPEC procedure generates one or two model specifications to be used in a stability monitoring (SM) project. Stability monitoring implements a collection of anomaly detection methods that can be used to monitor the "health" of a system in order to detect anomalous behavior of various kinds of signals, including signals from sensor and event data. One of the goals of stability monitoring is to predict serious issues, such as catastrophic failures, before they occur in order to manage the cost and frequency of preventive maintenance. The SMSPEC procedure adds the following new feature:

- In the ARIMA statement, the new P= and Q= options enable you to specify a custom range for the moving average (MA) and autoregressive (AR) orders of the ARIMA model specification.

New TSMODEL Procedure Packages

The TSMODEL procedure adds support for the following new packages:

- The external languages (EXTLANG) package enables you to integrate external-language code that is written in Python or R into your SAS program. The package supports various versions of the Python 2, Python 3, and R languages.
- The time series dimension reduction (TDR) package enables you to reduce the dimensionality of time series. The dimension reduction methods include piecewise aggregate approximation, symbolic aggregate approximation, discrete Fourier transformation, discrete wavelet transformation, random projection, and singular value decomposition.

Chapter 2
Introduction

Contents

Overview of SAS Visual Forecasting Procedures

SAS Visual Forecasting procedures include time series analysis and forecasting tools that have been specially developed to take advantage of the distributed environment that SAS Viya provides. Methods include time series analysis (time domain and frequency domain), time series decomposition and filtering, time series modeling, time series forecasting and monitoring, and temporal data mining (motif discovery and time series distance measures). The procedures provide time series diagnostics, automatic variable selection and calendar events, automatic time series model selection for forecasting and monitoring, and out-of-sample performance monitoring.

NOTE: When you license SAS Visual Forecasting, you also have access to SAS/ETS software and SAS Forecast Server Procedures. For more information about SAS/ETS procedures, see *SAS/ETS User's Guide*. For more information about SAS Forecast Server Procedures, see *SAS Forecast Server Procedures: User's Guide*.

About This Book

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

Chapter Organization

This book contains the following chapters:

- Chapter 2, this chapter, provides an overview of SAS Econometrics procedures and summarizes related information, products, and services.
- Chapter 3 describes how to use SAS Cloud Analytic Services (CAS) sessions and how to load a SAS data set onto a CAS server.
- The remaining chapters 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.

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 bold`	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

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

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

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

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

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

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

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

Where to Turn for More Information

Online Documentation

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

SAS Technical Support Services

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

Chapter 3
Shared Concepts

Contents

Introduction to Shared Concepts

SAS Visual Forecasting 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.

Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

```
cas mysess terminate;
```

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

Loading a SAS Data Set onto a CAS Server

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

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

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

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

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

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

- You can use the CASUTIL procedure as follows:

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

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

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

```
proc tsmodel 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*.

Chapter 4
The DFIL Procedure

Contents

Overview: DFIL Procedure

Digital filtering is an important part of digital signal processing (DSP). Signals usually originate as sensory data from the real world: seismic vibrations, visual images, sound waves, and so on. Digital signal processing uses various algorithms and techniques to manipulate these signals in digital form. DSP applications include image enhancement, speech recognition, data compression, and others. In digital signal processing, the function of a digital filter is to remove unwanted parts of an input signal (such as random noise) or to extract useful parts of the signal (such as the components within a certain frequency range). A digital filter uses a digital processor (for example, a general-purpose computer or a digital signal processing microprocessor) to perform numerical calculations on digital signals in order to reduce or enhance certain aspects of that signal.

The DFIL procedure designs a digital filter that meets the input design specifications. After the filter is designed, the procedure displays the frequency response and zero-pole plot of the designed filter, so that you can evaluate whether the filter has met the design requirements. PROC DFIL can internally estimate the minimum required order for a digital filter that satisfies the design specifications and use it as an input to design a digital filter. The DFIL procedure also enables you to use the designed digital filter to filter any input data.

For more information about digital filters and filter design specifications, see the sections "Introduction to Digital Filtering" on page 30 and "Digital Filter Design" on page 34.

It is important to note that digital filtering requires strict maintenance of the sequential order of the input data samples; otherwise, incorrect filtering results will be generated. For more information about how to ensure the sequential order of the input data, see the methods that are used in "Example 4.3: Digital Filtering of Signals Using Different Methods" on page 47.

PROC DFIL Features

The DFIL procedure enables you to do the following:

- design a digital infinite impulse response (IIR) filter of any type (lowpass, highpass, bandpass, or bandstop). The supported IIR filters are the Butterworth filter and the Chebyshev types I and II filters.
- visualize the frequency response of the designed filter
- visualize the zero-pole plot of the designed filter for stability analysis
- filter any input data by using the designed digital filter

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 7 in Chapter 3, "Shared Concepts."

Getting Started: DFIL Procedure

This example shows how to use the DFIL procedure to design a lowpass Butterworth filter and how to visualize the designed filter's characteristics by viewing the frequency response curve and the zero-pole plot. In this example, you can input the specification of the digital filter to be designed by specifying the filter type (lowpass filter), the filter design type (Butterworth), and the filter's normalized upper cutoff frequency (0.5) and filter order (10). For more information about a digital filter's specification, see the section "Digital Filter Design" on page 34.

The following statements run PROC DFIL and output the results to ODS tables and graphs:

```
ods graphics on;
ods html;
proc dfil;
   lowpass upper=0.5 order=10 / butter;
run;
```

Figure 4.1 shows the input specification of the digital filter. The designed filter's transfer function coefficients and zeros, poles, and gain are included in the ODS tables shown in Figure 4.2 and Figure 4.3. If the input filter order is n, then the designed filter order is also n for lowpass and highpass filters or $2n$ for bandpass and bandstop filters.

Figure 4.1 Filter Specification

The DFIL Procedure

Filter Definition	
Filter ID	1
Type	Lowpass
Method	Butterworth
Order	10
Frequency Upper Limit	0.500000

Figure 4.2 Filter Transfer Function Coefficients

Filter Coefficients	
Numerator	**Denominator**
0.0029	1
0.029	0
0.1303	1.3404
0.3476	0
0.6083	0.5454
0.7299	0
0.6083	0.077
0.3476	0
0.1303	0.0032
0.029	0
0.0029	168E-7

Figure 4.3 Filter Zeros, Poles, and Gain

Zeroes, Poles, and Gain				
	Zeroes		Poles	
Gain	Real	Imaginary	Real	Imaginary
0.002896	-1	0	0	0.8541
	-1	0	0	-0.854
	-1	0	0	0.6128
	-1	0	0	-0.613
	-1	0	0	0.4142
	-1	0	0	-0.414
	-1	0	0	0.2401
	-1	0	0	-0.24
	-1	0	0	0.0787
	-1	0	0	-0.079

In addition to the tables shown in Figure 4.1 through Figure 4.3, PROC DFIL also generates the filter's frequency response curve (shown in Figure 4.4) and the zero-pole plot (shown in Figure 4.5). The frequency response curve enables you to inspect whether the input requirements have been met. The zero-pole plot enables you to visualize all zeros and poles and to analyze the filter's stability by inspecting the locations of the poles. A digital filter is stable if and only if all the poles of the irreducible filter transfer function lie inside the unit circle. As shown in Figure 4.5, all poles of the designed filter are inside the unit circle, so the designed filter is stable.

Figure 4.4 Filter Frequency Response Curve

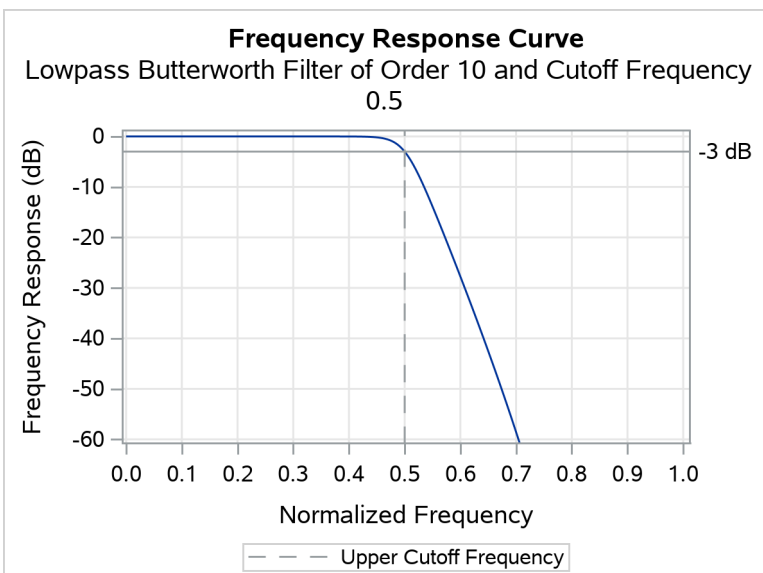
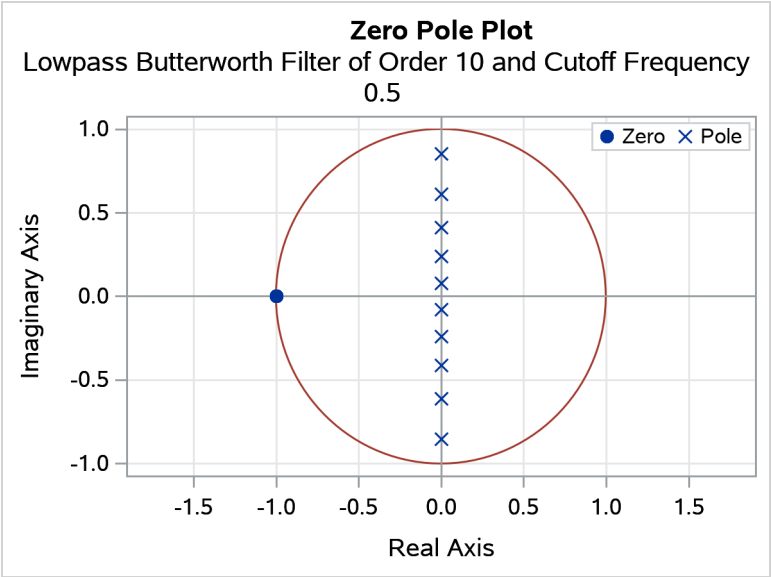


Figure 4.5 Filter Zero Pole Plot



Digital filtering requires strict maintenance of the sequential order of the input data samples; otherwise, incorrect filtering results will be generated. For more information about how to ensure the sequential order of the input data, see the methods that are used in "Example 4.3: Digital Filtering of Signals Using Different Methods" on page 47.

Syntax: DFIL Procedure

The following statements are available in the DFIL procedure (these statements are ordered by their most common usage):

PROC DFIL < *options* > **;**
  LOWPASS UPPER=*number* **ORDER=***number* < *options* > < / *ftype* > **;**
  LOWPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**
  HIGHPASS LOWER=*number* **ORDER=***number* < *options* > < / *ftype* > **;**
  HIGHPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**
  BANDPASS LOWER=*number* **UPPER=***number* **ORDER=***number* < *options* > < / *ftype* > **;**
  BANDPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**
  BANDSTOP LOWER=*number* **UPPER=***number* **ORDER=***number* < *options* > < / *ftype* > **;**
  BANDSTOP ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**

The PROC DFIL statement and at least one of the other statements are required.

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

PROC DFIL Statement

PROC DFIL < *options* > **;**

The PROC DFIL statement invokes the procedure. You can specify the following *option*:

NOPRINT
suppresses the generation of ODS outputs. If you specify this option, no ODS outputs are generated.

BANDPASS Statement

BANDPASS LOWER=*number* **UPPER=***number* **ORDER=***number* < *options* > < / *ftype* > **;**

BANDPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**

The BANDPASS statement specifies the parameters for designing a bandpass digital filter. The BANDPASS statement runs in one of two modes: ORDER or ESTIMATE. These modes use different syntax, and they cannot run simultaneously. You can use an alias of BPASS instead of BANDPASS for either syntax.

The options you specify for the BANDPASS statement differ as follows according to the mode you want to use:

- In the ORDER mode, you need to specify the order and the lower and upper cutoff frequencies of the bandpass filter in order for the DFIL procedure to design the filter. In the ORDER mode, you must specify the following options:

  ORDER=*number*
  specifies the order of the digital filter, where *number* is a positive integer scalar. The maximum supported filter order is 200.

  LOWER=*number*
  specifies the lower cutoff frequency ω_{c1} of the bandpass filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

  UPPER=*number*
  specifies the upper cutoff frequency ω_{c2} of the bandpass filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

- In the ESTIMATE mode, you need to specify the passband edge frequencies, stopband edge frequencies, and the passband ripple and stopband attenuation of the bandpass filter. The DFIL procedure estimates the minimum required order to meet these filter design specifications, calculates the lower and upper cutoff frequencies of the bandpass filter, and uses all three values in the filter design process. In the ESTIMATE mode, you must specify the following option:

  ESTIMATE=(PASS1=*number* **PASS2=***number* **STOP1=***number* **STOP2=***number* **RP=***number* **RS=***number***)**

  specifies the parameters to use to estimate the minimum required order to meet the filter design specifications and to calculate the lower and upper cutoff frequencies of the filter.

  You must specify the following suboptions in the order shown. The PASS1=, PASS2=, STOP1=, and STOP2= suboptions must be specified such that their values (ω_{p1}, ω_{p2}, ω_{s1} and ω_{s2},

respectively) satisfy $\omega_{s1} < \omega_{p1} < \omega_{p2} < \omega_{s2}$. The RP= and RS= options must be specified such that the absolute value of the passband ripple, RP, is less than the absolute value of the stopband attenuation, RS (that is, |RP| < |RS|).

PASS1=*number*
specifies the lower passband edge frequency of the bandpass filter ω_{p1}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

PASS2=*number*
specifies the upper passband edge frequency of the bandpass filter ω_{p2}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

STOP1=*number*
specifies the lower stopband edge frequency of the bandpass filter ω_{s1}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

STOP2=*number*
specifies the upper stopband edge frequency of the bandpass filter ω_{s2}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

RP=*number*
specifies the passband ripple, which is expressed in decibels (dB), where *number* is a positive scalar.

RS=*number*
specifies the stopband attenuation, which is expressed in decibels (dB), where *number* is a positive scalar.

In either ORDER mode or ESTIMATE mode, you can specify the following optional *options*:

CODE=(FILE=*filename* **IVAR=***(input-variable-list)* **SVAR=***(output-variable-list)***)**
generates SAS DATA step code that mimics the computations that are done by using the designed digital filter to filter any input data. You must specify the following suboptions:

FILE=*filename*
specifies the filename of the file to write the SAS DATA step code to, where *filename* can be a quoted string that contains the path of the file in which to store the code or a fileref that refers to an external file in which to save code.

IVAR=*(input-variable-list)*
specifies a list of input variables, each of which contains a vector of data to be filtered by the designed digital filter. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the input variables. When there is a single input variable, the parentheses are not required. The number of variables that are specified in the IVAR= and SVAR= options must be equal.

SVAR=*(output-variable-list)*
specifies a list of output variables, each of which contains a filtered output data vector. The output variable list should not contain variables that have identical names. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the output variables. When there is a single output variable, the parentheses are not required. The number of variables that are specified in the SVAR= and IVAR= options must be equal.

SAVESTATE=(RSTORE=*CAS-libref.data-table* **IVAR=***input-variable* **SVAR=***output-variable***)**
creates an analytic store for the designed digital filter and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to filter any input data. You must specify the following suboptions:

RSTORE=*CAS-libref.data-table*
specifies the name of the table in which to save the analytic store of the designed filter. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 13.

IVAR=*input-variable*
specifies the input variable, which contains a vector of data to be filtered by the designed digital filter. Only one input variable is supported.

SVAR=*output-variable*
specifies the output variable, which contains a filtered output data vector. Only one output variable is supported.

DBT=*number*
specifies the magnitude threshold, which is expressed in decibels (dB), of the frequency response to display in the frequency response plot, where *number* is a positive scalar. Any frequency range whose absolute magnitude is greater than or equal to *number* is not displayed.

You can also specify one of the following *ftype* values after a slash (/) to request the type of IIR filter to be designed:

BUTTER
designs a Butterworth bandpass filter.

CHEB1(RP=*number***)**
designs a Chebyshev type I bandpass filter. The filter's passband ripple, which is expressed in decibels (dB), is specified in the required RP= suboption, where *number* is a positive scalar.

CHEB2(RS=*number***)**
designs a Chebyshev type II bandpass filter. The filter's stopband attenuation, which is expressed in decibels (dB), is specified in the required RS= suboption, where *number* is a positive scalar.

The default *ftype* is BUTTER.

BANDSTOP Statement

BANDSTOP LOWER=*number* **UPPER=***number* **ORDER=***number* < *options* > < / *ftype* > **;**

BANDSTOP ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**

The BANDSTOP statement specifies the parameters for designing a bandstop digital filter. The BANDSTOP statement runs in one of two modes: ORDER or ESTIMATE. These modes use different syntax, and they cannot run simultaneously. You can use an alias of BSTOP instead of BANDSTOP for either syntax.

The options you specify for the BANDSTOP statement differ as follows according to the mode you want to use:

- In the ORDER mode, you need to specify the order and the lower and upper cutoff frequencies of the bandstop filter in order for the DFIL procedure to design the filter. In the ORDER mode, you must specify the following options:

  ORDER=*number*
  specifies the order of the digital filter, where *number* is a positive integer scalar. The maximum supported filter order is 200.

  LOWER=*number*
  specifies the lower cutoff frequency ω_{c1} of the bandstop filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

  UPPER=*number*
  specifies the upper cutoff frequency ω_{c2} of the bandstop filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

- In the ESTIMATE mode, you need to specify the passband edge frequencies, stopband edge frequencies, and the passband ripple and stopband attenuation of the bandstop filter. The DFIL procedure estimates the minimum required order to meet these filter design specifications, calculates the lower and upper cutoff frequencies of the bandstop filter, and uses all three values in the filter design process. In the ESTIMATE mode, you must specify the following option:

  ESTIMATE=(PASS1=*number* **PASS2=***number* **STOP1=***number* **STOP2=***number* **RP=***number* **RS=***number***)**

  specifies the parameters to use to estimate the minimum required order to meet the filter design specifications and to calculate the lower and upper cutoff frequencies of the filter.

  You must specify the following suboptions in the order shown. The PASS1=, PASS2=, STOP1=, and STOP2= suboptions must be specified such that their values (ω_{p1}, ω_{p2}, ω_{s1} and ω_{s2}, respectively) satisfy $\omega_{p1} < \omega_{s1} < \omega_{s2} < \omega_{p2}$. The RP= and RS= options must be specified such that the absolute value of the passband ripple, RP, is less than the absolute value of the stopband attenuation, RS (that is, |RP| < |RS|).

PASS1=*number*
specifies the lower passband edge frequency of the bandstop filter ω_{p1}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

PASS2=*number*
specifies the upper passband edge frequency of the bandstop filter ω_{p2}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

STOP1=*number*
specifies the lower stopband edge frequency of the bandstop filter ω_{s1}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

STOP2=*number*
specifies the upper stopband edge frequency of the bandstop filter ω_{s2}, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

RP=*number*
specifies the passband ripple, which is expressed in decibels (dB), where *number* is a positive scalar.

RS=*number*
specifies the stopband attenuation, which is expressed in decibels (dB), where *number* is a positive scalar.

In either ORDER mode or ESTIMATE mode, you can specify the following optional *options*:

CODE=(FILE=*filename* **IVAR=***(input-variable-list)* **SVAR=***(output-variable-list)***)**
generates SAS DATA step code that mimics the computations that are done by using the designed digital filter to filter any input data. You must specify the following suboptions:

FILE=*filename*
specifies the filename of the file to write the SAS DATA step code to, where *filename* can be a quoted string that contains the path of the file in which to store the code or a fileref that refers to an external file in which to save code.

IVAR=*(input-variable-list)*
specifies a list of input variables, each of which contains a vector of data to be filtered by the designed digital filter. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the input variables. When there is a single input variable, the parentheses are not required. The number of variables that are specified in the IVAR= and SVAR= options must be equal.

SVAR=*(output-variable-list)*
specifies a list of output variables, each of which contains a filtered output data vector. The output variable list should not contain variables that have identical names. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the output variables. When there is a single output variable, the parentheses are not required. The number of variables that are specified in the SVAR= and IVAR= options must be equal.

SAVESTATE=(RSTORE=*CAS-libref.data-table* **IVAR=***input-variable* **SVAR=***output-variable***)**
creates an analytic store for the designed digital filter and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to filter any input data. You must specify the following suboptions:

RSTORE=*CAS-libref.data-table*
specifies the name of the table in which to save the analytic store of the designed filter. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 13.

IVAR=*input-variable*
specifies the input variable, which contains a vector of data to be filtered by the designed digital filter. Only one input variable is supported.

SVAR=*output-variable*
specifies the output variable, which contains a filtered output data vector. Only one output variable is supported.

DBT=*number*
specifies the magnitude threshold, which is expressed in decibels (dB), of the frequency response to display in the frequency response plot, where *number* is a positive scalar. Any frequency range whose absolute magnitude is greater than or equal to *number* is not displayed.

You can also specify one of the following *ftype* values after a slash (/) to request the type of IIR filter to be designed:

BUTTER
designs a Butterworth bandstop filter.

CHEB1(RP=*number***)**
designs a Chebyshev type I bandstop filter. The filter's passband ripple, which is expressed in decibels (dB), is specified in the required RP= suboption, where *number* is a positive scalar.

CHEB2(RS=*number***)**
designs a Chebyshev type II bandstop filter. The filter's stopband attenuation, which is expressed in decibels (dB), is specified in the required RS= suboption, where *number* is a positive scalar.

The default *ftype* is BUTTER.

HIGHPASS Statement

HIGHPASS LOWER=*number* **ORDER=***number* < *options* > < / *ftype* > **;**

HIGHPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**

The HIGHPASS statement specifies the parameters for designing a highpass digital filter. The HIGHPASS statement runs in one of two modes: ORDER or ESTIMATE. These modes use different syntax, and they cannot run simultaneously. You can use an alias of HPASS instead of HIGHPASS for either syntax.

The options you specify for the HIGHPASS statement differ as follows according to the mode you want to use:

- In the ORDER mode, you need to specify the order and the cutoff frequency of the highpass filter in order for the DFIL procedure to design the filter. In the ORDER mode, you must specify the following options:

  ORDER=*number*
  specifies the order of the digital filter, where *number* is a positive integer scalar. The maximum supported filter order is 200.

  LOWER=*number*
  specifies the cutoff frequency ω_c of the highpass filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

- In the ESTIMATE mode, you need to specify the passband edge frequency, stopband edge frequency, and the passband ripple and stopband attenuation of the highpass filter. The DFIL procedure estimates the minimum required order to meet these filter design specifications, calculates the cutoff frequency of the highpass filter, and uses these two values in the filter design process. In the ESTIMATE mode, you must specify the following option:

  ESTIMATE=(PASS1=*number* **STOP1=***number* **RP=***number* **RS=***number***)**
  specifies the parameters to use to estimate the minimum required order to meet the filter design specifications and to calculate the cutoff frequency of the filter.

  You must specify the following suboptions in the order shown. The PASS1= and STOP1= suboptions must be specified such that their values (ω_p and ω_s, respectively) satisfy $\omega_s < \omega_p$. The RP= and RS= options must be specified such that the absolute value of the passband ripple, RP, is less than the absolute value of the stopband attenuation, RS (that is, |RP| < |RS|).

  PASS1=*number*
  specifies the passband edge frequency of the highpass filter ω_p, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

  STOP1=*number*
  specifies the stopband edge frequency of the highpass filter ω_s, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

RP=*number*
specifies the passband ripple, which is expressed in decibels (dB), where *number* is a positive scalar.

RS=*number*
specifies the stopband attenuation, which is expressed in decibels (dB), where *number* is a positive scalar.

In either ORDER mode or ESTIMATE mode, you can specify the following optional *options*:

CODE=(FILE=*filename* **IVAR=***(input-variable-list)* **SVAR=***(output-variable-list)***)**
generates SAS DATA step code that mimics the computations that are done by using the designed digital filter to filter any input data. You must specify the following suboptions:

FILE=*filename*
specifies the filename of the file to write the SAS DATA step code to, where *filename* can be a quoted string that contains the path of the file in which to store the code or a fileref that refers to an external file in which to save code.

IVAR=*(input-variable-list)*
specifies a list of input variables, each of which contains a vector of data to be filtered by the designed digital filter. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the input variables. When there is a single input variable, the parentheses are not required. The number of variables that are specified in the IVAR= and SVAR= options must be equal.

SVAR=*(output-variable-list)*
specifies a list of output variables, each of which contains a filtered output data vector. The output variable list should not contain variables that have identical names. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the output variables. When there is a single output variable, the parentheses are not required. The number of variables that are specified in the SVAR= and IVAR= options must be equal.

SAVESTATE=(RSTORE=*CAS-libref.data-table* **IVAR=***input-variable* **SVAR=***output-variable***)**
creates an analytic store for the designed digital filter and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to filter any input data. You must specify the following suboptions:

RSTORE=*CAS-libref.data-table*
specifies the name of the table in which to save the analytic store of the designed filter. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 13.

IVAR=*input-variable*
specifies the input variable, which contains a vector of data to be filtered by the designed digital filter. Only one input variable is supported.

SVAR=*output-variable*
specifies the output variable, which contains a filtered output data vector. Only one output variable is supported.

DBT=*number*
specifies the magnitude threshold, which is expressed in decibels (dB), of the frequency response to display in the frequency response plot, where *number* is a positive scalar. Any frequency range whose absolute magnitude is greater than or equal to *number* is not displayed.

You can also specify one of the following *ftype* values after a slash (/) to request the type of IIR filter to be designed:

BUTTER
designs a Butterworth highpass filter.

CHEB1(RP=*number***)**
designs a Chebyshev type I highpass filter. The filter's passband ripple, which is expressed in decibels (dB), is specified in the required RP= suboption, where *number* is a positive scalar.

CHEB2(RS=*number***)**
designs a Chebyshev type II highpass filter. The filter's stopband attenuation, which is expressed in decibels (dB), is specified in the required RS= suboption, where *number* is a positive scalar.

The default *ftype* is BUTTER.

LOWPASS Statement

LOWPASS UPPER=*number* **ORDER=***number* < *options* > < / *ftype* > **;**

LOWPASS ESTIMATE=(*suboptions***)** < *options* > < / *ftype* > **;**

The LOWPASS statement specifies the parameters for designing a lowpass digital filter. The LOWPASS statement runs in one of two modes: ORDER or ESTIMATE. These modes use different syntax, and they cannot run simultaneously. You can use an alias of LPASS instead of LOWPASS for either syntax.

The options you specify for the LOWPASS statement differ as follows according to the mode you want to use:

- In the ORDER mode, you need to specify the order and the cutoff frequency of the lowpass filter in order for the DFIL procedure to design the filter. In the ORDER mode, you must specify the following options:

  ORDER=*number*
  specifies the order of the digital filter, where *number* is a positive integer scalar. The maximum supported filter order is 200.

UPPER=*number*
specifies the cutoff frequency ω_c of the lowpass filter, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

- In the ESTIMATE mode, you need to specify the passband edge frequency, stopband edge frequency, and the passband ripple and stopband attenuation of the lowpass filter. The DFIL procedure estimates the minimum required order to meet these filter design specifications, calculates the cutoff frequency of the lowpass filter, and uses these two values in the filter design process. In the ESTIMATE mode, you must specify the following option:

  ESTIMATE=(PASS1=*number* **STOP1=***number* **RP=***number* **RS=***number***)**
  specifies the parameters to use to estimate the minimum required order to meet the filter design specifications and to calculate the cutoff frequency of the filter.

  You must specify the following suboptions in the order shown. The PASS1= and STOP1= suboptions must be specified such that their values (ω_p and ω_s, respectively) satisfy $\omega_p < \omega_s$. The RP= and RS= options must be specified such that the absolute value of the passband ripple, RP, is less than the absolute value of the stopband attenuation, RS (that is, $|RP| < |RS|$).

  PASS1=*number*
  specifies the passband edge frequency of the lowpass filter ω_p, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

  STOP1=*number*
  specifies the stopband edge frequency of the lowpass filter ω_s, where *number* is between 0 and 1, with 1 corresponding to the normalized Nyquist frequency (π rad/sample).

  RP=*number*
  specifies the passband ripple, which is expressed in decibels (dB), where *number* is a positive scalar.

  RS=*number*
  specifies the stopband attenuation, which is expressed in decibels (dB), where *number* is a positive scalar.

In either ORDER mode or ESTIMATE mode, you can specify the following optional *options*:

CODE=(FILE=*filename* **IVAR=***(input-variable-list)* **SVAR=***(output-variable-list)***)**
generates SAS DATA step code that mimics the computations that are done by using the designed digital filter to filter any input data. You must specify the following suboptions:

FILE=*filename*
specifies the filename of the file to write the SAS DATA step code to, where *filename* can be a quoted string that contains the path of the file in which to store the code or a fileref that refers to an external file in which to save code.

IVAR=*(input-variable-list)*
specifies a list of input variables, each of which contains a vector of data to be filtered by the designed digital filter. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the input variables. When there is a single input variable, the parentheses are not required. The number of variables that are specified in the IVAR= and SVAR= options must be equal.

SVAR=*(output-variable-list)*
specifies a list of output variables, each of which contains a filtered output data vector. The output variable list should not contain variables that have identical names. Internal variables of the code file always start with _d_, so avoid using the prefix _d_ in naming the output variables. When there is a single output variable, the parentheses are not required. The number of variables that are specified in the IVAR= and SVAR= options must be equal.

SAVESTATE=(RSTORE=*CAS-libref.data-table* **IVAR=***input-variable* **SVAR=***output-variable***)**
creates an analytic store for the designed digital filter and saves it as a binary object in a data table. You can use the analytic store in the ASTORE procedure to filter any input data. You must specify the following suboptions:

RSTORE=*CAS-libref.data-table*
specifies the name of the table in which to save the analytic store of the designed filter. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 13.

IVAR=*input-variable*
specifies the input variable, which contains a vector of data to be filtered by the designed digital filter. Only one input variable is supported.

SVAR=*output-variable*
specifies the output variable, which contains a filtered output data vector. Only one output variable is supported.

DBT=*number*
specifies the magnitude threshold, which is expressed in decibels (dB), of the frequency response to display in the frequency response plot, where *number* is a positive scalar. Any frequency range whose absolute magnitude is greater than or equal to *number* is not displayed.

You can also specify one of the following *ftype* values after a slash (/) to request the type of IIR filter to be designed:

BUTTER
designs a Butterworth lowpass filter.

CHEB1(RP=*number***)**
designs a Chebyshev type I lowpass filter. The filter's passband ripple, which is expressed in decibels (dB), is specified in the required RP= suboption, where *number* is a positive scalar.

CHEB2(RS=*number*)
designs a Chebyshev type II lowpass filter. The filter's stopband attenuation, which is expressed in decibels (dB), is specified in the required RS= suboption, where *number* is a positive scalar.

The default *ftype* is BUTTER.

Details: DFIL Procedure

Information Representation in the Time and Frequency Domains

To understand digital filtering, it is important to understand how a digital signal is represented in the time domain and the frequency domain.

In the time domain, a *sample* of a digital signal is defined as the signal's amplitude at the point in time at which it occurs. Each sample in the signal indicates both what is happening at that instant (the event) and the level of the event. Time-domain representation is the waveform of a signal that is usually seen for real-world data.

In contrast, information that is represented in the frequency domain is more challenging to understand. A signal's frequency-domain representation is related to the concept of using "trigonometric sums" (that is, sums of harmonically related sines and cosines or periodic complex exponentials) to describe periodic phenomena within a signal (Oppenheim, Willsky, and Nawab 1997). The most commonly used frequency-domain representation is the Fourier transform of a signal. The discrete-time Fourier transform of a digital signal $x(n)$ is defined as

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

To understand the physical meaning of the Fourier transform, start with a simple periodic signal, the complex exponential signal $x(n)$,

$$x(n) = e^{j\omega_0 n}$$

where $x(n)$ is periodic with a fundamental period N and $\omega_0 = \frac{2\pi}{N}$ is the fundamental frequency of the signal. The Fourier transform of $x(n)$, by definition, is a periodic impulse train:

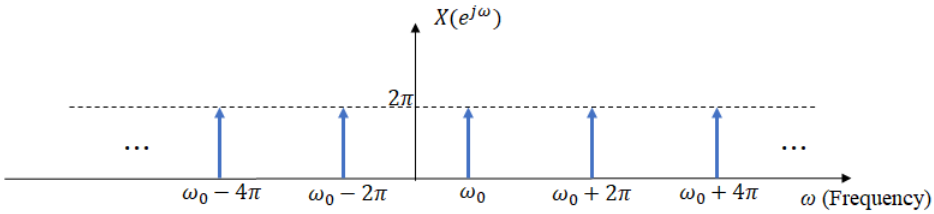
$$\begin{aligned} X(e^{j\omega}) &= \sum_{n=-\infty}^{\infty} e^{j\omega_0 n} e^{-j\omega n} \\ &= \sum_{k=-\infty}^{\infty} 2\pi\delta(\omega - \omega_0 + 2\pi k) \end{aligned}$$

where $\delta(x)$ is a unit impulse function, which is defined as

$$\delta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \neq 0 \end{cases}$$

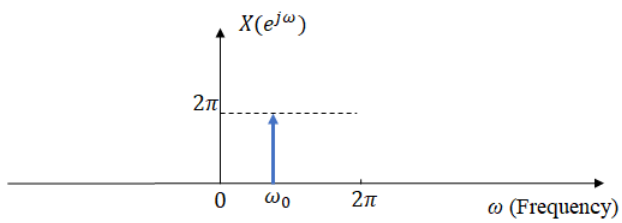
Figure 4.6 shows the Fourier transform of $x(n) = e^{j\omega_0 n}$.

Figure 4.6 Fourier Transform of the Complex Exponential Signal



Because by definition the discrete-time Fourier transform $X(e^{j\omega})$ is periodic in ω with period 2π, $X(e^{j\omega})$ for $\omega \in [0, 2\pi)$ is referred to mostly as the Fourier transform of a signal $x(n)$. The Fourier transform of $x(n) = e^{j\omega_0 n}$ when $\omega \in [0, 2\pi)$ is a single impulse in the frequency domain, as shown in Figure 4.7.

Figure 4.7 Fourier Transform of the Complex Exponential Signal in One Period



It has been shown that any real discrete-time signal $x(n)$ with length N can be represented by a linear combination of complex exponential signals as

$$\begin{aligned} x(n) &= \sum_{k=0}^{N-1} a_k e^{jk\omega_0 n} \\ &= \sum_{k=0}^{N-1} a_k e^{jk(2\pi/N)n} \end{aligned}$$

where $\omega_0 = \frac{2\pi}{N}$ and where a_k $(k = 0, 1, \ldots, N-1)$ are the Fourier series coefficients of the signal $x(n)$, which can be obtained by the following formulas (Oppenheim, Willsky, and Nawab 1997):

$$\begin{aligned} a_k &= \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-jk\omega_0 n} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-jk(2\pi/N)n} \end{aligned}$$

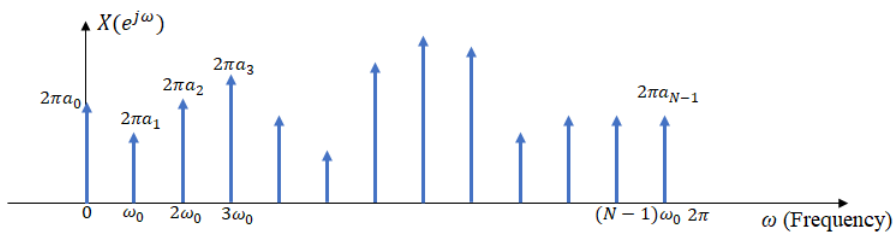
Fourier transforms have an additivity property, which means that addition in the time domain corresponds to addition in the frequency domain. So the Fourier transform of a real signal $x(n)$ is the sum of the Fourier transform of signals $a_k e^{jk(2\pi/N)n}$ $(k = 0, 2, \ldots, N-1)$. The Fourier transform of $x(n)$ is actually a series

of impulses in the frequency domain as follows:

$$X(e^{j\omega}) = \sum_{k=0}^{N-1} 2\pi a_k \delta(\omega - \frac{2\pi k}{N})$$
$$= \sum_{k=0}^{N-1} 2\pi a_k \delta(\omega - k\omega_0)$$

Figure 4.8 is a graphical representation of these equations.

Figure 4.8 Fourier Transform of a Real Signal $x(n)$ with Length N



The Fourier transform, which is a frequency-domain representation of a signal, decomposes the signal into multiple components on harmonic frequencies $(0, \omega_0, 2\omega_0, \ldots)$ similar to how a musical chord can be expressed as the frequencies (pitches) of its constituent notes. This representation makes it easier to extract or remove information by its frequency range.

Introduction to Digital Filtering

A digital filter is defined by its impulse response, $h(n)$, which is the filter's output when the input signal is a unit impulse signal $\delta(n)$. The unit impulse signal $\delta(n)$ is defined as follows:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

Figure 4.9 illustrates the impulse response of a digital filter. The importance of the impulse response is that a linear time-invariant filter is fully characterized by its impulse response. The output of a linear time-invariant digital filter, $y(n)$, is the convolution of its impulse response, $h(n)$, with the input signal, $x(n)$, as shown in Figure 4.10.

Figure 4.9 Impulse Response of a Digital Filter

Figure 4.10 Digital Filter Output as a Convolution of the Input Signal and Impulse Response

It is known that if a signal $y(n)$ is the convolution of two signals $h(n)$ and $x(n)$ in the time domain and if the z transforms of these three signals are $Y(z)$, $H(z)$, and $X(z)$, respectively, then $Y(z) = H(z) \times X(z)$. The transfer function of a digital filter, $H(z)$, is defined as the ratio between the output signal z transform and the input signal z transform:

$$H(z) = \frac{Y(z)}{X(z)}$$

A basic property of the z transform is that if z is evaluated on the unit circle ($z = e^{j\omega}$), the z transform of a signal is equivalent to its Fourier transform. The frequency response of a linear time-invariant digital filter is defined as the filter's transfer function $H(z)$ evaluated on the unit circle, $H(e^{j\omega})$. This definition implies that a filter's frequency response is also the Fourier transform of the impulse response $h(n)$ (Oppenheim and Schafer 2010). A filter's frequency response and the Fourier transforms of the filter's input and output satisfy the following equation:

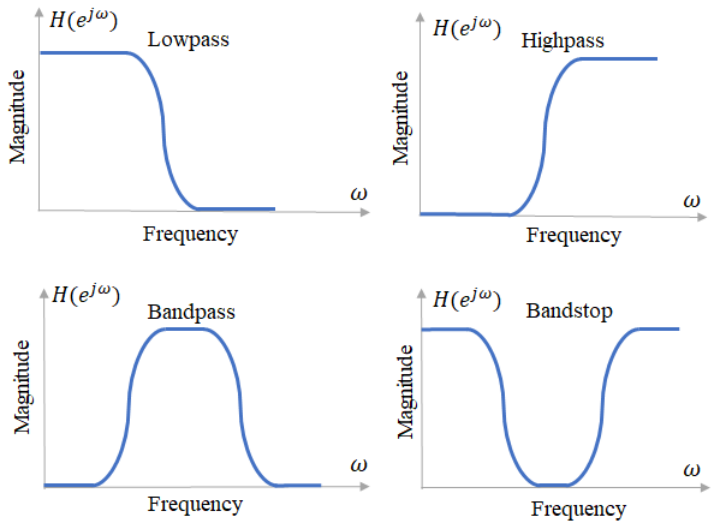
$$H(e^{j\omega}) = \frac{Y(e^{j\omega})}{X(e^{j\omega})}$$

The function of a digital filter can be better explained by its frequency response. The frequency response measures how much the magnitude and phase of the output signal have changed compared to those of the input signal in frequency domain. A digital filter is essentially a system that allows signals of only certain frequencies to pass while blocking all others. Depending on the regime of frequencies that a digital filter allows through (or not), it is characterized as lowpass, highpass, bandpass, or bandstop:

- A *lowpass filter* retains the low-frequency part of an input signal and blocks (attenuates or removes) the high-frequency part of the input.
- A *highpass filter* retains the high-frequency part of an input signal and blocks (attenuates or removes) the low-frequency part of the input.
- A *bandpass filter* allows the middle frequencies of an input signal to pass and blocks (attenuates or removes) other frequencies.
- A *bandstop filter* is the opposite of a bandpass filter. It blocks (attenuates or removes) the middle frequencies and retains other frequencies.

The shape of the magnitude of the frequency response determines which frequencies will pass. Figure 4.11 shows the magnitude of the frequency response for the four different types of filters.

Figure 4.11 Magnitude of the Frequency Response for Digital Filters



The term digital filtering encompasses both the design of a filter and the filtering of an input signal. The purpose of filter design is to obtain a desired filter transfer function $H(z)$. Evaluating $H(z)$ on the unit circle ($z = e^{j\omega}$) results in the frequency response of the filter—that is, how the filter will pass or block certain frequencies in the input signal. The causal linear time-invariant filter's transfer function can be written as

$$H(z) = \frac{Y(z)}{X(z)}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_M z^{-M}}$$

$$H(z) = k\frac{(1 - q_1 z^{-1})(1 - q_2 z^{-1})\ldots(1 - q_N z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})\ldots(1 - p_M z^{-1})}$$

where the order of the filter is the greater of N or M; $b_i (i = 0, 1, 2, \ldots, N)$ are the transfer function's numerator polynomial coefficients; $a_i (i = 0, 1, 2, \ldots, M)$ are the denominator polynomial coefficients; and $q_i (i = 1, 2, \ldots, N)$, $p_i (i = 1, 2, \ldots, M)$, and k are the zeros, poles, and gain of the transfer function, respectively.

In the time domain, the preceding transfer function is equivalent to the following relationship between the filter's input signal $x(n)$ and output $y(n)$:

$$\begin{aligned} y(n) =& \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_N x[n-N] \\ & - a_1 y[n-1] - a_2 y[n-2] + \ldots + a_M y[n-M]) \end{aligned}$$

Based on the transfer function structure, there are two categories of digital filters: the infinite impulse response (IIR) filter and the finite impulse response (FIR) filter. When $a_0 = 1$ and $a_i = 0$ for $i = 1, 2, \ldots, M$, a digital filter is an FIR filter and the output $y(n)$ depends only on the current and previous inputs:

$$y(n) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_N x[n-N]$$

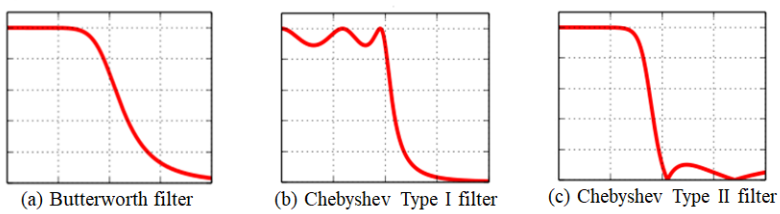
IIR filters, on the other hand, are recursive, with the output depending not only on the current and previous inputs, but also on previous outputs. The general form of an IIR filter is thus:

$$\sum_{m=0}^{M} a_m y[n-m] = \sum_{k=0}^{N} b_k x[n-k]$$

A digital filter can be designed as an IIR filter or an FIR filter. The advantage of IIR filters over FIR filters is that they typically meet a particular set of specifications by using a much lower filter order than a corresponding FIR filter. The classical IIR filters include Butterworth, Chebyshev types I and II, elliptic, and Bessel (Parks and Burrus 1987; Roy 2005; Constantinides 1970). Currently, the DFIL procedure supports the Butterworth, Chebyshev type I, and Chebyshev type II filters.

Figure 4.12 shows examples of lowpass filter frequency response for the Butterworth, Chebyshev type I, and Chebyshev type II filters. The Butterworth filter's frequency response monotonically decreases. It has the flattest passband but a poor roll-off between the passband and the stopband. The Chebyshev filters achieve a faster roll-off but have equiripple in the frequency response. The Chebyshev type I filter has ripples only in the passband, and the Chebyshev type II filter has ripples only in the stopband.

Figure 4.12 Frequency Responses of Different Types of Filters



Which digital filter you select in signal processing depends on the intended application. Butterworth filters are used mostly in applications that require maximum passband flatness. For example, in audio signal processing, Butterworth filters are preferable, because they have no ripples in the passband and cause less distortion in the audio signals. Because Chebyshev filters can achieve a sharper transition between the passband and the stopband with a much lower filter order, they are more attractive in applications where smaller absolute errors or faster execution speed is required.

Once a digital filter is designed, it can be applied to an input signal in order to extract certain frequencies from the signal and remove others. This filtering process can be implemented directly (the direct form) or by using a second-order section form of the filter's transfer function (the SOS form). Any transfer function $H(z)$ can have a second-order section representation as follows:

$$H(z) = \prod_{k=1}^{N} H_k(z)$$
$$H(z) = \prod_{k=1}^{N} \frac{b_{0k} + b_{1k}z^{-1} + b_{2k}z^{-2}}{1 + a_{1k}z^{-1} + a_{2k}z^{-2}}$$

By careful pairing of the poles and zeros and careful ordering of the sections in the cascade, it is possible to reduce quantization noise gain and avoid overflow in the fixed-point filter implementations. So the second-order section form of filtering is more numerically stable than the direct form of filtering, and is thus preferred for real-world digital filtering applications.

Digital Filter Design

Digital filters are designed by specifying the frequency response of the filters in the frequency domain. The specification includes a digital filter's cutoff frequencies, passband and stopband edge frequencies, passband ripple, and stopband attenuation. Figure 4.13 through Figure 4.16 illustrate the filter specification for each type of digital filter (Oppenheim and Schafer 2010; Parks and Burrus 1987).

Figure 4.13 Filter Specification for Lowpass Filter

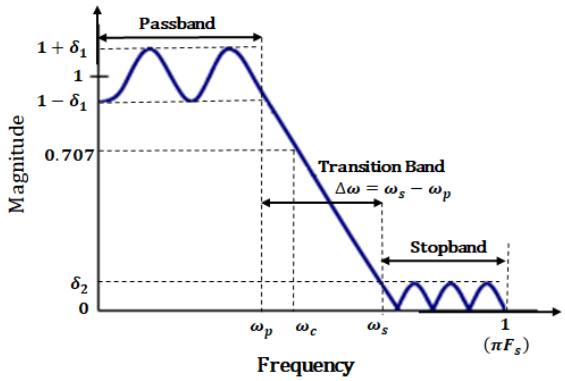


Figure 4.14 Filter Specification for Highpass Filter

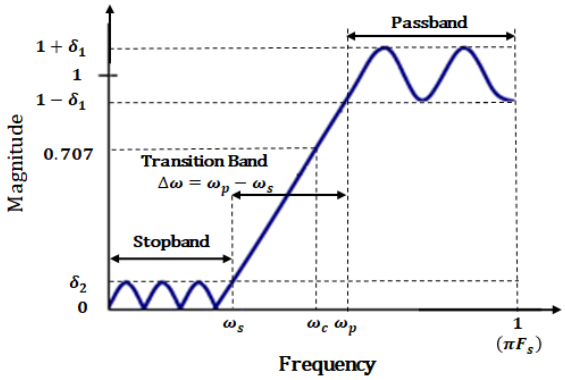


Figure 4.15 Filter Specification for Bandpass Filter

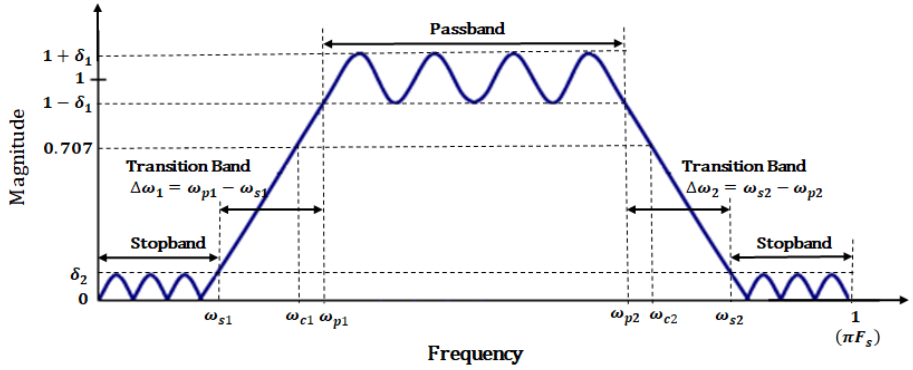
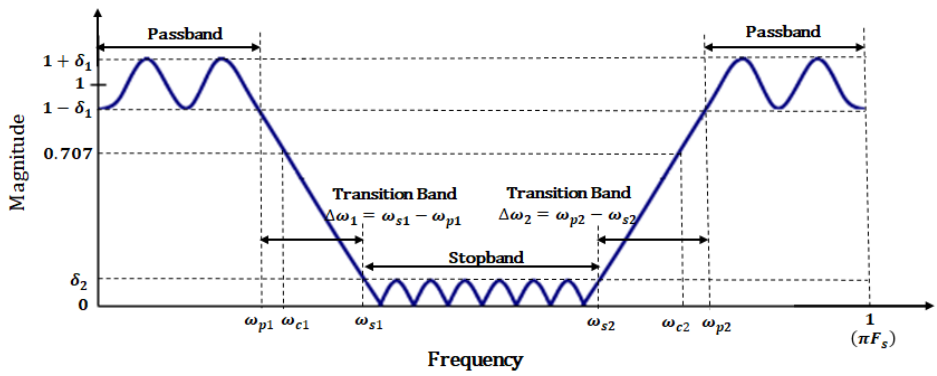


Figure 4.16 Filter Specification for Bandstop Filter



A digital filter's cutoff frequency, ω_c, is defined as the frequency at which the power of the frequency response reaches half the unity power, or equivalently $\sqrt{1/2} \approx 0.707$ of the unity magnitude, which is approximately $-20\log_{10}(0.707) = 3$ in dB. Because half power is about 3dB away from unity power, this frequency is often called the 3dB cutoff frequency. For lowpass and highpass filters, only one cutoff frequency value needs to be specified. However, for bandpass and bandstop filters, two cutoff frequency values need to be specified, as illustrated in Figure 4.15 and Figure 4.16.

The digital filter passband and stopband edge frequencies are ω_p and ω_s, respectively. For lowpass and highpass filters, only one value of each edge frequency is needed. However, for bandpass and bandstop filters, two values of the passband edge frequencies (ω_{p1} and ω_{p2}) and two values of the stopband edge frequencies (ω_{s1} and ω_{s2}) are needed. The frequency range from the passband edge frequency to the stopband edge frequency is the transition band. The transition band has a frequency response that is unspecified (Oppenheim and Schafer 2010; Parks and Burrus 1987; Roy 2005; Constantinides 1970).

The digital filter passband and stopband can contain oscillations known as ripples. The symbol δ_1 represents the magnitude of the passband ripple, which equals the maximum deviation from the unity magnitude. The symbol δ_2 represents the magnitude response of the stopband attenuation, which equals the maximum deviation from zero. The passband ripple (R_p) and stopband attenuation (R_s) are usually measured in decibels (dB) and are defined, respectively, as

$$R_p = -20\log_{10}(1-\delta_1)\ \text{(dB)}$$
$$R_s = -20\log_{10}(\delta_2)\ \text{(dB)}$$

where the absolute value of the passband ripple, R_p, must be less than the absolute value of the stopband attenuation, R_s; that is, $|R_p| < |R_s|$.

The values of the passband and stopband edge frequencies are normalized values between 0 and 1, where 1 corresponds to the normalized Nyquist frequency (π rad/sample). When the passband and stopband edge frequencies are specified, the following expressions must be satisfied:

Lowpass filter: $\omega_p < \omega_s$

Highpass filter: $\omega_s < \omega_p$

Bandpass filter: $\omega_{s1} < \omega_{p1} < \omega_{p2} < \omega_{s2}$

Bandstop filter: $\omega_{p1} < \omega_{s1} < \omega_{s2} < \omega_{p2}$

Digital Filter Stability

A linear time-invariant digital filter is stable if the impulse response, $h(n)$, approaches zero as n goes to infinity. Every FIR filter is stable. For an IIR filter, which is defined as the following, the feedback coefficients $a_i (i = 1, 2, \ldots, M)$ can cause instability:

$$\sum_{m=0}^{M} a_m y[n-m] = \sum_{k=0}^{N} b_k x[n-k]$$

The stability of an IIR filter can be analyzed by the zero-pole plot of the filter's transfer function. A digital filter is stable if and only if all the poles of the irreducible filter transfer function lie inside the unit circle in the z plane. Any pole outside the unit circle introduces an exponentially increasing component in the filter's impulse response, thus causing the filter to be unstable.

If the transfer function has poles on the unit circle, it is called marginally stable. In this case, the impulse response neither decays nor grows as n goes to infinity.

Displayed Output

The DFIL procedure displays various tables that are related to input and results. The following sections describe the output tables in the order of their appearance. When displayed in a table, a numerical value is rounded to the nearest integer if it is within 1E–12 of that integer.

Filter Definition

The "Filter Definition" table displays basic information about the parameters that are used in the DFIL procedure. This information includes the filter ID, the filter type, the filter design method, and the order and cutoff frequencies of the designed filter.

The order that is displayed in the "Filter Definition" table is the designed filter order. If the input filter order is n, then the designed filter order is also n for lowpass and highpass filters or $2n$ for bandpass and bandstop filters.

Zeros, Poles, and Gain

The "Zeros, Poles, and Gain" table displays the zeros, poles, and gain of the designed filter's transfer function. The gain is a scalar. The zeros and poles are vectors of complex numbers with length n, where n is the designed filter order. The zeros and poles are each displayed with two columns: the first column contains the real part of the complex numbers, and the second column contains the imaginary part.

Filter Coefficients

The "Filter Coefficients" table displays the coefficients of the designed filter's transfer function. The first column is the transfer function's numerator polynomial coefficients, and the second column is the transfer function's denominator polynomial coefficients,

ODS Table Names

Each table that the DFIL procedure creates has a name associated with it. You must use this name to refer to the table when you use ODS statements. The names of each table and a short description of the contents are listed in Table 4.1.

Table 4.1 ODS Tables Produced by PROC DFIL

Table Name	Description	Statement	Option
CoeffTable	Filter transfer function coefficients	PROC DFIL	Default
FilterDef	Filter definition parameters	PROC DFIL	Default
ZPGTable	Filter transfer function zeros, poles, and gain	PROC DFIL	Default

ODS Graphics

Statistical procedures use ODS Graphics to create graphs as part of their output. ODS Graphics is described in detail in the "Statistical Graphics Using ODS" chapter in *SAS/STAT User's Guide*.

Before you create graphs, ODS Graphics must be enabled (for example, by using the ODS GRAPHICS ON statement). For more information about enabling and disabling ODS Graphics, see the section "Enabling and Disabling ODS Graphics" in that chapter.

The overall appearance of graphs is controlled by ODS styles. Styles and other aspects of using ODS Graphics are discussed in the section "A Primer on ODS Statistical Graphics" in that chapter.

When ODS Graphics is enabled, the DFIL procedure by default produces the designed filter's frequency response curve and its zero-pole plot.

PROC DFIL assigns a name to each graph that it creates by using ODS. You can use these names to refer to the graphs when using ODS. The names are listed in Table 4.2.

Table 4.2 Graphs Produced by PROC DFIL

ODS Graph Name	Plot Description	PLOTS= Option
FreqResponsePlot	Filter frequency response curve	Default
ZPPlot	Filter zero-pole plot	Default

Examples: DFIL Procedure

Example 4.1: Digital Filter Design with ORDER Mode and a Butterworth Bandpass Filter

This example shows how to use the DFIL procedure to design a Butterworth bandpass filter with the ORDER mode and use it to filter input signals.

The following DATA step generates a data table that contains an input signal x and its timestamps t. The input signal is the sum of three sinusoidal signals, whose frequencies are $f1$ = 50 Hz, $f2$ = 150 Hz, and $f3$ = 300 Hz. The sampling frequency of the signal is 2,000 Hz. The Nyquist frequency of this signal is 1,000 Hz (half of the sampling frequency), and the normalized frequencies of the three sinusoids are $\frac{50}{1000} = 0.05$, $\frac{150}{1000} = 0.15$, and $\frac{300}{1000} = 0.3$.

```
data input1;
   keep t x;
   sf  = 2000;     /* sampling frequency */
   sf2 = sf/2;
   t0  = 1.0/sf;   /* sampling period */
   N   = 300;
   pi  = 3.1415926535;

   f1 = 50;
   f2 = 150;
   f3 = 300;
   do i = 1 to N;
      t  = (i-1)*t0;
      x1 = sin(2*pi*f1*t);
      x2 = sin(2*pi*f2*t);
      x3 = sin(2*pi*f3*t);
      x  = x1 + x2 + x3;     /* input signal */
      output;
   end;
run;
```

You can use the following DFIL procedure statements to design a Butterworth bandpass filter that uses the ORDER mode. The filter's normalized lower cutoff frequency is 0.1, and the normalized upper cutoff frequency is 0.2. The input filter order n is 8.

```
ods graphics on;
ods html;
filename f1 catalog "work.temp.example1.source";
proc dfil;
   bandpass lower=0.1 upper=0.2 order=8 code=(file=f1 ivar=x svar=y)/ butter;
run;
```

Output 4.1.1 shows the input specification of the bandpass Butterworth filter. The designed filter's transfer function coefficients and zeros, poles, and gain are included in the ODS tables in Output 4.1.2 and Output 4.1.3. If the input filter order is n, the designed filter order is also n for lowpass and highpass filters, and is $2n$ for bandpass and bandstop filters. So the designed bandpass filter in this example is 16.

Output 4.1.1 Butterworth Bandpass Filter Specification

The DFIL Procedure

Filter Definition	
Filter ID	1
Type	Bandpass
Method	Butterworth
Order	16
Frequency Lower Limit	0.100000
Frequency Upper Limit	0.200000

Output 4.1.2 Butterworth Bandpass Filter Transfer Function Coefficients

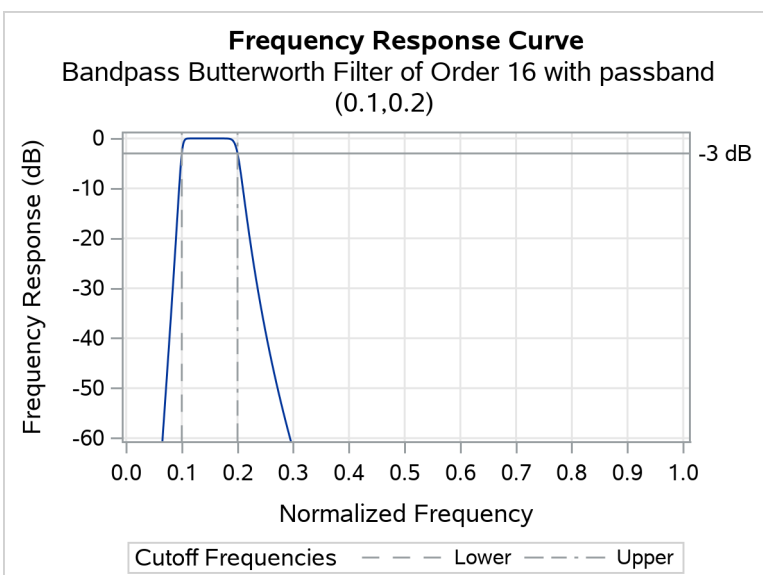
Filter Coefficients	
Numerator	**Denominator**
176E-9	1
0	-12.98
-14E-7	80.23
0	-313.2
494E-8	863.81
0	-1785
-99E-7	2856.6
0	-3612
123E-7	3647.3
0	-2950
-99E-7	1905
0	-972
494E-8	384.2
0	-113.8
-14E-7	23.806
0	-3.147
176E-9	0.1981

Output 4.1.3 Butterworth Bandpass Filter Zeros, Poles, and Gain

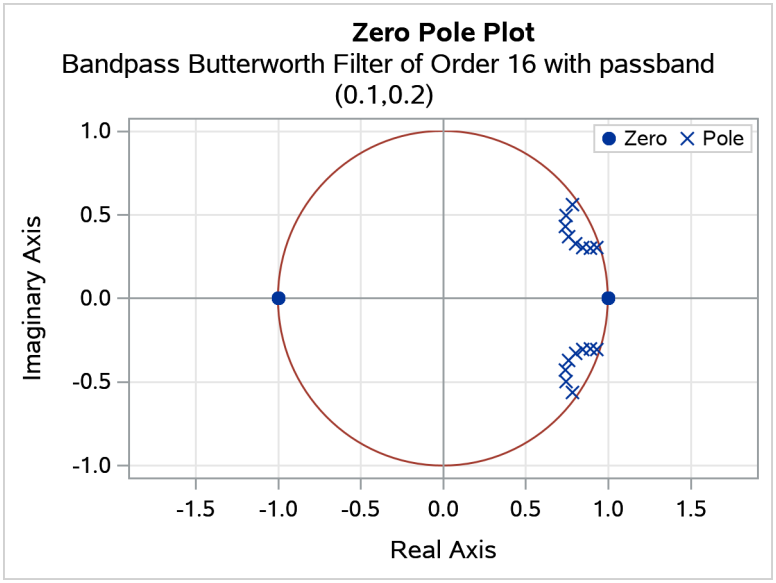
Zeroes, Poles, and Gain				
	Zeroes		Poles	
Gain	Real	Imaginary	Real	Imaginary
0.000000176	1	0	0.931	-0.304
	1	0	0.931	0.3037
	1	0	0.8895	-0.3
	1	0	0.8895	0.2997
	1	0	0.8453	-0.306
	1	0	0.8453	0.3064
	1	0	0.7999	-0.328
	1	0	0.7999	0.3282
	-1	0	0.7803	0.5615
	-1	0	0.7803	-0.562
	-1	0	0.7449	0.4976
	-1	0	0.7449	-0.498
	-1	0	0.7394	0.4296
	-1	0	0.7394	-0.43
	-1	0	0.7606	0.37
	-1	0	0.7606	-0.37

PROC DFIL also generates the filter's frequency response curve, as shown in Output 4.1.4, and the zero-pole plot, as in Output 4.1.5. The frequency response curve enables you to inspect whether the input requirements have been met. The zero-pole plot enables you to visualize all zeros and poles and analyze the filter's stability by inspecting the locations of the poles. A digital filter is stable if and only if all the poles of the irreducible filter transfer function lie inside the unit circle. As shown in Output 4.1.5, all poles of the designed filter are inside the unit circle; therefore, the designed bandpass filter is stable.

Output 4.1.4 Butterworth Bandpass Filter Frequency Response Curve



Output 4.1.5 Butterworth Bandpass Filter Zero Pole Plot



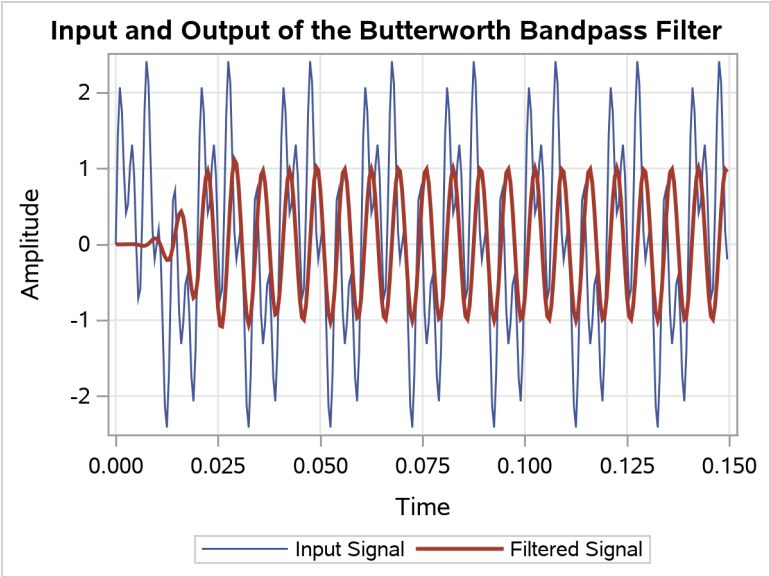
The following DATA step uses the designed bandpass filter to filter the input signal:

```
data output1;
   set input1;
   %inc f1;
   keep t x y;
run;
```

The following code plots the input signal and filtered output signal, which are shown in Output 4.1.6. The bandpass filter filters out any signal components that are outside of its passband ([0.1 0.2]), and the output signal now contains only the component that is of frequency 100 Hz (normalized frequency 0.15).

```
proc sgplot data=output1;
   title "Input and Output of the Butterworth Bandpass Filter";
   series x=t y=x / legendlabel="Input Signal";
   series x=t y=y / legendlabel="Filtered Signal" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.1.6 Butterworth Bandpass Filtering Example



Example 4.2: Digital Filter Design with ESTIMATE Mode and a Chebyshev Type I Highpass Filter

This example shows how to use the DFIL procedure to design a Chebyshev type I highpass filter that uses the ESTIMATE mode and then apply that filter to input signals.

The following DATA step generates a data table that contains two input signals, $x1$ and $x2$, and the timestamps t. The first input signal $x1$ is the sum of two sinusoidal signals, whose frequencies are $f1$ = 250 Hz and $f2$ = 500 Hz. The second input signal $x2$ is also the sum of two sinusoidal signals, whose frequencies are $f1$ = 250 Hz and $f3$ = 700 Hz. The sampling frequency of the input signals is 5,000 Hz. The Nyquist frequency of the signals is 2,500 Hz (half of the sampling frequency), and the normalized frequencies of these sinusoids are $\frac{250}{2500} = 0.1$, $\frac{500}{2500} = 0.2$, and $\frac{700}{2500} = 0.28$.

```
data input2;
   keep t x1 x2;
   sf  = 5000;     /* sampling frequency */
   sf2 = sf/2;
   t0  = 1.0/sf;   /* sampling period */
   N   = 500;
   pi  = 3.1415926535;

   f1 = 250;
   f2 = 500;
   f3 = 700;
   do i = 1 to N;
      t  = (i-1)*t0;
      x11 = 0.5*sin(2*pi*f1*t);
      x12 = sin(2*pi*f2*t);
      x1  = x11 + x12;     /* input signal 1 */
      x22 = sin(2*pi*f3*t);
```

```
      x2  = x11 + x22;     /* input signal 2 */
      output;
   end;
run;
```

You can use the following DFIL procedure statements to design a Chebyshev type I highpass filter that uses the ESTIMATE mode. In this mode, you need to specify the highpass filter's passband edge frequency, stopband edge frequency, passband ripple, and stopband attenuation. The DFIL procedure estimates the minimum required order to meet these filter design specifications, calculates the cutoff frequency of the highpass filter, and uses the order and cutoff frequency in the filter design process. You can specify multiple input variables in the IVAR= suboption of the CODE= option, and you can specify the corresponding output variables in the SVAR= suboption.

```
ods graphics on;
ods html;
filename f2 "example2.sas";
proc dfil;
   highpass estimate(pass1=0.15,stop1=0.12,rp=5,rs=50)
            code=(file=f2 ivar=(x1 x2) svar=(y1 y2))/ cheb1;
run;
```

Output 4.2.1 shows the input specification of the highpass Chebyshev type I filter. The designed filter's transfer function coefficients and zeros, poles, and gain are included in the ODS tables that are shown in Output 4.2.2 and Output 4.2.3. PROC DFIL estimates a filter order of 9 and an estimated normalized cutoff frequency of 0.15.

Output 4.2.1 Chebyshev Type I Highpass Filter Specification

Input and Output of the Butterworth Bandpass Filter

The DFIL Procedure

Filter Definition	
Filter ID	1
Type	Highpass
Method	Chebyshev Type I
Order	9
Frequency Lower Limit	0.150000
Passband Ripple	5.000000

Output 4.2.2 Chebyshev Type I Highpass Filter Transfer Function Coefficients

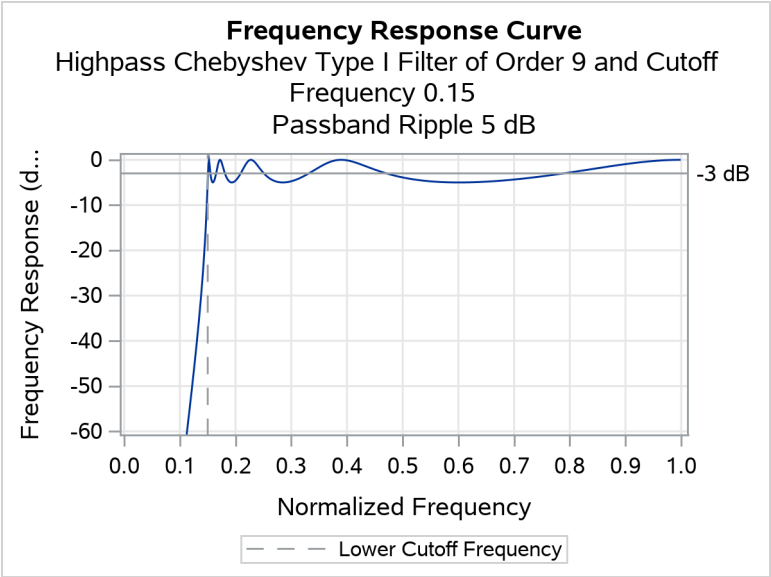
Filter Coefficients	
Numerator	**Denominator**
0.0934	1
-0.841	-4.948
3.3641	11.445
-7.849	-15.61
11.774	13.129
-11.77	-5.879
7.8495	-0.278
-3.364	2.214
0.841	-1.351
-0.093	0.3232

Output 4.2.3 Chebyshev Type I Highpass Filter Zeros, Poles, and Gain

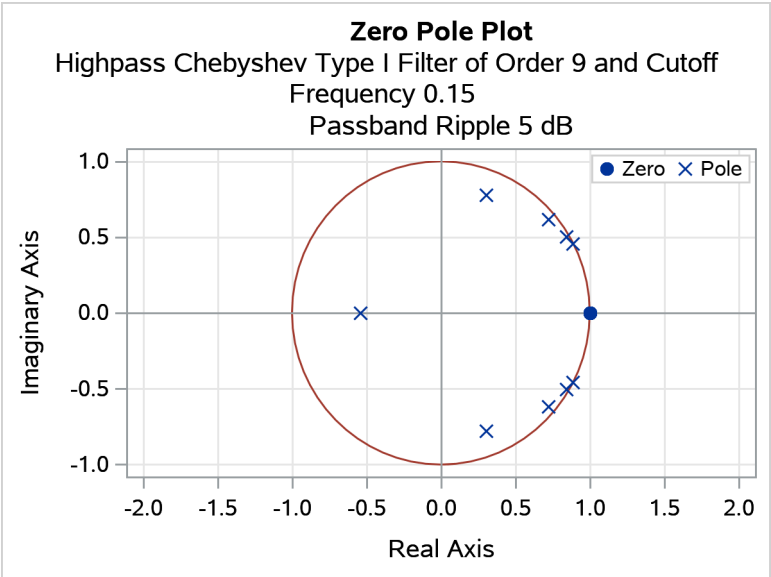
Zeroes, Poles, and Gain				
	Zeroes		Poles	
Gain	**Real**	**Imaginary**	**Real**	**Imaginary**
0.093446	1	0	0.8833	-0.457
	1	0	0.8406	-0.502
	1	0	0.7184	-0.617
	1	0	0.3042	-0.778
	1	0	-0.545	0
	1	0	0.3042	0.7782
	1	0	0.7184	0.6165
	1	0	0.8406	0.5025
	1	0	0.8833	0.4565

PROC DFIL also generates the filter's frequency response curve, as shown in Output 4.2.4, and the zero-pole plot, as shown in Output 4.2.5. The frequency response curve enables you to inspect whether the input requirements have been met. The zero-pole plot enables you to visualize all zeros and poles and analyze the filter's stability by inspecting the locations of the poles. A digital filter is stable if and only if all the poles of the irreducible filter transfer function lie inside the unit circle. As shown in Output 4.2.5, all poles of the designed filter are inside the unit circle; therefore, the designed highpass filter is stable.

Output 4.2.4 Chebyshev Type I Highpass Filter Frequency Response Curve



Output 4.2.5 Chebyshev Type I Highpass Filter Zero Pole Plot



The following DATA step uses the designed highpass filter to filter the input signals:

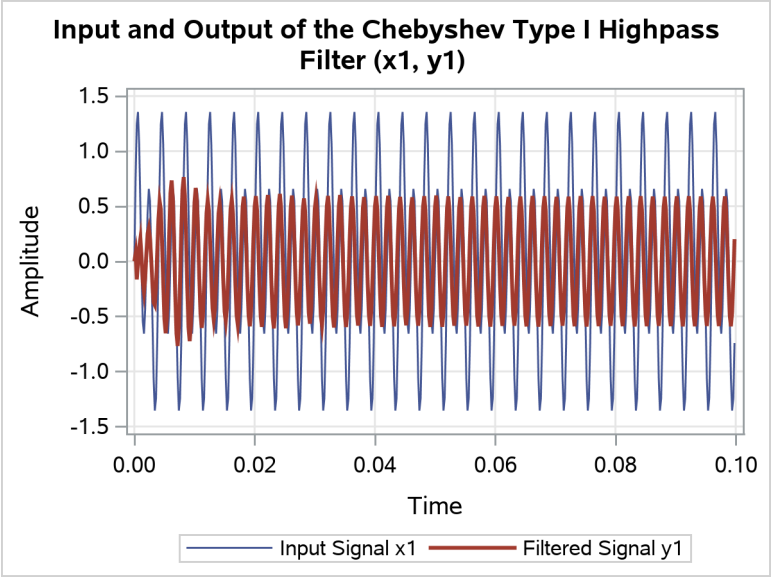
```
data output2;
   set input2;
   %inc f2;
   keep t x1 x2 y1 y2;
run;
```

The following code plots the input signals and filtered output signals, which are shown in Output 4.2.6. The highpass filter filters out any signal components that are outside of its passband. So the first output signal $y1$

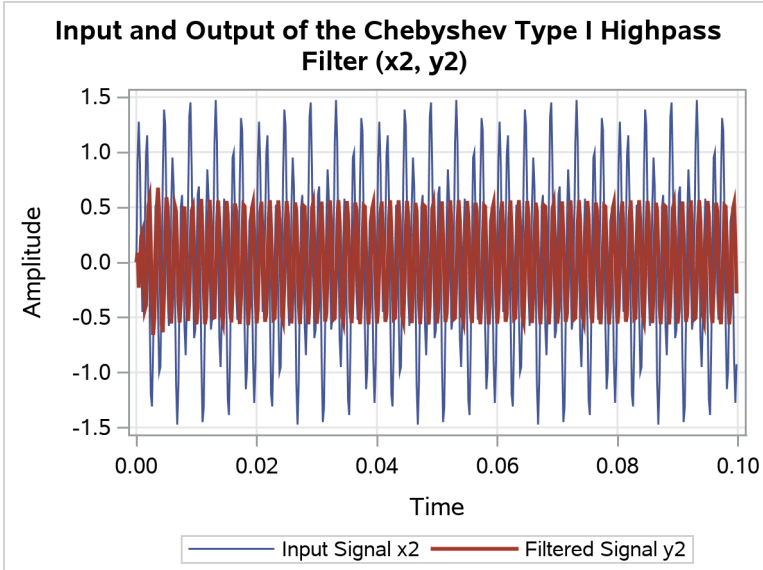
now contains only the component that is of frequency 500 Hz (normalized frequency 0.2), and the second output signal $y2$ now contains only the component that is of frequency 700 Hz (normalized frequency 0.28),

```
proc sgplot data=output2;
   title "Input and Output of the Chebyshev Type I Highpass Filter (x1, y1)";
   series x=t y=x1 / legendlabel="Input Signal x1";
   series x=t y=y1 / legendlabel="Filtered Signal y1" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;

proc sgplot data=output2;
   title "Input and Output of the Chebyshev Type I Highpass Filter (x2, y2)";
   series x=t y=x2 / legendlabel="Input Signal x2";
   series x=t y=y2 / legendlabel="Filtered Signal y2" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.2.6 Chebyshev Type I Highpass Filtering Example



Output 4.2.6 *continued*



Example 4.3: Digital Filtering of Signals Using Different Methods

You can use a digital filter that is designed by PROC DFIL to filter any input signal. Digital filtering requires strict maintenance of the sequential order of the input data samples; otherwise, incorrect filtering results will be generated. This example shows four different methods to filter an input signal by using a Butterworth lowpass filter that is designed by the DFIL procedure. Each method ensures the sequential order of the input data samples and generates correct filtering results.

The following DATA step generates a data table that contains an input signal x and its timestamps t. The input signal is the sum of two sinusoidal signals, whose frequencies are $f1$ = 300 Hz and $f2$ = 600 Hz. The sampling frequency of the signal is 2,000 Hz. The Nyquist frequency of this signal is 1,000 Hz (half of the sampling frequency), and the normalized frequencies of the two sinusoids are $\frac{300}{1000} = 0.3$ and $\frac{150}{1000} = 0.6$.

```
data input;
   keep t x;
   sf  = 2000;     /* sampling frequency */
   sf2 = sf/2;
   t0  = 1.0/sf;   /* sampling period */
   N   = 300;
   pi  = 3.1415926535;

   f1 = 300;
   f2 = 600;
   do i = 1 to N;
      t  = (i-1)*t0;
      x1 = 0.5*sin(2*pi*f1*t);
      x2 = 0.5*sin(2*pi*f2*t);
      x  = x1 + x2;     /* input signal */
      output;
   end;
```

```
run;
```

You can use the following DFIL procedure statements to design a Butterworth lowpass filter that uses the ORDER mode. The filter's normalized upper cutoff frequency is 0.5 and its order *n* is 6. The DFIL procedure generates SAS DATA step code that mimics the computations that are done by applying the designed filter to any input data in the code file f3 that has a file name *example3.sas*. The DFIL procedure also creates an analytic store for the designed filter and saves that store as a binary object in the CAS data table mycas.f.

```
ods graphics on;
ods html;
filename f3 "example3.sas";
proc dfil;
   lowpass upper=0.5 order=6 code=(file=f3 ivar=x svar=y)
           savestate=(rstore=mycas.f ivar=x svar=y);
run;
```

The following four methods show how to use the designed Butterworth lowpass filter to filter the input signal. Each method ensures the sequential order of the input data samples.

Method 1: Filter the Input Signal by Using the Code File in the SAS Client

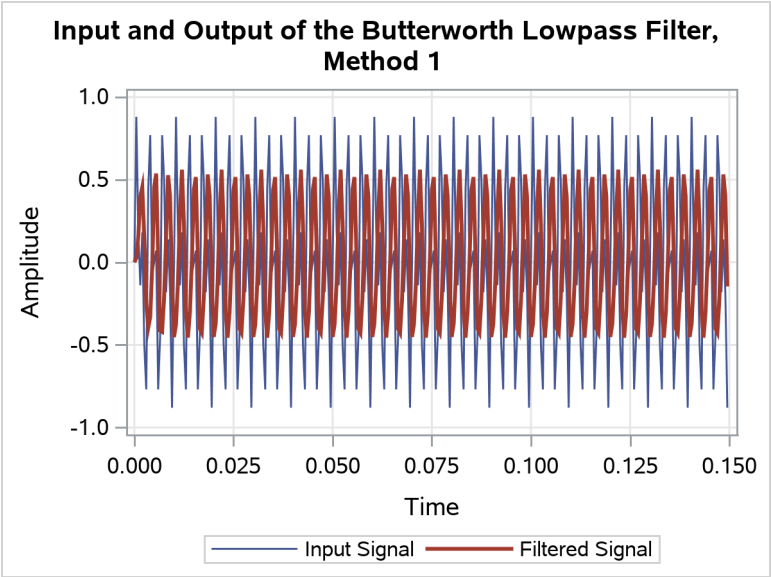
The following DATA step applies the designed lowpass filter to the input signal by using the code file f3 in the SAS client and generates the output in the data table output1.

```
data output1;
   set input;
   %inc f3;
   keep t x y;
run;
```

The following code plots the input signal and filtered output signal, which are shown in Output 4.3.1. The lowpass filter filters out any signal components that are outside of its passband, and the output signal now contains only the component that is of frequency 300 Hz (normalized frequency 0.3).

```
proc sgplot data=output1;
   title "Input and Output of the Butterworth Lowpass Filter, Method 1";
   series x=t y=x / legendlabel="Input Signal";
   series x=t y=y / legendlabel="Filtered Signal" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.3.1 Butterworth Lowpass Filtering, Method 1



Method 2: Filter the Input Signal by Using a Code File in CAS

The samples in a data vector are not guaranteed to be in sequential order in a CAS session. To maintain the sequential order of the input data, use the following steps.

The following DATA step ensures that the input data have only one partition:

```
data input2;
   set input;
   retain p 1;
run;
```

Now you can load the input2 data set into a CAS session by specifying the CAS engine libref in the following DATA step. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.input2;
   set input2;
run;
```

The following DATA step applies the designed lowpass filter to the input signal by using the code file f3 in a CAS session. The input data are partitioned by the first variable (p) and ordered by the second variable (t) in the BY option. The SINGLE=YES option ensures that the DATA step runs in a single thread. This option is useful for running functions that have inter-row dependencies and require that the order of the rows be maintained. Now the sequential order of the input data samples is guaranteed in the CAS session, and the output is generated in the data table mycas.output2.

```
data mycas.output2/single=yes;
   set mycas.input2;
   by p t;
   %inc f3;
run;
```

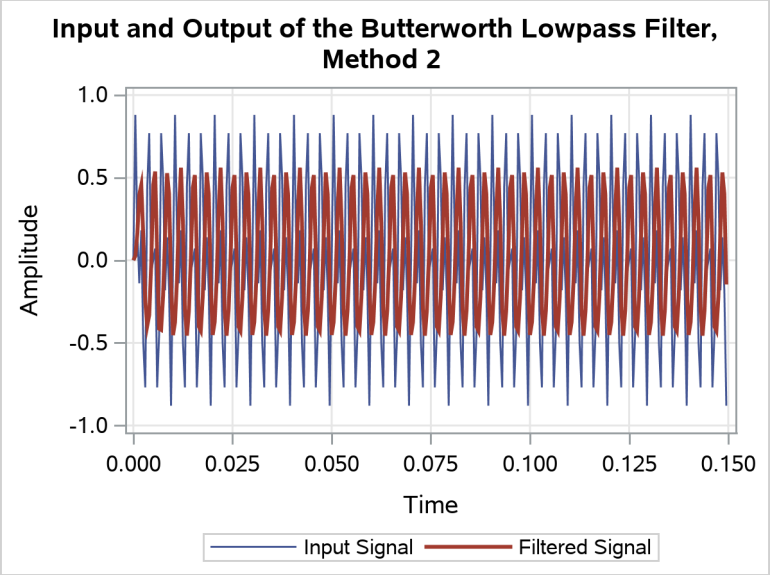
The following two steps copy the CAS data table mycas.output2 to a local data table work.output2 and sort the output data by *t*, so the filtered results are in the same sequential order as the input signal:

```
proc copy in=mycas out=work;
   select output2;
run;

proc sort data=output2;
   by t;
run;
```

The following code plots the input signal and filtered output signal; the plots are shown in Output 4.3.2.

```
proc sgplot data=output2;
   title "Input and Output of the Butterworth Lowpass Filter, Method 2";
   series x=t y=x / legendlabel="Input Signal";
   series x=t y=y / legendlabel="Filtered Signal" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.3.2 Butterworth Lowpass Filtering, Method 2



Method 3: Filter the Input Signal by Downloading the Analytic Store to a Local File and Score the Data in the SAS Client

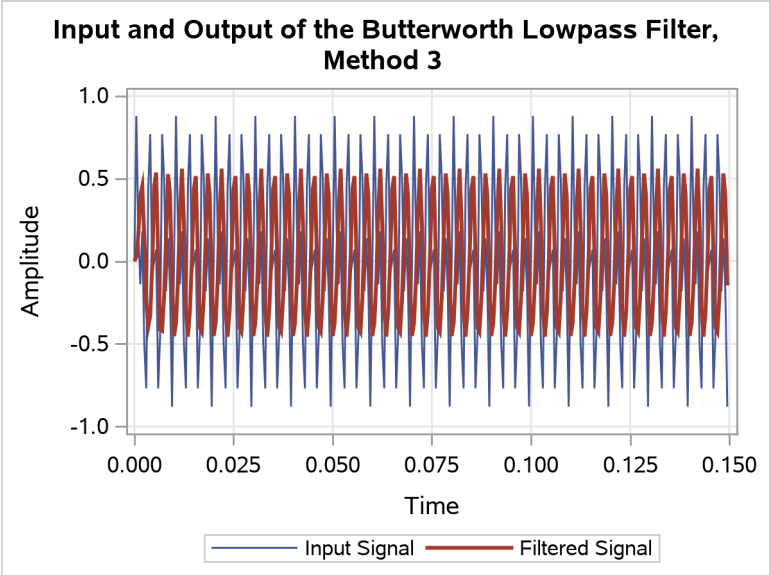
The following ASTORE procedure statements download the analytic store from the CAS table mycas.f to a local analytic store file *f1.astor*, and then use the local analytic store to score the input data. The scoring output is saved in the data table output3.

```
proc astore;
   download rstore=mycas.f store="f1.astor";
   score data=input out=output3 copyvars=(t x) store="f1.astor";
run;
```

The following code plots the input signal and filtered output signal; the plots are shown in Output 4.3.3.

```
proc sgplot data=output3;
   title "Input and Output of the Butterworth Lowpass Filter, Method 3";
   series x=t y=x / legendlabel="Input Signal";
   series x=t y=y / legendlabel="Filtered Signal" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.3.3 Butterworth Lowpass Filtering, Method 3



Method 4: Filter the Input Signal by Using PROC FCMP and a Local Analytic Store File in the SAS Client

The following code uses PROC ASTORE to download the analytic store from the CAS table mycas.f to a local analytic store file *f2.astor*, and then uses the SAS Function Compiler (FCMP) procedure to create a function myfilter to filter the input signal by using the local analytic store file. The advantage of using PROC FCMP is that you can use a DATA step to call the filtering function and perform other tasks (computing the power of the output signal $y2$ in this example), whereas PROC ASTORE in Method 3 can perform only the filtering task.

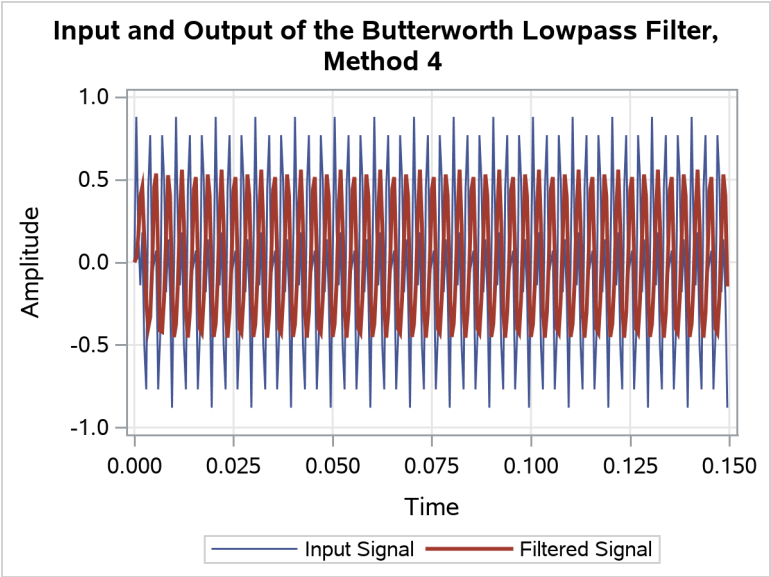
```
proc astore;
   download rstore=mycas.f store="f2.astor";
run;

proc fcmp outlib=work.score.funcs;
   function myfilter(x);
   declare object myscore(astore);
   call myscore.score("f2.astor");
   return (y);
   endfunc;
quit;
```

```
options cmplib=work.score;

data output4;
   set input;
   y = myfilter(x);
   y2 = y*y;
run;
```

The following code plots the input signal and filtered output signal; the plots are shown in Output 4.3.4.

```
proc sgplot data=output4;
   title "Input and Output of the Butterworth Lowpass Filter, Method 4";
   series x=t y=x / legendlabel="Input Signal";
   series x=t y=y / legendlabel="Filtered Signal" lineattrs=(thickness=2);
   xaxis grid label="Time";
   yaxis grid label="Amplitude";
run;
```

Output 4.3.4 Butterworth Lowpass Filtering, Method 4



References

Constantinides, A. G. (1970). "Spectral Transformations for Digital Filters." *Proceedings of the Institution of Electrical Engineers* 117:1585–1590.

Oppenheim, A. V., and Schafer, R. W. (2010). *Discrete-Time Signal Processing*. 3rd ed. New York: Pearson.

Oppenheim, A. V., Willsky, A. S., and Nawab, S. H. (1997). *Signals and Systems*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Parks, T. W., and Burrus, C. S. (1987). *Digital Filter Design*. New York: Wiley-Interscience.

Roy, S. C. D. (2005). "A Simple Derivation of the Spectral Transformations for IIR Filters." *IEEE Transactions on Education* 48:274–278.

Chapter 5
The SMCALIB Procedure

Contents

Overview: SMCALIB Procedure

The SMCALIB procedure runs calibration for one or more projects. PROC SMCALIB is one of five procedures in SAS Visual Forecasting that you can use to manage stability monitoring projects. For more information about stability monitoring, see the section "Stability Monitoring Overview" on page 57.

The SMCALIB procedure performs the following for each model in a project:

1. transforms the variables
2. selects the variables
3. runs the appropriate model type
4. performs the holdout analysis
5. creates the following output:
    - calibration status
    - fit statistics
    - historical fit (fitted and actual values of the target variable, upper and lower prediction limits, and fit error)
    - holdout statistics (also called holdout forecast statistics)
    - holdout forecast (also called holdout fit; forecast and actual values of target variables, upper and lower prediction limits, and forecast error)
    - parameter estimates
    - auxiliary information necessary for scoring (depending on model type)
    - a plot of holdout forecast (forecast and actual values of target variables, upper and lower prediction limits)

Before SMCALIB procedure can be run, you need to have the following:

- one or more model specifications generated by the SMSPEC procedure
- a project repository generated by the SMPROJECT procedure
- a training data table

Stability Monitoring Overview

Four stability monitoring procedures in SAS Visual Forecasting (the SMCALIB, SMSPROJECT, SMSCORE, and SMSPEC procedures) implement a collection of anomaly detection methods that can be used to monitor the "health" of a system and detect anomalous behavior of various kinds of signals, including sensor and event data. One of the goals of stability monitoring is to predict serious issues, such as catastrophic failures, before they occur in order to manage the cost and frequency of preventive maintenance.

The analytical approach of stability monitoring is based on the assumption that within a subset of sensors, events, or both (collectively called "signals") that are related to the system being monitored there exists a "target" signal whose behavior can be explained by other signals (called "explanatory" signals). The main idea is that in a healthy system (that is, healthy during a stable period of time), there is a robust statistical relationship between the target signal and explanatory signals. After the relevant statistical model is selected, fitted on historical data during stable periods, and saved using the SMCALIB procedure, it can then be used for monitoring data that arrive in a new time period. The stored model is used with new data that contain both target and explanatory signals to generate a forecast of a target signal from the explanatory signals (this process is called "scoring" and is performed using the SMSCORE procedure). The forecast is then compared to the actual values of the target signal, and, based on a set of user-defined rules, anomalies in the behavior of the monitored system during the new time period can be flagged.

Stability monitoring can be used in a variety of applications, including the oil and gas industry (oil well pumps), the power generation industry (wind turbines), and so on.

The following list outlines the workflow of stability monitoring:

1. Create one or more model specifications by using the SMSPEC procedure.
2. Create a project repository by using the SMPROJECT procedure. The project repository contains the full project description, which includes the following:
   - training data table and its structure
   - scoring data table and its structure
   - one or more model specifications (generated by the SMSPEC procedure)
   - statistical parameters such as holdout length or significance level.
3. Run project calibration by using SMCALIB procedure on the training data table. For more information, see the section "Calibration Steps" on page 66.
4. Select one or more models for scoring by using the calibration results and the SMSELECT procedure.
5. Run project scoring (also called monitoring) by using the SMSCORE procedure on the scoring data 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 7 in Chapter 3, "Shared Concepts."

Getting Started: SMCALIB Procedure

This example demonstrates the use of stability monitoring procedures, including the SMCALIB procedure, on turbofan engine data. The data are from the turbofan engine degradation simulation data set (Saxena and Goebel 2008) from a data repository that is maintained by the Prognostics Center of Excellence of the National Aeronautics and Space Administration (NASA). The data were generated using NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) software, which simulates the behavior of a large commercial turbofan engine under various environmental, operating, and health conditions. The engine has five rotating parts, each of which can be subject to deterioration. The C-MAPSS software takes as inputs operating conditions such as altitude, speed, and sea-level temperature, in addition to engine health parameters. The outputs include various sensor responses. Each observation is a snapshot for a cycle (called a flight), and the data for each simulation are generated until engine failure. The full data in the repository consist of four training data sets with multiple engine simulations. The data set in this example is taken from training set #3 for engine 24. This example uses the following sensor responses: pressure at fan inlet (InletPressure), total pressure in bypass-duct (TotPressure), corrected core speed (CoreSpeed), bleed enthalpy (Enthalpy), high-pressure and low-pressure turbine coolant bleeds (HPCBleed and LPCBleed, respectively), and the ratio of fuel flow to static pressure at the high-pressure compressor outlet, which is used as the target variable (FuelRatio). As the engine degrades, achieving similar levels of operational performance is expected to be reflected in changes in fuel consumption. Therefore, a variable that is related to fuel consumption is selected as the target (the variable that is monitored to determine the health of the system). The following DATA step creates the data set train:

```
data train;
   input DateTime InletPressure TotPressure FuelRatio
         CoreSpeed Enthalpy HPCBleed LPCBleed;
   datalines;
1 14.62 21.59 519.25 8116.67 390 38.78 23.2995
2 14.62 21.59 520.60 8124.89 391 39.12 23.3948
3 14.62 21.59 519.26 8112.68 390 38.89 23.3420
4 14.62 21.58 518.61 8119.49 389 38.83 23.2713

   ... more lines ...
```

The data for this engine contain 494 observations (cycles). The first 90 cycles are used as the training period (for calibration using SMCALIB procedure), and three consecutive 50-cycle windows starting from cycle 91 are used for condition monitoring (see the section "Getting Started: SMSCORE Procedure" on page 85). The following DATA step loads the training data into a CAS session. This DATA step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.project_1;
   set train(obs=90);
   stable=1;
run;
```

As you can see from the DATA step, stable periods for the training data need to be defined. Any observations that are marked as unstable will be removed from the training.

Next, you need to create a model specification by using the SMSPEC procedure:

```
proc smspec;
   input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
   reg name=mycas.reg1;
   arima name=mycas.arima1;
run;
```

The previous statements request both OLS regression and ARIMA model specifications (stored as mycas.reg1 and mycas.arima1, respectively). Both model specifications use all six explanatory variables, which are specified in the INPUT statement. Because the INPUT statement does not include a TRANSFORM= option or a REQUIRED= option, input variables are not transformed and none of them is required. The target variable does not need to be transformed; therefore, the TARGET statement is omitted.

Next, you need to create a project repository by using the SMPROJECT procedure:

```
proc smproject name=mycas.myproj;
    target FuelRatio;
    input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
    model reg1 arima1;
run;
```

This SMPROJECT procedure call is run with mostly default values. It generates a project repository mycas.myproj (specified in the NAME= option) that contains a single project whose ID is 1 by default (because no ID= option is specified in the PROC SMPROJECT statement). The project expects the training data table to be named mycas.project_1 (because the DATA= option is not specified) and the scoring data set to be named mycas.project_1_score (because the SCOREDATA= option is not specified). The ALPHA= and HOLDOUT= options are omitted; therefore, the significance level is set at 0.05, and the holdout length is set at 10 observations. The project uses FuelRatio as the target variable (as specified in the TARGET statement), and it has six explanatory variables (specified in the INPUT statement). The DATETIME and STABLE statements are omitted; therefore, the datetime variable is named datetime, and the stable flag variable is named stable. The project uses two model specifications, mycas.reg1 and mycas.arima1.

Next, you run the project calibration by using the SMCALIB procedure:

```
proc smcalib name = mycas.myproj;
   output modelfit   = mycas.mfit
          holdoutfit = mycas.hfit
          scoreinfo  = mycas.sinfo;
   store mycas.scoreout;
run;
```

The PROC SMCALIB statement looks for the mycas.myproj project repository. Because the PROJECT statement is omitted, all projects in that repository are calibrated. In this example, the repository contains only one project, whose ID is 1. The OUTPUT statement requests the following SAS output tables to be generated: model fit, historical fit, and scoring information. The STORE statement requests that information for each model necessary for scoring be stored in a binary file mycas.scoreout. The following ODS tables are also produced:

- calibration status
- fit statistics
- holdout statistics
- parameter estimates

The ODS plot of the holdout forecast for each successfully calibrated model for each project is also produced.

The calibration status (Figure 5.1) indicates whether the calibration was successful for each model.

Figure 5.1 Calibration Status

Calibration Status					
Project ID	Model ID	Model Name	Status	RC	Message
1	**0**	Simple	0	0	Command successfully completed.
	1	arima1	0	0	Command successfully completed.
	2	reg1	0	0	Command successfully completed.

As you can see, three models were successfully calibrated. Two of the models were defined by the SMSPEC procedure, and the model whose ID is 0 is an internal model that uses only the target variable (for more information, see the section "Simple Model" on page 68). Each model is given a numeric ID, where an ID of 0 is always an internal simple model and user models are numbered consecutively from 1. Model IDs are used later for scoring.

As you can see in the "Parameter Estimates" table in Figure 5.2, all variables except InletPressure are selected for both models 1 and 2.

Figure 5.2 Parameter Estimates

Parameter Estimates							
Project ID	Model ID	Variable	Transformation	Estimate	Standard Error	t Value	Pr > |t|
1	**1**	Enthalpy	None	0.1385	0.0360	3.85	0.0002
		TotPressure	None	27.2561	8.1585	3.34	0.0013
		HPCBleed	None	1.3936	0.2823	4.94	<.0001
		CoreSpeed	None	0.0398	0.007097	5.60	<.0001
		LPCBleed	None	1.5820	0.4398	3.60	0.0006
	2	Enthalpy	None	0.1412	0.0394	3.59	0.0006
		TotPressure	None	35.3549	8.7616	4.04	0.0001
		HPCBleed	None	1.2343	0.3315	3.72	0.0004
		CoreSpeed	None	0.0420	0.008534	4.93	<.0001
		LPCBleed	None	1.6191	0.5499	2.94	0.0042

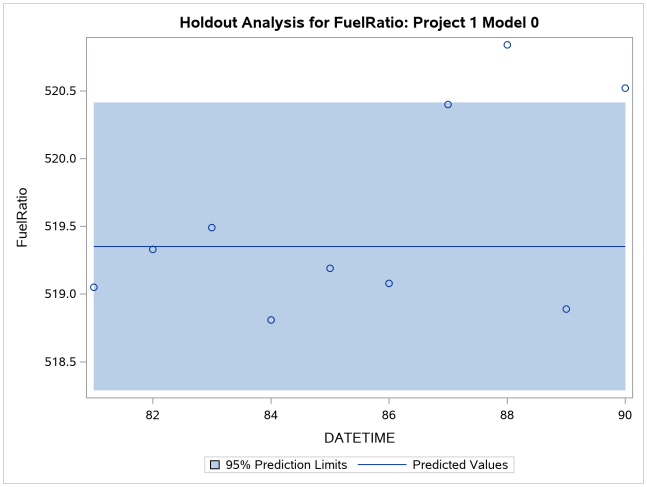
Each model's potential for scoring is evaluated by holdout analysis, which is done as a part of the model calibration process. By default, the last 10 observations are used for the holdout analysis, and the significance level is set to 0.05. To request different values, you can specify the ALPHA= and HOLDOUT= options in the PROC SMPROJECT statement. The holdout analysis results are shown in Figure 5.3. Holdout statistics are slightly better for the ARIMA model, and both model specifications significantly outperform the default (simple) model.

Figure 5.3 Holdout Statistics

Holdout Statistics				
Project ID	Model ID	Model Name	MAPE	RMSE
1	0	Simple	0.1185	0.7044
	1	arima1	0.0589	0.3514
	2	reg1	0.0593	0.3610

In addition to using holdout statistics, you can use the plot of holdout forecast to evaluate a model's potential performance for scoring. The holdout forecasts for all models are shown in Figure 5.4. You can see that the models ID=1 (Arima1) and ID=2 (Reg1) show very similar performance; for both models, all actual values are inside the 95% prediction band. For the model ID=0 (Simple), some of the actual values are outside the 95% prediction band.

Figure 5.4 Holdout Analysis Graphs for Project 1



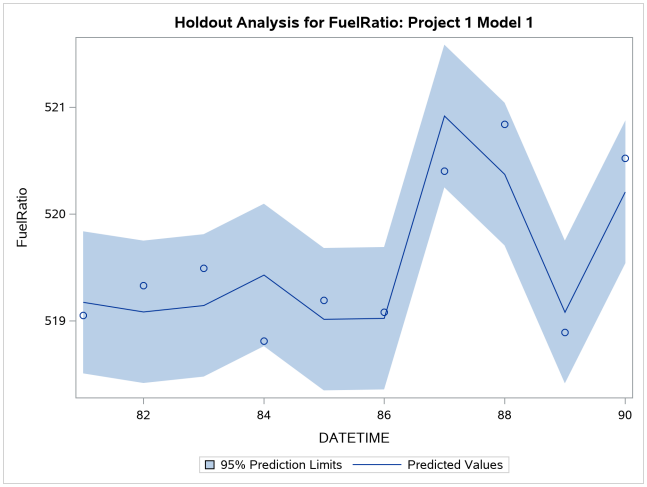
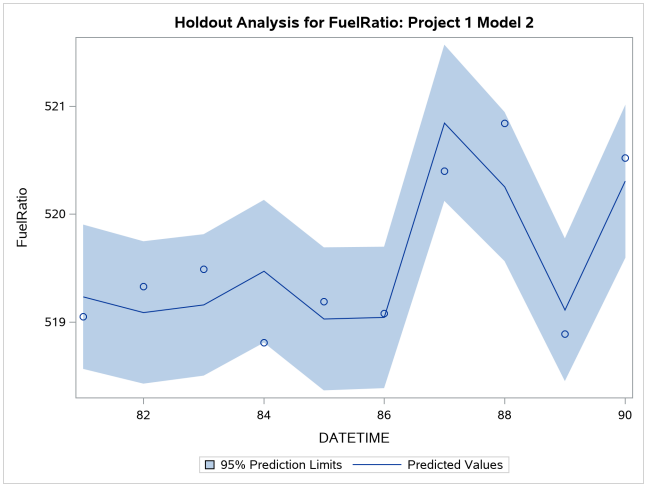


Figure 5.4 *continued*



Syntax: SMCALIB Procedure

The following statements are available in the SMCALIB procedure:

PROC SMCALIB NAME=*CAS-libref.data-table* < *options* > **;**
OUTPUT < *options* > **;**
PROJECT ID= *(project-ID-list)* **;**
STORE *CAS-libref.data-table* **;**

None of the statements is required. If you want to perform scoring after the calibration, then include the STORE statement.

PROC SMCALIB Statement

PROC SMCALIB NAME=*CAS-libref.data-table* < *options* > **;**

The PROC SMCALIB statement invokes the procedure.

You must specify the following required option:

NAME=*CAS-libref.data-table*
specifies the SAS data table (project repository) that contains the projects to be calibrated. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 58.

You can also specify the following *option*:

NTHREADS=*number-of-threads*
specifies the number of threads to use in the computation. The default value is the number of CPUs available in the machine.

OUTPUT Statement

OUTPUT < *options* > **;**

The OUTPUT statement generates output tables that show the project calibration results.

You can specify the following *options*:

HOLDOUTFIT=*CAS-libref.data-table*
H=*CAS-libref.data-table*
specifies the name of the SAS data table in which to store the holdout forecast values and the caslib in which to store the 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 section "Using CAS Sessions and CAS Engine Librefs" on page 58.

The table has the following columns:

- project ID
- model ID
- time ID
- actual values of the target variable, y
- fitted values of the target variable, $\widehat{y}$
- lower confidence limit (LCL)
- upper confidence limit (UCL)
- error defined as $y - \widehat{y}$

The time ID range is for the holdout length (which is specified in either the SMPROJECT procedure or the SMSPEC procedure).

MODELFIT=*CAS-libref.data-table*
FIT=*CAS-libref.data-table*
F=*CAS-libref.data-table*
specifies the name of the SAS data table in which to store fitted values for each model within each project that is calibrated and the caslib in which to store the data table. The structure of this data table is identical to the structure of the HOLDOUTFIT= data table, except that it also contains the time ID range for the whole training 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 section "Using CAS Sessions and CAS Engine Librefs" on page 58.

SCOREINFO=*CAS-libref.data-table*

SI=*CAS-libref.data-table*

specifies the name of the SAS data table in which to store model-specific score information in human-readable format and the caslib in which to store the 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 section "Using CAS Sessions and CAS Engine Librefs" on page 58.

The table has the following columns:

- project ID
- model ID
- specification name
- specification value

PROJECT Statement

PROJECT ID=*project-ID* **;**

PROJECT ID=(*project-ID-list***) ;**

PROJECT ID=_ALL_ ;

The PROJECT statement specifies which project IDs from the current project repository are to be calibrated.

You must specify the ID= option. You can specify it in any of the following forms:

ID=*project-ID*	specifies a single project ID.
ID=(*project-ID-list***)**	specifies a comma-separated list of project IDs in parentheses.
ID=_ALL_	specifies all projects.

A project ID can be any integer between 1 and 999,999,999.

If the PROJECT statement is omitted, then all projects in the current project repository are calibrated.

STORE Statement

STORE *CAS-libref.data-table* **;**

The STORE statement specifies a binary scoring repository in which to store scoring information for each model that is needed by the SMSCORE procedure.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 58.

Details: SMCALIB Procedure

The purpose of stability monitoring calibration is to capture a statistical relationship (model) between the target and explanatory variables in a "healthy" system (that is, a system that is functioning properly and is fully operational). Stability monitoring enables you to specify multiple models and use holdout analysis to select one or more best-performing models. All successfully calibrated models are stored and can be used for monitoring the condition of the machine during scoring.

Selecting Stable Historical Data for Calibration

Before you calibrate, you need to select data from a historical time period during which the system is known to be working properly. Sometimes it is impossible to find a time period that has no downtime or anomaly. For this reason, one of the data requirements is a stable flag variable. The stable flag variable can take one of two values: 1 if the system was stable at the time of the measurement, and 0 if not. Stability monitoring ignores data that are marked as unstable during the calibration process. **NOTE:** It is important that the calibration data are stable at the end of the selected historical period in order for holdout results to be useful for model selection.

Calibration Steps

Calibration performs four major steps: (optional) variable transformation, (optional) variable selection, model parameter estimation, and holdout analysis. These steps are described in the following subsections.

Variable Transformation

You can specify a transformation of the target variable or explanatory variables (or both types of variables) from the list of available transformation types when you use the SMSPEC procedure to generate a model specification.

Target Variable Transformation Types

The following transformation types are available for the target variable:

- none
- natural log: $\ln \mathsf{X}$
- symmetric log: $\operatorname{logm} \mathsf{X} = \operatorname{sign} \mathsf{X} \ln(1 + \operatorname{abs} \mathsf{X})$. This transformation type is useful if the variable can take negative values because $\operatorname{logm}(-\mathsf{X}) = -\operatorname{logm} \mathsf{X}$.

Explanatory Variables Transformation Types

The following transformation types are available for explanatory variables:

- none

- natural log: $\ln \mathsf{X}$
- symmetric log: $\operatorname{logm} \mathsf{X} = \operatorname{sign} \mathsf{X} \ln (1 + \operatorname{abs} \mathsf{X})$
- square: X^2
- cube: X^3

For power transformations (square and cube), all terms up to the requested power are generated. That is, if cube transformation is requested for X_1, then the system generates three variables: X_1, X_1^2, and X_1^3.

Variable Selection

When you specify a model using the SMSPEC procedure, you can mark each explanatory variable as required or optional for the model calibration by using the REQUIRED= option in the INPUT statement. Variables that are marked optional will go through automatic variable selection as a part of calibration process. This variable selection uses backward selection with Akaike's information criterion as the criterion for model fit and a *p*-value threshold of 0.05 for variable significance.

If an explanatory variables is required and its type is a square or cube transform, then only a linear transform is required for modeling, and the higher power transforms are selected (or not) by the calibration process based on their significance. If a variable is marked as required but its parameter value cannot be estimated (for example, the variable has a constant value or all values are missing), then this variable is dropped from the modeling.

Model Parameter Estimation

You list the models for the project calibration in the MODEL statement in the SMPROJECT procedure. Each model listed must be defined by running the SMSPEC procedure. During calibration, each model is assigned a unique identifier inside the project. This identifier is called the model ID. The model IDs are sequential starting from 1. (Model ID=0 is reserved for the simple model type; see the section "Simple Model" on page 68.) The following subsections describe the types of statistical models that are available.

REG Model

An ordinary least squares (OLS) model is referred to as a REG model throughout this chapter. This model assumes an OLS relationship between the target and explanatory variables. Standard OLS assumptions are applied: strict exogeneity, spherical errors (homoscedasticity and no autocorrelation), linearly independent regressors, and iid observations. This model is requested by specifying the REG statement in the SMSPEC procedure.

ARIMA Model

An ARIMA (autoregressive integrated moving average) model predicts the value of the target variable as a linear combination of its own past values, past errors, and current values of explanatory variables. The ARIMA model was first introduced by Box and Jenkins (1976). The following restrictions are imposed on an ARIMA model that is used in stability monitoring:

- All variables are assumed to be stationary. It is assumed that all variables have finite variance, so no differencing of the variables is applied during modeling.
- Only the contemporaneous effect of explanatory variables on the target variable is estimated.

- AR and MA orders are selected automatically by the solution during the calibration process. For any model ID that has specified an ARIMA type, all ARIMA models with AR orders from 0 to 5 and MA orders from 0 to 5 are considered for calibration. That is, 36 possible ARIMA model combinations are created: ARMA(0,0), ARMA(0,1), ARMA(0,2), . . . , ARMA(5,5). The best model is selected by the AICC model fit criterion.

The ARIMA model is requested by specifying ARIMA statement in SMSPEC procedure.

Simple Model

The simple model computes the predicted value of the target variable as its mean historical value during the stable period, and it computes upper and lower prediction limits from the target variable variance and a specified significance level. No explanatory variables are used. You cannot specify a simple model as a model type for calibration; it is run by default during any calibration process. The simple model's results are used if all other models fail to converge or produce results. There is always only one model ID (model ID=0) that has simple model type. In contrast, you can specify multiple models with a REG or ARIMA type in the MODEL statement in the SMPROJECT procedure; of course, each model needs to be defined by running the SMSPEC procedure. Model ID=0 is not specified in the calibration; however, if you want to use model ID=0 as one of the models for scoring, you need to specify it explicitly when you use the SMSCORE procedure to perform the scoring.

Holdout Analysis

Holdout analysis judges a model's ability to score—that is, to predict the value of the target variable. Holdout analysis is used to select one or more models for scoring among all the models that you specify for the project. After significant explanatory variables have been selected and model parameters have been estimated on the full set of the stable historical data for the project, the tail end of the historical data is held out, and the estimated model is applied to the explanatory variables in this date range to predict the values of the target variable and compare the predicted values to the actual values. Holdout analysis has two parameters: the length of the holdout period and the type of holdout statistics. Another parameter, significance level, is used to compute prediction limits for both the historical fit and the holdout forecast.

Length of Holdout

You specify how many observations at the end of the calibration data are to be used for the holdout analysis in HOLDOUT= option in the PROC SMPROJECT statement or the HOLDOUT= option in the REG or ARIMA statement in the SMSPEC procedure. The number of observations is called the holdout length, and it should be strictly less than the length of the calibration data. It is also important that all observations in the holdout period are stable in order to properly compare different models based on holdout statistics. Unstable periods of calibration data are ignored during the model calibration. Unstable data during a holdout period might mean that the estimated statistical model is no longer valid, thus also invalidating the judgment of the model's accuracy from the holdout statistics.

Holdout Statistics

Holdout statistics measure each model's accuracy by comparing actual and predicted values of the target variable from the holdout analysis. The stability monitoring calibration supports two types of holdout statistics: root mean squared error (RMSE) or mean absolute percent error (MAPE),

- $\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} \left(y(i) - \widehat{y}(i)\right)^2}{N}}$

- $\text{MAPE} = \frac{100}{N} \sum_{i=1}^{N} \frac{|y(i) - \widehat{y}(i)|}{|y(i)|}$

where N is the number of valid observations (defined as observations from the stable period, which has no missing values for any of the model variables), $y(i)$ is actual value, and $\widehat{y}(i)$ is predicted value of the target variable at time i.

Output Delivery System Tables and Plots

When SMCALIB procedure is run, the following ODS tables are produced:

- The "Calibration Status" table indicates whether calibration was successful for each model in a project. The table's columns are: project ID, model ID, model name, status (0 for success, 1 for failure), return code (0 if there was no error; otherwise, the error code is stored), and a message that explains the error code (if any).
- The "Fit Statistics" table contains several measures of fit for each model within the project. The table has the following columns: project ID, model ID, model name, AIC, SBC, MAPE, $\overline{R^2}$, RMSE, and σ^2. Each measure is computed using the full range of stable calibration data.
- The "Holdout Statistics" table contains project ID, model ID, model name, MAPE, and RMSE. Each holdout measure is computed over the holdout period (which is determined by the holdout length).
- The "Parameters Estimates" table contains parameter estimates. The table's columns are project ID, model ID, explanatory variable name, explanatory variable transformation, parameter estimate, standard error, t value, and p-value.

The SMCALIB procedure also produces a plot of holdout forecast for each successfully calibrated model for each project. The plot shows actual and forecast values of the target variable, as well as its lower and upper prediction limits for the holdout period.

Formulas for Statistical Measures

The stability monitoring action set uses the following formulas for the statistical measures:

- Akaike's information criterion, $\text{AIC} = N \ln\left(\frac{\text{SSE}}{N}\right) + 2D$
- Schwarz Bayesian criterion, $\text{SBC} = N \ln\left(\frac{\text{SSE}}{N}\right) + D \ln(N)$
- variance, $\sigma^2 = \frac{\text{SSE}}{N-D}$
- mean absolute percentage error, $\text{MAPE} = \frac{100}{N} \sum_{i=1}^{N} \frac{|y(i) - \widehat{y}(i)|}{|y(i)|}$
- root mean square error, $\text{RMSQ} = \sqrt{\frac{\text{SSE}}{N}}$

- adjusted coefficient of determination, $\overline{R^2} = 1 - \left(\left(1 - R^2\right)\left(\frac{N-1}{N-D}\right)\right)$

where:

- R^2 is the coefficient of determination
- N is the number of valid observations
- $y(i)$ is the actual value
- $\widehat{y}(i)$ is the predicted value
- SSE is the sum of squared errors: $\text{SSE} = \sum_{i=1}^{N} (y(i) - \widehat{y}(i))^2$
- D is the degrees of freedom: $D = N_x + I$ where
  - N_x is the number of dependent variables
  - $I = 1$ if there is an intercept; $I = 0$ otherwise

Example: SMCALIB Procedure

Example 5.1: Using the PROJECT Statement

The following example demonstrates how a SMCALIB procedure call can be done using the PROJECT statement. The data are the same as the data that are used in the section "Getting Started: SMCALIB Procedure" on page 59.

```
proc smcalib name = mycas.myproj;
   project id = 1;
   output modelfit   = mycas.mfit
          holdoutfit = mycas.hfit
          scoreinfo  = mycas.sinfo;
   store mycas.scoreout;
run;
```

The PROJECT statement explicitly requests the calibration of project ID=1. Because the project repository mycas.myproj contains only one project and that project's ID=1, this call is identical to a call in which the PROJECT statement is omitted or the ID=_ALL_ option is specified in the PROJECT statement.

References

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

Saxena, A., and Goebel, K. (2008). "Turbofan Engine Degradation Simulation Data Set." Accessed January 17, 2017. NASA Ames Prognostics Data Repository. NASA Ames Research Center, Moffett Field, CA. https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/.

Chapter 6
The SMPROJECT Procedure

Contents

Overview: SMPROJECT Procedure

The SMPROJECT procedure generates a description of a stability monitoring (SM) project. PROC SMPROJECT is one of five procedures in SAS Visual Forecasting that you can use to manage stability monitoring projects. For more information about stability monitoring, see the section "Stability Monitoring Overview" on page 57.

A stability monitoring project description consists of the following components:

- training data set and its structure
- scoring data set and its structure
- one or more model specifications (generated by SMSPEC procedure)
- statistical parameters such as holdout length or significance level

The SMPROJECT procedure generates a SAS data set (called a project repository) that describes the structure of an SM project. The project repository is used for the project calibration by the SMCALIB procedure and project scoring (also called monitoring) by the SMSCORE procedure.

It is recommended that you use the SMSPEC procedure to create model specifications before you run the SMPROJECT procedure.

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 7 in Chapter 3, “Shared Concepts.”

Getting Started: SMPROJECT Procedure

The following code illustrates the use of the SMPROJECT procedure:

```
libname a ".";
proc smproject
      name      = a.myproj
      id        = 100
      holdout   = 20
      alpha     = 0.05
      data      = a.project_100
      scoredata = a.project_100_score
      forceholdout
      forcealpha
      ;
   target FuelRatio;
   input FanInletPressure BypassTotPressure CoreSpeed
         Enthalpy HPTCoolantBleed LPTCoolantBleed;
   stable stable;
   datetime datetime;
   model arima1 reg1 arima_log reg_log / force nowarn;
run;
```

The PROC SMPROJECT statement creates a project repository named a.myproj (as specified in the NAME= option) and with a project ID of 100 (as specified in the ID= option). The DATA= option specifies the name a.project_100 for the training data set, and the SCOREDATA= option specifies the name a.project_100_score for the scoring data set. The project description specifies 20 holdout observations (as specified in the HOLDOUT= option) and a significance level of ALPHA=0.05. The FORCEHOLDOUT and FORCEALPHA options will cause any model-specific holdout and significance level values to be overwritten. The INPUT statement specifies six explanatory variables in the training data set to be used in the project. The TARGET statement specifies that the target variable is FuelRatio. The STABLE statement names the stable flag variable stable, and the DATETIME statement names the datetime variable datetime. The variable structure of the scoring data set must be identical to the variable structure of the training data set except for the stable flag variable. The MODEL statement requests that four model specifications for the project be saved in the project repository, and the FORCE option causes the specification names to be saved even if they do not exist yet. The NOWARN option suppresses any warnings that the model specifications do not yet exist. However, you must use the SMSPEC procedure to create these model specifications before you can use them in project calibration and scoring.

Syntax: SMPROJECT Procedure

The following statements are available in the SMPROJECT procedure:

PROC SMPROJECT NAME=< *CAS-libref.*>*data-table* < *options*> **;**
DATETIME *variable-name* **;**
INPUT *variable-list* **;**
MODEL *model-spec-list* < / < *options*> > **;**
STABLE *variable-name* **;**
TARGET *variable-name* **;**

The TARGET and MODEL statements are required.

PROC SMPROJECT Statement

PROC SMPROJECT NAME=< *CAS-libref.*>*data-table* < *options*> **;**

The PROC SMPROJECT statement invokes the procedure. You must specify the following required option:

NAME=< *CAS-libref.*>*data-table*

N=< *CAS-libref.*>*data-table*
generates a SAS data set to contain the project repository. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 74.

You can also specify the following *options*:

ALPHA=*significance-level*

A=*significance-level*
specifies the significance level for computing confidence limits for historical fit and holdout forecast. By default, ALPHA=0.05.

BEST=*number*
specifies how many best models from the successfully calibrated models for the project are to be selected for scoring. This option can be overridden by specifying BEST= option in the PROJECT statement of the SMSELECT procedure (see the section "PROJECT Statement" on page 106). By default, BEST=1.

DATA=< *CAS-libref.*>*data-table*
specifies a name to be used for the training data table for this project. This data table is used by the SMCALIB procedure for project calibration. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 74.

By default, the data table name is PROJECT_ followed by the project ID (as specified in the ID= option), and the data table is stored in the same libref where the project repository is stored.

FORCEALPHA

FA

forces the value of the ALPHA= option in this statement (this value is called the project-defined significance level) to override any model-specific values that are specified in the ALPHA= option in the REG or ARIMA statement of the SMSPEC procedure.

FORCEHOLDOUT

FH

forces the value of the HOLDOUT= option in this statement (this value is called the project-defined holdout length) to override any model-specific holdout values that are specified in the HOLDOUT= option in the REG or ARIMA statement of the SMSPEC procedure. If any model specification uses model-specific holdout values, it is important to include this option for a proper cross-model comparison in order to select one or more of the best-performing models for scoring.

HOLDOUT=*holdout-length*

H=*holdout-length*

specifies the number of observations at the end of the training data set to be used for a holdout analysis for the project. The holdout length should be strictly less than the length of the calibration data. It is also important that all observations in the holdout period are stable in order to properly compare different models on the basis of holdout statistics. Unstable periods of calibration data are ignored during the model calibration. Unstable data during a holdout period might mean that the estimated statistical model is no longer valid, thus also invalidating judgment of the model's accuracy from the holdout statistics. By default, HOLDOUT=10.

ID=*project-ID*

specifies a numerical project ID. A project ID can be any integer between 1 and 999,999,999. By default, ID=1.

SELECT=MAPE | RMSE

specifies the model selection criterion to select the best models for scoring according to the holdout analysis of the successfully calibrated models for the project: mean absolute percentage error (MAPE) or root mean square error (RMSE). This option can be overridden by specifying SELECT= option in the PROJECT statement of the SMSELECT procedure (see the section "PROJECT Statement" on page 106). By default, SELECT=MAPE.

SCOREDATA=< *CAS-libref.***>***data-table*

SCORE=< *CAS-libref.***>***data-table*

specifies the name of the scoring data table for this project. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 74.

This data table is used by the SMSCORE procedure for project scoring (monitoring). By default, the data table name is the project ID (as specified in the ID= option) prefixed by PROJECT_ and suffixed by _SCORE, and the data table is stored in the same libref where the project repository is stored.

DATETIME Statement

DATETIME *variable-name* **;**

The DATETIME statement specifies a datetime variable. Training and scoring data sets are required to be a proper time series (that is, they have a constant time difference between any two consecutive observations). If the statement is omitted, then datetime is used as the name of the datetime variable.

INPUT Statement

INPUT *variable-list* **;**

The INPUT statement lists all explanatory variables available in the project. The explanatory variables that are specified in the INPUT statement in the SMSPEC procedure must be in this list. If the project has no explanatory variables, then this statement can be omitted.

MODEL Statement

MODEL *model-spec-list* < / *options* > **;**

SPEC *model-spec-list* < / *options* > **;**

The MODEL statement lists the names of model specifications that are used for the project calibration. When SMCALIB procedure performs project calibration, the model specifications generated by the SMSPEC procedure need to be stored in SAS data sets in the same CAS engine libref where the project repository (as specified in the NAME= option in the PROC SMPROJECT statement) is stored. Because of this restriction, only one-level model specification names are accepted in MODEL statement.

You can specify the following *options*:

FORCE

forces model specifications that do not exist in the project repository libref to be included in the project. By default, when the SMPROJECT procedure creates a project repository, it checks whether each model specification that is listed in the MODEL statement exists as a SAS data set in the same libref as the project repository. If a model specification that is specified in the MODEL statement is not found in that libref, then that model specification is excluded from the project, and an appropriate warning message is written in the log. If none of the model specification data sets are found, then the project repository is not created. To override the default behavior, you can specify the FORCE option so that model specification names are included in the project description even if they are not found. In this case, a warning is written in the log unless the NOWARN option is specified.

NOWARN

suppresses warnings that one or more model specifications are not found.

NOTE: The FORCE and NOWARN options are mutually independent.

STABLE Statement

STABLE *variable-name* **;**

The STABLE statement defines a stable flag variable to use for the project calibration. This variable is required for the training data set. This variable can take only one of two values: 1 if an observation is for the stable condition, or 0 otherwise. All observations marked unstable will be excluded from the project calibration. If the STABLE statement is omitted, then stable is used as stable flag variable name.

TARGET Statement

TARGET *variable-name* **;**

The TARGET statement specifies the target variable—that is, the variable that is used for condition monitoring. This variable needs to exist in both the training and scoring data sets. Only one target variable is allowed for a project. This statement is required.

Details: SMPROJECT Procedure

Target and Explanatory Variables

You must specify one and only one target variable (that is, the variable to be monitored) for each project. This variable (or its transformation, see the section "TARGET Statement" on page 115) is used as a dependent variable in all models that are specified for the project. Typically, you also select a list of explanatory variables for the project, which are used to build statistical models for calibration. Specifying explanatory variables is not required; however, you should include in the project all the variables that can potentially explain the behavior of the target variable.

Specifying Significance Level and Holdout Length in the SMPROJECT and SMSPEC Procedures

You can specify the significance level both for a particular model specification (in the ALPHA= option in the REG statement or the ARIMA statement in the SMSPEC procedure) and for an entire project (in the ALPHA= option in the PROC SMPROJECT statement). The same applies for the holdout length (in the HOLDOUT= option in the REG statement or the ARIMA statement in the SMSPEC procedure or for an entire project in the HOLDOUT= option in the PROC SMPROJECT statement). This section describes the behavior of both the SMSPEC and SMPROJECT procedures regarding these options and offers suggestions for recommended use.

Significance level is used in stability monitoring to compute confidence limits of historical fit and holdout forecast during model calibration and to compute prediction limits during scoring. Holdout length specifies how many observations at the end of the training data are used for holdout analysis during model calibration;

holdout analysis is used to select one or more best models for scoring. For proper comparison of different models during model calibration, it is important to have the same holdout length for all models in the project in order to compare holdout statistics. During scoring, process stability is evaluated by comparing the actual value of the target variable to prediction limits, so it is important to have the same significance level for all scoring models if more than one model is selected for scoring. Because of these considerations, significance level and holdout length are not stored by default when the SMSPEC procedure runs; however, both values are stored by default when the SMPROJECT procedure runs. You can choose to store a model-specific value of significance level or holdout length (or both). By default, model-specific values override project values of these two options. Therefore, you need to be aware that for a proper cross-model comparison during calibration and scoring, both significance level and holdout length need to be the same for all models in the project. This can be done by specifying the FORCEALPHA and FORCEHOLDOUT options in the PROC SMPROJECT statement.

Repository with Multiple Projects

A single call of SMCALIB procedure can calibrate multiple projects that are located in the same project repository. Similarly, a single call of the SMSCORE procedure can score multiple projects that are located in the same project repository. Currently, the SMPROJECT procedure generates a project repository for only a single project. A second call of the procedure with the same name specified for the project repository will overwrite the existing project repository. To create a project repository that has multiple projects, you need to make several calls of SMPROJECT procedure, and then merge the resulting project repositories by using either the DATA step or the APPEND procedure. For an example that uses the DATA step, see "Example 6.2: Creating a Repository That Contains Multiple Projects" on page 81.

Examples: SMPROJECT Procedure

Example 6.1: A Call with Default Options

The following code illustrates a call of the SMPROJECT procedure that omits all optional statements (except the INPUT statement) and all optional options:

```
libname a ".";
proc smproject name = a.myproj;
   target FuelRatio;
   input FanInletPressure BypassTotPressure CoreSpeed
         Enthalpy HPTCoolantBleed LPTCoolantBleed;
   model arima1 reg1;
run;
```

The PROC SMPROJECT stores the project repository in a SAS data set a.myproj (as specified in the NAME= option). The ID= option is omitted; therefore, the default value of 1 is used for the project ID. The HOLDOUT= option is omitted; therefore, the holdout length is set to 10. The ALPHA= option is omitted, so the default value of 0.05 is used for the significance level. Because the FORCEHOLDOUT and FORCEALPHA options are omitted, both holdout length and significance level are not enforced and will be overridden by corresponding model-specific values if they are specified in the SMSPEC procedure. The

DATA= and SCOREDATA= options are omitted; therefore, the training data set name is a.project_1, and the scoring data set name is a.project_1_score. The TARGET statement specifies FuelRatio variable as the target variable for the project, and the INPUT statement specifies six explanatory variables that are available for the project. The DATETIME and STABLE statements are omitted; therefore, the name of the datetime variable and stable flag variable are datetime and stable, respectively. The MODEL statement requests two model specifications, arima1 and reg1, to be used for the project calibration. The FORCE option in the MODEL statement is omitted; therefore, the corresponding SAS data sets, a.arima1 and a.reg1, must exist. **NOTE:** It is assumed that prior to this call, model specifications a.arima1 and a.reg1 were generated by SMSPEC procedure call; otherwise, the project repository is not created.

Example 6.2: Creating a Repository That Contains Multiple Projects

The following code demonstrates how to create a repository that contains two projects:

```
libname a ".";
proc smproject name = a.myproj id = 10 holdout = 20
               forceholdout forcealpha;
   target FuelRatio;
   input FanInletPressure BypassTotPressure CoreSpeed
         Enthalpy HPTCoolantBleed LPTCoolantBleed;
   model arima1 reg1 / force nowarn;
run;

proc smproject name = a.myproj1 id = 11 holdout = 25 alpha = 0.1
               forceholdout forcealpha;
   target FuelConsumption;
   input TirePressure RPM Gear OilTemperature AirTemperature;
   datetime timevar;
   stable stablevar;
   model model1 model2  / force nowarn;
run;

data a.myproj;
   merge a.myproj a.myproj1;
   by proj_id;
run;
```

The first PROC SMPROJECT call creates a project with ID=10. The NAME= option specifies a.myproj as the name of the project repository. The HOLDOUT= option specifies 20 holdout observations. The ALPHA= option is omitted; therefore, the default significance level value of 0.05 will be used. The FORCEHOLDOUT and FORCEALPHA options are specified, so both the holdout length and significance level are enforced and will take precedence over any model-specific values. The DATA= and SCOREDATA= options are omitted; therefore, the training data set name is a.project_10 and the scoring data set name is a.project_10_score. The TARGET statement specifies FuelRatio variable as the target variable for the project, and the INPUT statement specifies six explanatory variables that are available for the project. The DATETIME and STABLE statements are omitted; therefore, the name of the datetime variable and stable flag variable are datetime and stable, respectively. The MODEL statement requests two model specifications (arima1 and reg1) to be used for the project calibration. The model specifications are saved in the project even if they do not yet exist (because the FORCE option is specified), and warnings are suppressed if a model specification does not exist (because the NOWARN option is specified).

The second PROC SMPROJECT call creates a project with ID=11. The NAME= option specifies a.myproj1 as the name of the project repository. The HOLDOUT= option specifies 11 holdout observations, and the ALPHA= option specifies a significance level of 0.1. The FORCEHOLDOUT and FORCEALPHA options are specified, so both the holdout length and significance level are enforced and will take precedence over any model-specific values. The DATA= and SCOREDATA= options are omitted; therefore, the training data set name is a.project_11 and the scoring data set name is a.project_11_score. The TARGET statement specifies FuelConsumption variable as the target variable for the project, and the INPUT statement specifies five explanatory variables that are available for the project. The DATETIME statement names timevar as a datetime variable, and the STABLE statement names stablevar as a stable flag variable. The MODEL statement requests two model specifications (model1 and model2) to be used for the project calibration. The model specifications are saved in the project even if they do not yet exist (because the FORCE option is specified), and warnings are suppressed if a model specification does not exist (because the NOWARN option is specified).

The DATA step merges two project repositories into one repository named a.myproj. This project repository now contains two projects, and project information is sorted by proj_id variable.

Chapter 7
The SMSCORE Procedure

Contents

Overview: SMSCORE Procedure

The SMSCORE procedure runs scoring for one or more stability monitoring projects. PROC SMSCORE is one of five procedures in SAS Visual Forecasting that you can use to manage stability monitoring projects. For more information about stability monitoring, see the section "Stability Monitoring Overview" on page 57.

Before you can run the SMSCORE procedure for a project, you need to run calibration for this project by using the SMCALIB procedure (see Chapter 5, "The SMCALIB Procedure").

Stability monitoring scoring can also be done by using an analytic store and the ASTORE procedure. For this method, it is important to note that the scoring for stability monitoring requires strict maintenance of the sequential order of the input data samples; otherwise, the scoring will fail or incorrect scoring results will be generated. For more information about how to ensure the sequential order of the input data for this method, see "Example 7.2: Scoring Using a Local Analytic Store" on page 98.

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 7 in Chapter 3, "Shared Concepts."

Getting Started: SMSCORE Procedure

This example demonstrates the ability of stability monitoring to capture gradual degradation of a turbofan engine long before a catastrophic failure occurs. The example uses all four of the stability monitoring procedures: PROC SMSPEC, PROC SMPROJECT, PROC SMCALIB, and PROC SMSCORE. This example is similar to the one in the section "Getting Started: SMCALIB Procedure" on page 59; however, this example also includes scoring (also called monitoring).

The data are from the turbofan engine degradation simulation data set (Saxena and Goebel 2008) from a data repository that is maintained by the Prognostics Center of Excellence of the National Aeronautics and Space Administration (NASA). The data were generated using NASA's Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) software, which simulates the behavior of a large commercial turbofan engine under various environmental, operating, and health conditions. The engine has five rotating parts, each of which can be subject to deterioration. The C-MAPSS software takes as inputs operating conditions such as altitude, speed, and sea-level temperature, in addition to engine health parameters. The outputs include various sensor responses. Each observation is a snapshot for a cycle (called a flight), and the data for each simulation are generated until engine failure. The full data in the repository consist of four training data sets with multiple engine simulations. The data set in this example is taken from training set #3 for engine 24. This example uses the following sensor responses: pressure at fan inlet (InletPressure), total pressure in bypass-duct (TotPressure), corrected core speed (CoreSpeed), bleed enthalpy (Enthalpy), high-pressure and low-pressure turbine coolant bleeds (HPCBleed and (LPCBleed, respectively), and the ratio of fuel flow to static pressure at the high-pressure compressor outlet, which is used as the target variable (FuelRatio). As the engine degrades, achieving similar levels of operational performance is expected to be reflected in changes in fuel consumption. Therefore, a variable that is related to fuel consumption is selected as the target (the variable that is monitored to determine the health of the system). The following DATA step creates the data set train:

```
data train;
   input DateTime InletPressure TotPressure FuelRatio
         CoreSpeed Enthalpy HPCBleed LPCBleed;
   datalines;
1 14.62 21.59 519.25 8116.67 390 38.78 23.2995
2 14.62 21.59 520.60 8124.89 391 39.12 23.3948
3 14.62 21.59 519.26 8112.68 390 38.89 23.3420
4 14.62 21.58 518.61 8119.49 389 38.83 23.2713

   ... more lines ...
```

The data for this engine contain 494 observations (cycles). The first 90 cycles are used as the training period (which uses the SMCALIB procedure for calibration), and three consecutive 50-cycle windows (starting from cycle 91) are used for condition monitoring. The following DATA step loads the training data into a CAS session. This DATA step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.project_1;
   set train(obs=90);
   stable=1;
run;
```

As you can see from the DATA step, stable periods for the training data need to be defined. Any observations that are marked as unstable will be removed from the training.

Next, you need to create a model specification by using the SMSPEC procedure:

```
proc smspec;
   input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
   reg name=mycas.reg1;
   arima name=mycas.arima1;
run;
```

The previous statements request both an OLS regression model specification (stored as mycas.reg1) and an ARIMA model specification (stored as mycas.arima1). Both model specifications use all six explanatory variables, which are specified in the INPUT statement. Because the INPUT statement does not include a TRANSFORM= option or a REQUIRED= option, input variables are not transformed and none of them is required. The target variable does not need to be transformed; therefore, the TARGET statement is omitted.

Next, you need to create a project repository by using the SMPROJECT procedure:

```
proc smproject name=mycas.myproj;
    target FuelRatio;
    input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
    model reg1 arima1;
run;
```

This SMPROJECT procedure call is run with mostly default values. It generates a project repository mycas.myproj (as specified in the NAME= option) that contains a single project whose ID is 1 by default (because no ID= option is specified in the PROC SMPROJECT statement). The project expects the training data table to be named mycas.project_1 (because the DATA= option is not specified) and the scoring data table to be named mycas.project_1_score (because the SCOREDATA= option is not specified). The ALPHA= and HOLDOUT= options are omitted; therefore, the significance level is set at 0.05, and holdout length is set at 10 observations. The project uses FuelRatio as a target variable (as specified in the TARGET statement), and it has six explanatory variables (specified in the INPUT statement). The DATETIME and STABLE statements are omitted; therefore, the datetime variable is named datetime, and the stable flag variable is named stable. The project uses two model specifications, mycas.reg1 and mycas.arima1.

Next, you run the project calibration by using the SMCALIB procedure:

```
ods output holdoutstats=holdstat;

proc smcalib name = mycas.myproj;
   output modelfit   = mycas.mfit
          holdoutfit = mycas.hfit
          scoreinfo  = mycas.sinfo;
   store mycas.scoreout;
run;

ods output close;
```

The PROC SMCALIB statement looks for the mycas.myproj project repository. Because the PROJECT statement is omitted, all projects in that repository are calibrated. In this example, the repository contains only one project, whose ID is 1. The OUTPUT statement requests that SAS output tables be generated for model fit, historical fit, and scoring information. The STORE statement requests that information for each

model necessary for scoring be stored in a binary file mycas.scoreout. The following ODS tables are also produced: calibration status, fit statistics, holdout statistics, and parameter estimates.

The calibration status (Figure 7.1) indicates whether the calibration was successful for each model.

Figure 7.1 Calibration Status

Calibration Status					
Project ID	Model ID	Model Name	Status	RC	Message
1	0	Simple	0	0	Command successfully completed.
	1	arima1	0	0	Command successfully completed.
	2	reg1	0	0	Command successfully completed.

As you can see, three models were successfully calibrated. Two of the models were defined by the SMSPEC procedure, and the model whose ID is 0 is an internal model that uses only the target variable (for more information, see the section "Simple Model" on page 68). Each model is given a numeric ID, where an ID of 0 is always an internal simple model and user models are numbered consecutively from 1. Model IDs are used later for scoring.

As you can see in the "Parameter Estimates" table in Figure 7.2, all variables except InletPressure are selected for both models 1 and 2.

Figure 7.2 Parameter Estimates

Parameter Estimates							
Project ID	Model ID	Variable	Transformation	Estimate	Standard Error	t Value	Pr > |t|
1	1	Enthalpy	None	0.1385	0.0360	3.85	0.0002
		TotPressure	None	27.2561	8.1585	3.34	0.0013
		HPCBleed	None	1.3936	0.2823	4.94	<.0001
		CoreSpeed	None	0.0398	0.007097	5.60	<.0001
		LPCBleed	None	1.5820	0.4398	3.60	0.0006
	2	Enthalpy	None	0.1412	0.0394	3.59	0.0006
		TotPressure	None	35.3549	8.7616	4.04	0.0001
		HPCBleed	None	1.2343	0.3315	3.72	0.0004
		CoreSpeed	None	0.0420	0.008534	4.93	<.0001
		LPCBleed	None	1.6191	0.5499	2.94	0.0042

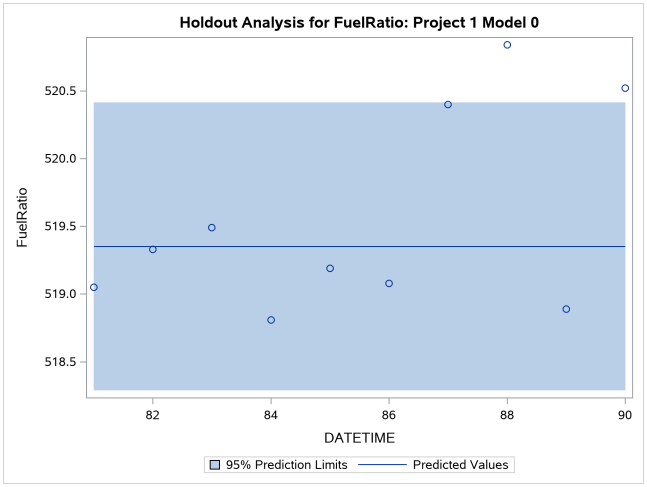
Each model's potential for scoring is evaluated by holdout analysis, which is done as part of the model calibration process. By default, the last 10 observations are used for the holdout analysis, and the significance level is set to 0.05. To request different values, you can specify the ALPHA= and HOLDOUT= options in PROC SMPROJECT statement. The holdout analysis results are shown in Figure 7.3. Holdout statistics are slightly better for the ARIMA model, and both model specifications significantly outperform the default (simple) model.

Figure 7.3 Holdout Statistics

Holdout Statistics				
Project ID	Model ID	Model Name	MAPE	RMSE
1	0	Simple	0.1185	0.7044
	1	arima1	0.0589	0.3514
	2	reg1	0.0593	0.3610

In addition to using holdout statistics, you can use the plot of holdout forecast to evaluate a model's potential performance for scoring. The holdout forecasts for all models are shown in Figure 7.4. You can see that the models ID=1 (Arima1) and ID=2 (Reg1) show very similar performance; for both models, all actual values are inside the 95% prediction band. For the model ID=0 (Simple), some of the actual values are outside the 95% prediction band.

Figure 7.4 Holdout Analysis Graphs for Project 1



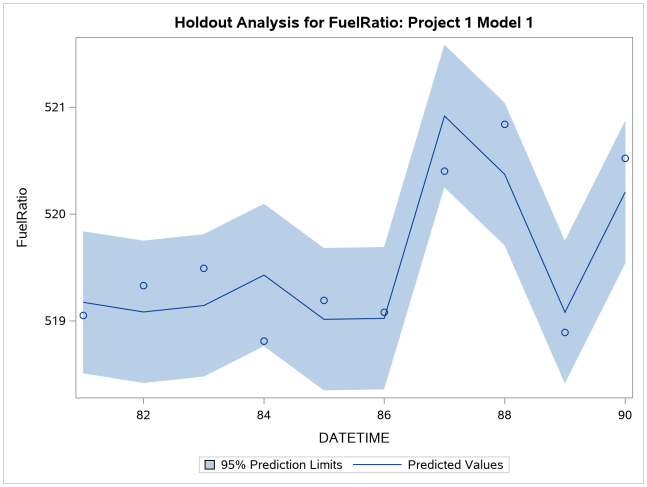
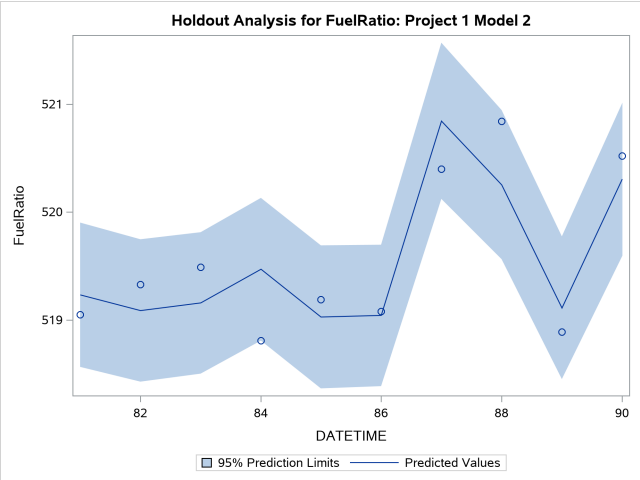


Figure 7.4 *continued*



Both the Arima1 and Reg1 model specifications demonstrate similar scoring performance according to the holdout analysis, so both models are used in stability monitoring scoring. **NOTE:** You can use PROC SMSELECT for model selection. For more information, see Chapter 8, "The SMSELECT Procedure." See "Example 7.1: Scoring with Automated Model Selection" on page 97 for an example of using PROC SMSELECT before calling PROC SMSCORE. Scoring data for the first 50-cycle window (starting from cycle 91) are created in the following DATA step:

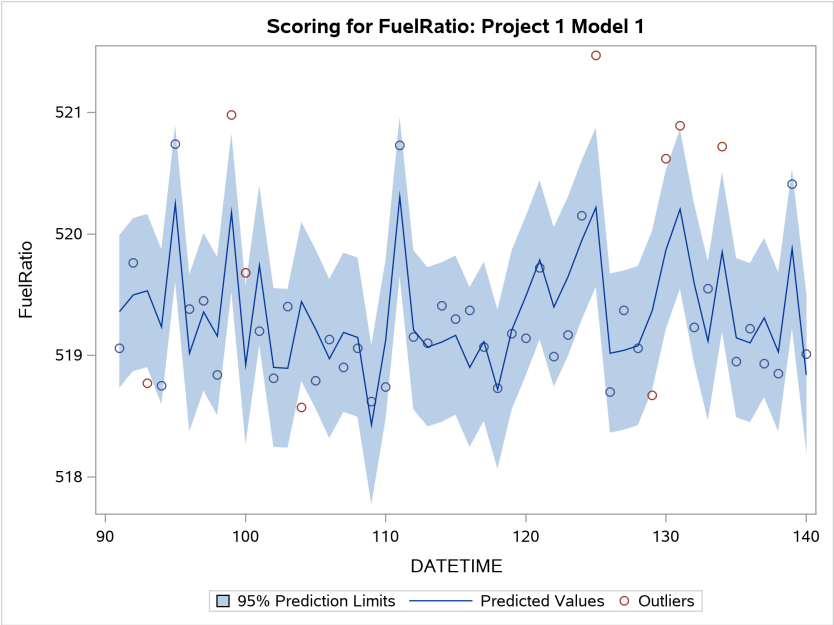
```
data mycas.project_1_score;
   set train(firstobs=91 obs=140);
run;
```

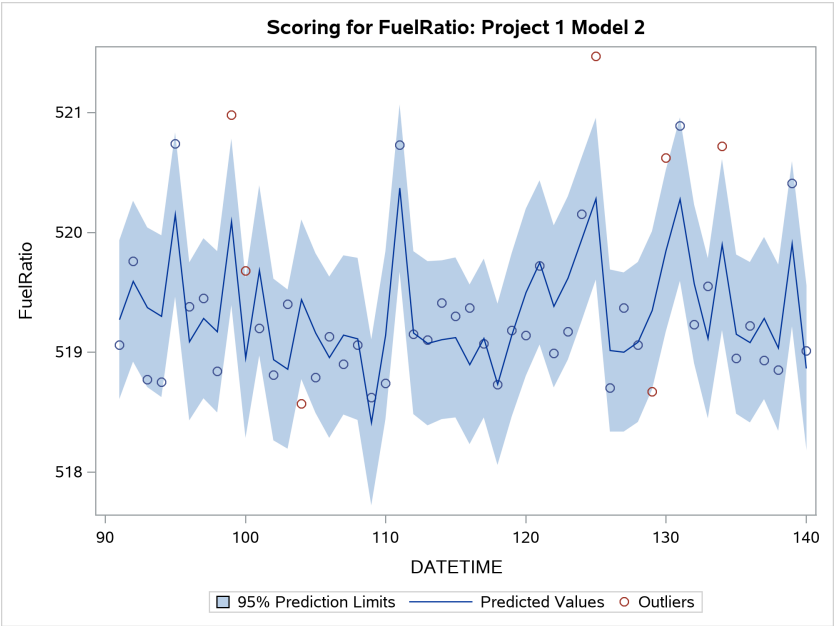
The following statements run scoring for this window:

```
proc smscore name = mycas.myproj scorerepository=mycas.scoreout;
   project id = 1 modelid = (1,2);
   output out = mycas.score;
run;
```

The PROC SMSCORE statement looks for the project repository mycas.myproj (as specified in the required NAME= option) and for the binary file mycas.scoreout (which contains information necessary for scoring and is specified in the required SCOREREPOSITORY= option). The PROJECT statement requests a project with ID=1 that uses models with IDs 1 and 2 (as specified in the MODELID= option). Scoring produces predicted values of the target variable with prediction limits for each model within the project. The OUT= option in the OUTPUT statement stores the output in the mycas.score data table. The scoring results for both model IDs for cycles 91–140 are shown in Figure 7.5.

Figure 7.5 Scoring for Cycles 91–140





Next, you want to perform scoring for the next 50-cycle window, starting from cycle 141. The data for scoring are created in the following DATA steps:

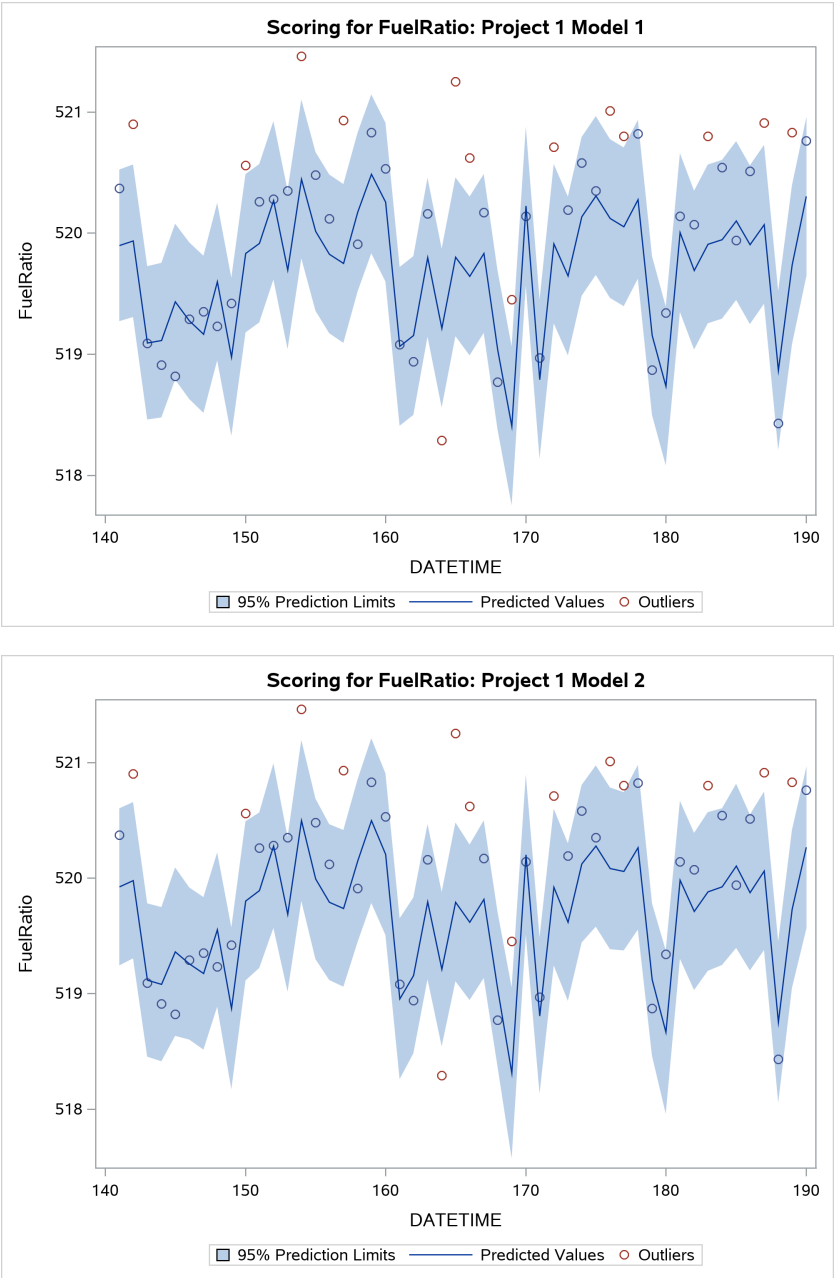
```
data mycas.project_1;
   set train(firstobs=91 obs=140);
run;

data mycas.project_1_score;
   set train(firstobs=141 obs=190);
run;
```

As you can see, the scoring data for the previous window (cycles 91–140) are used as historical data as required by the ARIMA model. (For more information, see the section "Scoring with the ARIMA Model" on page 96.) The following statements run scoring for these data:

```
proc smscore name = mycas.myproj scorerepository=mycas.scoreout;
   project id = 1 modelid = (1,2);
   output out = mycas.score;
run;
```

The scoring results for both model IDs for cycles 141–190 are shown in Figure 7.6.

Figure 7.6 Scoring for Cycles 141–190



The following statements do the scoring for the next consecutive window, cycles 191–240:

```
data mycas.project_1;
   set train(firstobs=141 obs=190);
run;

data mycas.project_1_score;
   set train(firstobs=191 obs=240);
run;

 proc smscore name = mycas.myproj scorerepository=mycas.scoreout;
    project id = 1 modelid = (1,2);
    output out = mycas.score;
 run;
```

The scoring results for both model IDs for cycles 191–240 are shown in Figure 7.7.

Figure 7.7 Scoring for Cycles 91–140

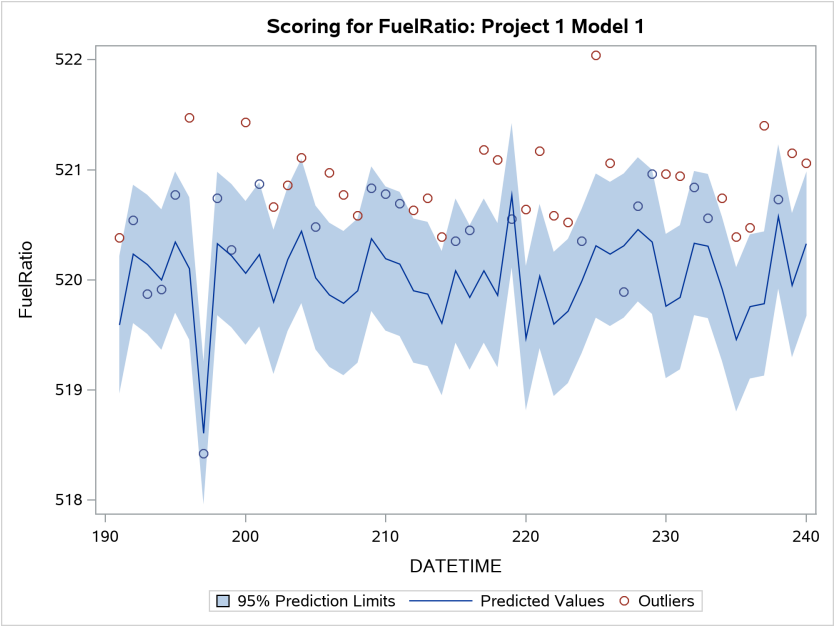


Figure 7.7 *continued*

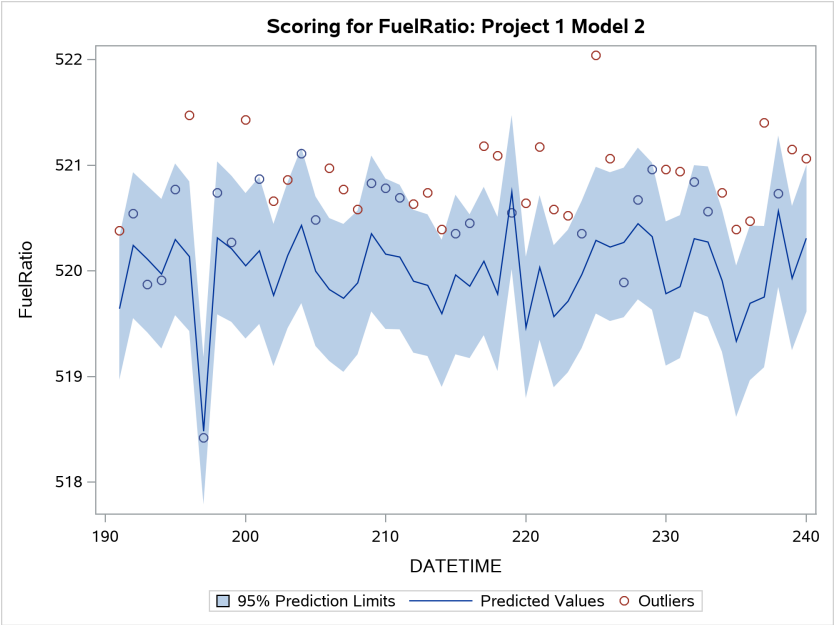


Figure 7.5 through Figure 7.7 show that the number of outliers increases from one scoring window to the next and there is a clear trend of increasing numbers of positive outliers (that is, actual values that are greater than the upper prediction limit). This trend indicates gradual structural changes in the model that manifest engine degradation. This example demonstrates how stability monitoring captures these gradual changes and signals the need for engine maintenance long before a catastrophic failure occurs.

Syntax: SMSCORE Procedure

The following statements are available in the SMSCORE procedure:

```
PROC SMSCORE NAME=CAS-libref.data-table REP=CAS-libref.data-table <options> ;
    OUTPUT OUT=CAS-libref.data-table ;
    PROJECT ID= project-ID-list <options> ;
```

None of the statements is required. The OUTPUT statement is recommended for monitoring.

PROC SMSCORE Statement

PROC SMSCORE NAME=*CAS-libref.data-table* **REP=***CAS-libref.data-table* < *options* > **;**

The PROC SMSCORE statement invokes the procedure.

You must specify the following required options:

NAME=*CAS-libref.data-table*
specifies the SAS data table (project repository) that contains the projects to be scored. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 84.

REP=*CAS-libref.data-table*
SCOREREPOSITORY=*CAS-libref.data-table*
specifies a binary scoring repository that contains information that is required for scoring for each calibrated model. This *CAS-libref.data-table* must match the name that was specified in the STORE statement in the SMCALIB procedure.

You must specify the following option if you want to use the data table of selected model IDs that PROC SMSELECT produces:

MODELS=*SAS-data-table*
specifies a SAS data table that contains a list of project IDs and model IDs for which scoring is requested. PROC SMSELECT generates this table as an output (for more information, see Chapter 8, "The SMSELECT Procedure"). You must specify either this option or the MODELID= option in the PROJECT statement. This option is ignored if the ID= option in the PROJECT statement specifies only one project ID and the MODELID= option is specified.

The *SAS-data-table* contains the following columns:

- proj_id, which contains the numeric project ID
- model_id, which contains the numeric model ID

See "Example 7.1: Scoring with Automated Model Selection" on page 97 for the use of this option.

You can also specify the following *option*:

NTHREADS=*number-of-threads*
specifies the number of threads to use in the computation. The default value is the number of CPUs available in the machine.

OUTPUT Statement

OUTPUT OUT=*CAS-libref.data-table* **;**

The OUTPUT statement generates the scoring output. You must specify the following option:

OUT=*CAS-libref.data-table*
specifies the SAS data table to contain the scoring output. The table has the following columns: project ID, model ID, time ID, actual target, predicted target, lower prediction limit, upper prediction limit, and error (computed as actual minus predicted). *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the section "Using CAS Sessions and CAS Engine Librefs" on page 84.

PROJECT Statement

PROJECT ID=(*project-ID-list* **)**< *option* > **;**

The PROJECT statement specifies which project IDs from the current project repository are to be scored. If the PROJECT statement is omitted, then the MODELS= option in the PROC SMSCORE is required to specify which project IDs and which model IDs within each project are to be scored.

You must specify the ID= option. You can specify it in any of the following forms:

ID=*project-ID*
specifies a single project ID.

ID=(*project-ID-list***)**
specifies a comma-separated list of project IDs within parentheses. If you use this form, you must specify the MODELS= option in the PROC SMSCORE statement.

ID=_ALL_
specifies all projects. If you use this form, you must specify the MODELS= option in the PROC SMSCORE statement.

A project ID can be any integer between 1 and 999,999,999.

If a single project ID is specified, then you can also specify the following *option*:

MODELID=(*model-ID-list***)**
specifies a comma-separated list of model IDs within parentheses. This option is valid only if a single project ID is specified in the ID= option; otherwise, this option is ignored and the MODELS= option in the PROC SMSCORE statement is required. If both the MODELID= option in the PROJECT statement and the MODELS= option in the PROC SMSCORE statement are specified when a single project ID is specified, then the MODELS= option in the PROC SMSCORE statement is ignored.

Details: SMSCORE Procedure

The stability monitoring calibration process stores an estimated model (also called score information) for each successfully calibrated model ID. Score information is model-dependent; it contains parameter estimates and other statistics necessary for scoring. The scoring process applies the estimated model to the explanatory variables in the scoring data to generate the predicted values of the target variable and its prediction limits. Predicted values and prediction intervals are used to monitor conditions by comparing them to the actual values of the target variables.

Select Models for Scoring

To select a model (or several models) for scoring, it is recommended that you use holdout statistics as a selection criterion (see the section "Holdout Statistics" on page 68). Holdout statistics measure how well an estimated model predicts the target variable by using the tail end (the holdout length) of the historical data, giving a better idea of how well the model will predict the target variable in the future. Holdout statistics are preferred over fit statistics in selecting models for scoring; the best-fitting model is not necessarily the best model for scoring. After one or more model IDs have been selected for scoring, you supply the corresponding model IDs during the scoring call.

Scoring and Model Type

Scoring with the REG Model

Ordinary least squares (OLS) is a memoryless model; given the estimated model information, it scores based only on the current score data. A missing explanatory variable results in a missing predicted target and missing prediction limit for the same datetime; however, it does not affect other observations.

Scoring with the ARIMA Model

In general, an ARIMA model specification includes lags of the target variable (which represents the autoregressive (AR) term) and past errors (which constitute the moving average (MA) term). Therefore, ARIMA scoring requires historical observations in addition to the current scoring data. The length of the required history depends on the estimated ARIMA specification. A built-in requirement of stability monitoring is that the history length for ARIMA scoring must be $AR + MA + N_x + 10$ observations at the end of history (either calibration data or the last score data), where AR is the autoregressive order, MA is the moving average order, and N_x is number of selected explanatory variables (including power terms). Another implication of the lag-dependent feature of ARIMA is that a single missing value of the dependent variable (either in the current scoring data or in the required history) will result in overall scoring failure for the ARIMA model.

Scoring with the Simple Model

If you want to include the simple model in scoring, then you need to explicitly specify it by including model ID=0 when you run scoring.

Examples: SMSCORE Procedure

Example 7.1: Scoring with Automated Model Selection

This example demonstrates the use of PROC SMSELECT for automated model selection before scoring. Note that you need to specify the MODELS= option in the PROC SMSCORE statement. The example uses the data from the section "Getting Started: SMSCORE Procedure" on page 85 for the first monitoring window (cycles 91–140).

```
proc smselect
        name=mycas.myproj
        input=work.holdstat;
   project id=1;
   output out=work.scoremodels;
run;

proc smscore
        name = mycas.myproj
        scorerepository=mycas.scoreout
        models=scoremodels;
   output out = mycas.score;
run;
```

PROC SMSELECT uses the holdout statistics data table work.holdstat, which was generated as ODS output from the PROC SMCALIB run, to select the best models for project 1 (as specified in the PROJECT statement). Automated model selection is performed according to the parameters that are stored in the project repository mycas.myproj, which is specified in the NAME= option. The OUTPUT statement requests that the IDs of the best models be stored in the work.scoremodels SAS data table. This table is then specified in MODELS= option in the PROC SMSCORE statement. Because the PROJECT statement is omitted, the MODELS= data set is used to determine which project IDs and model IDs are to be scored. This call is functionally identical to the first call of SMSCORE procedure in the section "Getting Started: SMSCORE Procedure" on page 85.

Example 7.2: Scoring Using a Local Analytic Store

Scoring can also be done by downloading the analytic store to a local file and using the ASTORE procedure in a SAS client.

The following code uses PROC ASTORE to download the analytic store from the CAS table mycas.scoreout (which was generated by PROC SMCALIB) to a local analytic store file *sm.sm_local*, and then use the local analytic store to score the input data. The scoring output is saved in the data table work.sm_score.

```
data p1_score;
   set mycas.project_1_score;
run;

 proc sort data=p1_score;
    by datetime;
 run;

 proc astore;
    download rstore=mycas.scoreout store="sm.sm_local";
    setoption ProjectId 1;
    setoption ModelId 2;
    score data=p1_score store="sm.sm_local" out=sm_score;
 run;
```

The SORT procedure step ensures the sequential order of the input score data by copying the input data to the Work libref and sorting them by the Datetime variable. Only one model for one project at a time can be scored by using this method, which includes the appropriate SETOPTION statements in PROC ASTORE. In this example, the scoring is requested for a model whose ID is 2 of a project whose ID is 1.

References

Saxena, A., and Goebel, K. (2008). "Turbofan Engine Degradation Simulation Data Set." Accessed January 17, 2017. NASA Ames Prognostics Data Repository. NASA Ames Research Center, Moffett Field, CA. https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/.

Chapter 8
The SMSELECT Procedure

Contents

Overview: SMSELECT Procedure

The SMSELECT procedure selects one or more models that are calibrated by the SMCALIB procedure for scoring by the SMSCORE procedure. When model selection is based on model selection criteria, this is referred to as automated model selection. The procedure also enables you to specify which models to select for scoring; this is referred to as manual model selection. PROC SMSELECT is one of five procedures in SAS Visual Forecasting that you can use to manage stability monitoring projects. The output of PROC SMSELECT is a data table that contains selected model IDs for each project, and it is used as input for scoring (also called monitoring) by PROC SMSCORE. In automated model selection, PROC SMSELECT provides a way to automate the stability monitoring process flow. For more information about stability monitoring, see the section "Stability Monitoring Overview" on page 57.

Before you can use PROC SMSELECT to run automated model selection for a project, you need to run calibration for this project by using PROC SMCALIB (see Chapter 5, "The SMCALIB Procedure"). For manual model selection, the minimum requirement is to create a project repository by using the SMPROJECT procedure (see Chapter 6, "The SMPROJECT Procedure").

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 7 in Chapter 3, "Shared Concepts."

Getting Started: SMSELECT Procedure

This example demonstrates the use of stability monitoring procedures, including the SMSELECT procedure, on turbofan engine data. For more information about the data, see the section "Getting Started: SMCALIB Procedure" on page 59. The following DATA step creates the data set train:

```
data train;
   input DateTime InletPressure TotPressure FuelRatio
         CoreSpeed Enthalpy HPCBleed LPCBleed;
   datalines;
1 14.62 21.59 519.25 8116.67 390 38.78 23.2995
2 14.62 21.59 520.60 8124.89 391 39.12 23.3948
3 14.62 21.59 519.26 8112.68 390 38.89 23.3420
4 14.62 21.58 518.61 8119.49 389 38.83 23.2713

   ... more lines ...
```

The data for this engine contain 494 observations (cycles). The first 90 cycles are used as the training period (for calibration using PROC SMCALIB), and the three consecutive 50-cycle windows starting from cycle 91 are used for condition monitoring (see the section "Getting Started: SMSCORE Procedure" on page 85). The following DATA step loads the training data into a CAS session. This DATA step assumes that the CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.project_1;
   set train(obs=90);
   stable=1;
run;
```

As you can see from the DATA step, stable periods for the training data need to be defined. Any observations that are marked as unstable are removed from the training data.

Next, you need to create a model specification by using the SMSPEC procedure:

```
proc smspec;
   input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
   reg name=mycas.reg1;
   arima name=mycas.arima1;
run;
```

The previous statements request both OLS regression and ARIMA model specifications (stored as mycas.reg1 and mycas.arima1, respectively). Both model specifications use all six explanatory variables, which are specified in the INPUT statement. Because the INPUT statement does not include a TRANSFORM= option or a REQUIRED= option, input variables are not transformed and none of them are required. The target variable does not need to be transformed; therefore, the TARGET statement is omitted.

Next, you need to create a project repository by using the SMPROJECT procedure:

```
proc smproject name=mycas.myproj;
    target FuelRatio;
    input InletPressure TotPressure CoreSpeed Enthalpy HPCBleed LPCBleed;
    model reg1 arima1;
run;
```

This PROC SMPROJECT code is run using mostly default values. It generates a project repository named mycas.myproj (specified in the NAME= option) that contains a single project whose ID is 1 by default (because no ID= option is specified in the PROC SMPROJECT statement). The project expects the training data table to be named mycas.project_1 (because the DATA= option is not specified) and the scoring data table to be named mycas.project_1_score (because the SCOREDATA= option is not specified). The ALPHA= and HOLDOUT= options are omitted; therefore, the significance level is set to 0.05, and the holdout length is set to 10 observations. The project uses FuelRatio as a target variable (as specified in the TARGET statement), and it has six explanatory variables (specified in the INPUT statement). The DATETIME and STABLE statements are omitted; therefore, the datetime variable is assumed to be datetime, and the stable flag variable is assumed to be stable. The project uses two model specifications, mycas.reg1 and mycas.arima1.

Next, you run the project calibration by using the SMCALIB procedure:

```
ods output holdoutstats=holdstat;

proc smcalib name = mycas.myproj;
   output modelfit   = mycas.mfit
          holdoutfit = mycas.hfit
          scoreinfo  = mycas.sinfo;
   store mycas.scoreout;
run;

ods output close;
```

The PROC SMCALIB statement looks for the mycas.myproj project repository. Because the PROJECT statement is omitted, all projects in that repository are calibrated. In this example, the repository contains only one project, whose ID is 1. The OUTPUT statement requests that SAS output tables be generated for model fit, historical fit, and scoring information. The STORE statement requests that information for each model necessary for scoring be stored in the binary file mycas.scoreout. The following ODS tables are also produced: calibration status, fit statistics, holdout statistics, and parameter estimates. The ODS OUTPUT requests that holdout statistics table be saved as the data table work.holdstat.

The calibration status table (Figure 8.1) indicates whether the calibration was successful for each model.

Figure 8.1 Calibration Status

Calibration Status

Project ID	Model ID	Model Name	Status	RC	Message
1	**0**	Simple	0	0	Command successfully completed.
	1	arima1	0	0	Command successfully completed.
	2	reg1	0	0	Command successfully completed.

As you can see, three models were successfully calibrated. Two of the models were defined by PROC SMSPEC, and the model whose ID is 0 is an internal model that uses only the target variable (for more information, see the section "Simple Model" on page 68). Each model is given a numeric ID; an ID of 0 always indicates an internal simple model and user models are numbered consecutively, starting from 1. Model IDs are used later for scoring.

Each model's potential for scoring is evaluated by holdout analysis, which is done as part of the model calibration process. By default, the last 10 observations are used for the holdout analysis, and the significance

level is set to 0.05. To request different values, you can specify the ALPHA= and HOLDOUT= options in the PROC SMPROJECT statement. The holdout analysis results are shown in Figure 8.2. Holdout statistics are slightly better for the ARIMA model, and both the ARIMA and REG models significantly outperform the default (simple) model. **NOTE:** Calibration also produces a holdout forecast graph for each successfully calibrated model for each project; the graphs are not shown here.

Figure 8.2 Holdout Statistics

Holdout Statistics				
Project ID	Model ID	Model Name	MAPE	RMSE
1	0	Simple	0.1185	0.7044
	1	arima1	0.0589	0.3514
	2	reg1	0.0593	0.3610

As the next step, you can select the best-performing models by using PROC SMSELECT:

```
proc smselect name=mycas.myproj input=work.holdstat;
   project id=1 select=RMSE best=2;
   output out=work.mymodels;
run;
```

The PROC SMSELECT statement looks for the mycas.myproj project repository that is specified in the NAME= option. It selects the best models according to holdout statistics for each model from a data table listed in the INPUT= option. Model selection criteria are listed in the PROJECT statement. The ID= option requests that the model selection be performed for a single project whose ID is 1. The BEST= option requests that the two best models be selected according to the RMSE criterion (SELECT= option). The OUTPUT statement requests that the IDs of the best models be stored in the work.mymodels SAS data set. As you can see from Output 8.3, the model IDs 1 and 2 are selected for project 1.

Figure 8.3 Selected Models

proj_id	model_id
1	1
1	2

Scoring (also called monitoring) is performed next. Scoring data for the first 50-cycle window (starting from cycle 91) are created in the following DATA step:

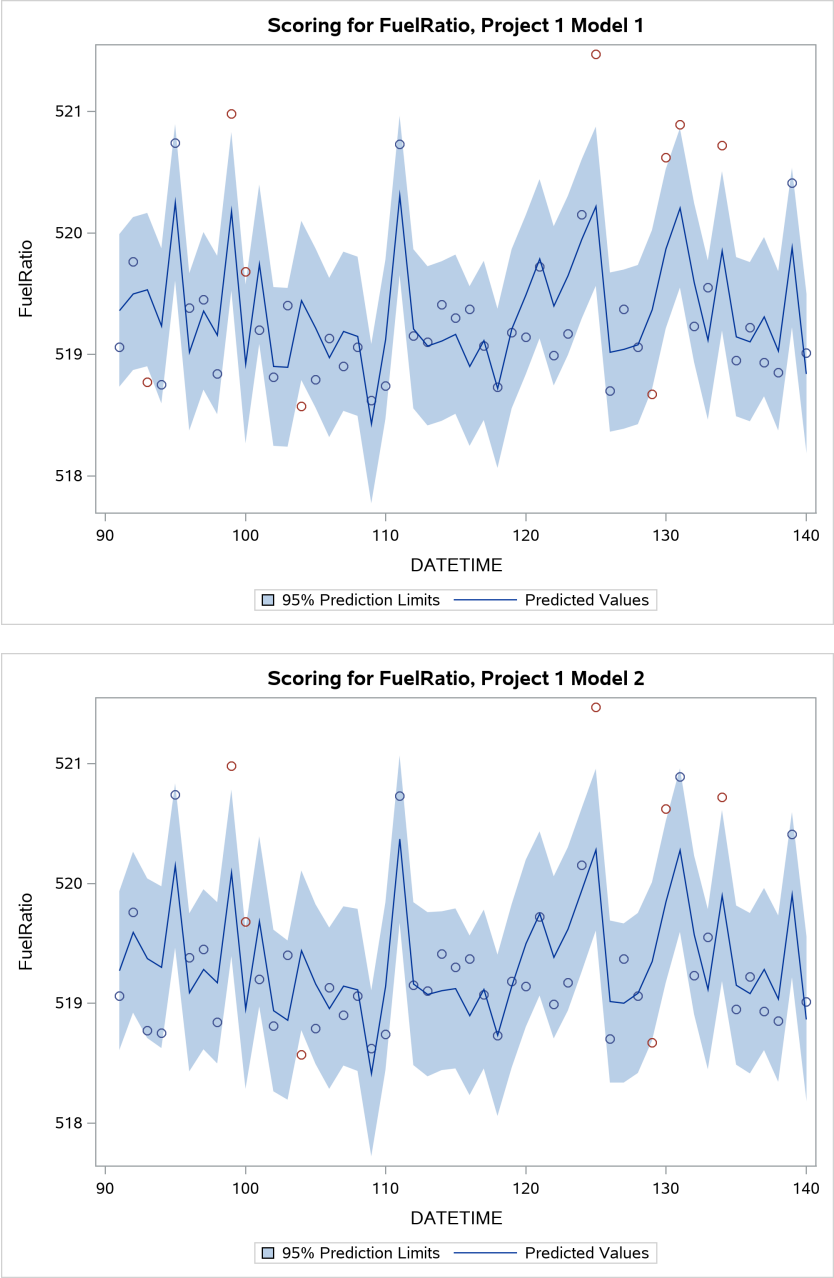
```
data mycas.project_1_score;
   set train(firstobs=91 obs=140);
run;
```

The following statements score the data from this window:

```
proc smscore
        name = mycas.myproj
        scorerepository=mycas.scoreout
        models=mymodels;
   output out = mycas.score;
run;
```

The PROC SMSCORE statement looks for the project repository mycas.myproj (as specified in the required NAME= option) and for the binary file mycas.scoreout (which contains information necessary for scoring and is specified in the required SCOREREPOSITORY= option). The IDs of the models to be scored are stored in the data table work.mymodels (as specified in the MODELS= option). Scoring produces predicted values of the target variable, with prediction limits for each model within the project. The OUT= option in the OUTPUT statement stores the output in the mycas.score data table. The scoring results for both models for cycles 91–140 are shown in Figure 8.4.

Figure 8.4 Scoring for Cycles 91–140



Syntax: SMSELECT Procedure

The following statements are available in the SMSELECT procedure:

PROC SMSELECT NAME=*CAS-libref.data-table* < *options* > **;**
  OUTPUT OUT=*SAS-data-table* **;**
  PROJECT ID=*project-ID-list* < *options* > **;**

None of the statements are required. If you want to store the selected models to be used later for scoring, then include the OUTPUT statement.

PROC SMSELECT Statement

PROC SMSELECT NAME=*CAS-libref.data-table* < *options* > **;**

The PROC SMSELECT statement invokes the procedure.

You must specify the following required option:

NAME=*CAS-libref.data-table*
specifies the SAS data table (project repository) that contains calibrated projects for which the model selection is to be performed. *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 section "Using CAS Sessions and CAS Engine Librefs" on page 100.

The following option is needed if you want the procedure to perform model selection:

INPUT=*SAS-data-table*
specifies the SAS data table that contains holdout statistics for all calibrated models. The procedure uses the statistics in this table to perform model selection according to the selection criteria that you specify in the PROJECT statement. You can omit this option if you want to manually specify the selected models in the PROJECT statement.

OUTPUT Statement

OUTPUT OUT=*SAS-data-table* **;**

The OUTPUT statement generates the output for the selected models. You must specify the following option:

OUT=*SAS-data-table*
specifies the SAS data table to contain the output for the selected models. This table can be used later for scoring in the SMSCORE procedure (for more information, see "MODELS=*SAS-data-table*" on page 94). The data table has the following columns:

- proj_id, which contains the numeric project ID
- model_id, which contains the numeric model ID

PROJECT Statement

PROJECT ID=*project-ID* < *options* > **;**

PROJECT ID=(*project-ID-list***)**< *options* > **;**

PROJECT ID=_ALL_ < *options* > **;**

The PROJECT statement specifies the subset of the projects from the current project repository for which model selection is to be performed. If you omit this statement, then model selection is performed for all projects that are listed in the INPUT= data table of the PROC SMSELECT statement.

You can specify multiple PROJECT statements to allow different model selection criteria to be applied to different projects.

You must specify the ID= option. You can specify it in any of the following forms:

ID=*project-ID*	specifies a single project ID.
ID=(*project-ID-list***)**	specifies a comma-separated list of project IDs within parentheses.
ID=_ALL_	specifies all projects.

A project ID can be any integer between 1 and 999,999,999.

You can also specify the following options:

BEST=*number*
specifies how many best models to select for each of the projects listed in the ID= option. If you omit this option, then the procedure uses a *number* that is stored in the project repository. This *number* is specified in the BEST= option of the PROC SMPROJECT statement. This option is ignored if the models to be selected are specified manually in the MODELID= option.

MODELID=(*model-ID-list* **)**
specifies the IDs for the models to be selected. This option enables you to override the model selection process of PROC SMSELECT; instead, you manually specify which models to select. If this option is used, then the BEST= and SELECT= options are ignored and the INPUT= option of the PROC SMSELECT statement is not needed.

SELECT=MAPE | RMSE
specifies the model selection criterion: mean absolute percentage error (MAPE) or root mean square error (RMSE). If you omit this option, then the procedure uses the model selection criterion that is stored in the project repository; this criterion is specified in the SELECT= option of the PROC SMPROJECT statement. This option is ignored if the models to be selected are specified manually in the MODELID= option.

Details: SMSELECT Procedure

Automated Model Selection

The SMSELECT procedure performs automated model selection when you specify a valid holdout statistics table in the INPUT= option of the PROC SMSELECT statement. The procedure then selects one or more best models for each project that is listed in the PROJECT statements. The model selection is based on the BEST= and SELECT= options that are specified in the PROC SMPROJECT statement (see the section "PROC SMPROJECT Statement" on page 76). For example, if BEST=2 and SELECT=RMSE, then the SMSELECT procedure selects up to two best models according to the root mean square error statistics from the INPUT= holdout statistics table. Either of these options can be overridden by the same option specified in the PROJECT statement of PROC SMSELECT. If you omit the PROJECT statement, then the procedure performs automated model selection for all projects that are listed in the NAME= project repository and that have results for at least one successfully calibrated model in the INPUT= holdout statistics table. **NOTE:** If both the MODELID= option is specified in the PROJECT statement and the INPUT= option is specified in the PROC SMSELECT statement, then the latter option is ignored, and manual model selection is performed (see the section "Manual Model Selection" on page 107).

Manual Model Selection

Instead of automated model selection, you can request that the models be scored manually by using the MODELID= option in the PROJECT statement. You can specify one or more comma-separated model IDs. You can specify a different model ID list for different projects by providing several PROJECT statements. Or you can specify the same model ID list for multiple projects by providing a comma-separated list of project IDs in a single PROJECT statement.

Examples: SMSELECT Procedure

Example 8.1: Automated Model Selection for All Projects

This example shows how to perform automated model selection for all projects in a repository. To do this, you can omit the PROJECT statement as follows:

```
proc smselect name=mycas.myproj input=work.holdstat;
   output out=work.mymodels;
run;
```

The SMSELECT procedure performs automated model selection for all successfully calibrated projects that are stored in the mycas.myproj project repository. The procedure uses holdout statistics from the work.holdstat data table to select one or more models according to the values of the BEST= and SELECT= options that you specify in the PROC SMPROJECT statement. You achieve the same result when you specify

the PROJECT statement with the ID=_ALL_ option. The selected model IDs for each project are stored in the work.mymodels output data table.

Example 8.2: Manual Model Selection

This example shows how to perform manual model selection. You can manually select the IDs of models to score by using the MODELID= option in the PROJECT statement as follows:

```
proc smselect name=mycas.myproj;
   project id=_ALL_ modelid=(1,2);
   output out=work.mymodels;
run;
```

The SMSELECT procedure creates the work.mymodels output data table by using two model IDs, 1 and 2, for each project from the mycas.myproj project repository.

Example 8.3: Multiple PROJECT Statements

This example shows how to select models for more than one project by using different selection criteria for each project that override the corresponding selection criteria from the SMPROJECT procedure. You can do this in a single call of the SMSELECT procedure by specifying multiple PROJECT statements as follows:

```
proc smselect name=mycas.myproj  input=work.holdstat;
   project id=1 best=1 select=MAPE;
   project id=500 best=2 select=RMSE;
   output out=work.mymodels;
run;
```

The SMSELECT procedure performs automated model selection for projects 1 and 500 that are stored in the mycas.myproj project repository. For project 1, it selects one best model by using the mean absolute percentage error (MAPE) criterion. For project 500, it selects up to two best models by using the root mean square error (RMSE) criterion. The selected model IDs for each project are stored in the work.mymodels output data table.

Chapter 9
The SMSPEC Procedure

Contents

Overview: SMSPEC Procedure

The SMSPEC procedure generates one or two model specifications to be used in a stability monitoring (SM) project. PROC SMSPEC is one of five procedures in SAS Visual Forecasting that you can use to manage stability monitoring projects. For more information about stability monitoring, see the section "Stability Monitoring Overview" on page 57.

A model specification contains a full description of a model, which includes the following:

- model type
- list of explanatory variables with each variable's transformation type and whether it is required
- a target variable and its transformation type
- optional parameters such as significance level and holdout length

The SMSPEC procedure saves each model specification as a SAS data set whose name you specify. You can use that name later in the SMPROJECT procedure when you specify a model to use in an SM project. One call of the SMSPEC procedure can generate either or both of the following model specifications: OLS regression (called REG in this and other stability monitoring documentation) and ARIMA.

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 7 in Chapter 3, "Shared Concepts."

Getting Started: SMSPEC Procedure

The following code illustrates the use of the SMSPEC procedure to generate a REG model specification and an ARIMA model specification:

```
libname a ".";
proc smspec;
   input FanInletPressure BypassTotPressure Enthalpy
         HPTCoolantBleed LPTCoolantBleed;
   input CoreSpeed / required=yes;
   reg name = a.reg1;
   arima name = a.arima1;
run;
```

Two INPUT statements are specified because the explanatory variables need to be treated differently. The first INPUT statement specifies five input variables that are optional, and the second INPUT statement specifies one input variable, CoreSpeed, that is required. Because no TRANSFORM= option is specified in either INPUT statement, no transformation is requested for input variables by default. Because no transformation is needed for the target variable, the TARGET statement is omitted. The procedure call generates two model specifications. The REG statement requests a model specification for OLS regression; the NAME= option in that statement names the data set (a.reg1) in which to store the model specification, and because at least one INPUT statement is specified, PROC SMSPEC also uses that name for an auxiliary SAS data set (a.reg1_aux) in which to store necessary information about the input variables. Similarly, the ARIMA statement requests a model specification for ARIMA; the NAME= option in that statement names the data set (a.arima1) in which to store the model specification, and because at least one INPUT statement is specified, PROC SMSPEC also uses that name for an auxiliary SAS data set (a.arima1_aux) in which to store necessary information about the input variables.

Syntax: SMSPEC Procedure

The following statements are available in the SMSPEC procedure:

PROC SMSPEC ;
  ARIMA NAME=*CAS-libref.data-table* < *options* > **;**
  INPUT *variable-list* < / *options* > **;**
  REG NAME=*CAS-libref.data-table* < *options* > **;**
  TARGET TRANSFORM=NONE | LOG | LOGM ;

The PROC SMSPEC statement and at least one of the REG or ARIMA statements are required. If both a REG statement and an ARIMA statement are specified, then two model specifications are generated. You are not required to specify a CAS engine libref in order to store a model specification data set. However, during model calibration (by the SMCALIB procedure), all model specification data sets and the project repository data set need to be located in a properly defined CAS engine libref. Multiple INPUT statements are allowed.

PROC SMSPEC Statement

PROC SMSPEC ;

The PROC SMSPEC statement invokes the procedure. This statement has no options.

ARIMA Statement

ARIMA NAME=*CAS-libref.data-table* < *options* > **;**

The ARIMA statement requests an ARIMA model specification.

You must specify the following required option:

NAME=*CAS-libref.data-table*

N=*CAS-libref.data-table*

generates a data table, named *data-table*, in the specified caslib to contain the model specification. If any input variables are specified in an INPUT statement, PROC SMSPEC also generates a second SAS data table in the same caslib, whose name is *data-table* with the suffix _AUX. This second data table contains information about the input variables, such as their transformation type and whether they are required.

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 section "Using CAS Sessions and CAS Engine Librefs" on page 110.

You can also specify the following *options*:

ALPHA=*significance-level*

A=*significance-level*

specifies the significance level for computing confidence limits for historical fit and holdout forecast. This option can be overwritten by the SMPROJECT procedure. If you omit this option, then no model-specific significance level is stored.

HOLDOUT=*holdout-length*

H=*holdout-length*

specifies the length of holdout—that is, the number of observations at the end of a training data set to be used for holdout analysis as a part of the model calibration process. This option can be overwritten by the SMPROJECT procedure. If you omit this option, then no model-specific holdout length is stored.

P=*Pmin***:***Pmax*

specifies the range of values of autoregressive (AR) orders for ARIMA specification, where *Pmin* is an integer greater than or equal to 0 and *Pmax* is an integer greater than or equal to *Pmin*. By default, P=0:5.

Q=*Qmin***:***Qmax*

specifies the range of values of moving average (MA) orders for ARIMA specification, where *Qmin* is an integer greater than or equal to 0 and *Qmax* is an integer greater than or equal to *Qmin*. By default, Q=0:5.

INPUT Statement

INPUT *variable-list* < / *options* > **;**

The INPUT statement specifies the list of explanatory variables for the model. You can specify multiple INPUT statements so that you can group explanatory variables that have the same transformation type and required status into one statement. If a model has no explanatory variables, then you can omit the INPUT statement.

All explanatory variables in the *variable-list* must be included in the INPUT statement of the SMPROJECT procedure.

You can specify the following *options*:

REQUIRED=YES | NO

REQ=YES | NO

specifies whether all the variables in the *variable-list* are required. You can specify the following values:

YES requires all the variables in the *variable-list*. If the value of the TRANSFORM= option is SQUARE or CUBE, then only a linear transform is required for modeling, and the higher power transforms are selected (or not) by the calibration process on the basis of their significance.

NO automatically selects variables as a part of calibration process. Variable selection uses backward selection with Akaike's information criterion as the criterion for model fit and a p-value threshold of 0.05 for variable significance.

By default, REQUIRED=NO.

TRANSFORM=NONE | LOG | SQUARE | CUBE | LOGM

TSF=NONE | LOG | SQUARE | CUBE | LOGM

specifies the transformation type for all variables in the *variable-list*. You can specify the following values:

CUBE specifies cube transformation. During model calibration, the SMCALIB procedure generates three variables for each variable in the *variable-list*. That is, for variable X_1, PROC SMCALIB generates the variables X_1, X_1^2, and X_1^3.

LOG specifies the natural log transformation: $\ln X$.

LOGM specifies the symmetric log transformation: $\operatorname{logm} X = \operatorname{sign} X \ln(1 + \operatorname{abs} X)$. This transformation type is useful if the input variable can take negative values because $\operatorname{logm}(-X) = -\operatorname{logm} X$.

NONE specifies no transformation.

SQUARE | SQ specifies the square transformation. During model calibration, the SMCALIB procedure generates two variables for each variable in the *variable-list*. That is, for variable X_1, PROC SMCALIB generates the variables X_1 and X_1^2.

LOG specifies the natural log transformation: $\ln X$.

By default, TRANSFORM=NONE.

REG Statement

REG NAME=*CAS-libref.data-table* < *options* > **;**

The REG statement requests an OLS regression model specification.

You must specify the following required option:

NAME=*CAS-libref.data-table*

N=*CAS-libref.data-table*

generates a data table, named *data-table*, in the specified caslib to contain the model specification. If any input variables are specified in an INPUT statement, PROC SMSPEC also generates a second SAS data table in the same caslib, whose name is *data-table* with the suffix _AUX. This second data table contains information about the input variables, such as their transformation type and whether they are required.

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 section "Using CAS Sessions and CAS Engine Librefs" on page 110.

You can also specify the following *options*:

ALPHA=*significance-level*

A=*significance-level*

specifies the significance level for computing confidence limits for historical fit and holdout forecast. This option can be overwritten by the SMPROJECT procedure. If you omit this option, then no model-specific significance level is stored.

HOLDOUT=*holdout-length*

H=*holdout-length*

specifies the length of holdout—that is, the number of observations at the end of a training data set to be used for holdout analysis as a part of the model calibration process. This option can be overwritten by the SMPROJECT procedure. If you omit this option, then no model-specific holdout length is stored.

TARGET Statement

TARGET TRANSFORM=NONE | LOG | LOGM ;

The TARGET statement specifies the transformation type for the target variable in the model. If you do not need to transform the target variable, you can omit the TARGET statement altogether.

NOTE: The name of the target variable is defined in the SMPROJECT procedure. There can be only one target variable per project.

You must specify the following required option:

TRANSFORM=NONE | LOG | LOGM

TSF=NONE | LOG | LOGM

specifies the transformation type:

LOG	specifies a natural log transformation: $\ln \mathsf{X}$.
LOGM	specifies a symmetric log transformation: $\operatorname{logm} \mathsf{X} = \operatorname{sign} \mathsf{X} \ln(1 + \operatorname{abs} \mathsf{X})$. This transformation type is useful if the target variable can take negative values because $\operatorname{logm}(-\mathsf{X}) = -\operatorname{logm} \mathsf{X}$.
NONE	specifies no transformation

By default, TRANSFORM=NONE.

Details: SMSPEC Procedure

REG Model

An ordinary least squares (OLS) model is referred to as a REG model throughout this chapter. This model assumes an OLS relationship between the target and explanatory variables. Standard OLS assumptions are applied: strict exogeneity, spherical errors (homoscedasticity and no autocorrelation), linearly independent regressors, and independently and identically distributed (iid) observations.

ARIMA Model

An ARIMA (autoregressive integrated moving average) model predicts the value of the target variable as a linear combination of its own past values, past errors, and current values of explanatory variables. The ARIMA model was first introduced by Box and Jenkins (1976). The following restrictions are imposed on an ARIMA model that is used in stability monitoring:

- All variables are assumed to be stationary. It is assumed that all variables have finite variance, so no differencing of the variables is applied during modeling.
- Only the contemporaneous effect of explanatory variables on the target variable is estimated.
- Autoregressive (AR) and moving average (MA) orders are selected automatically by the solution during the calibration process. For any model ID that has specified an ARIMA type, all ARIMA models with AR orders from 0 to 5 and MA orders from 0 to 5 are considered for calibration. That is, 36 possible ARIMA model combinations are created: ARMA(0,0), ARMA(0,1), ARMA(0,2), . . . , ARMA(5,5). The best model is selected by the AICC model fit criterion.

Examples: SMSPEC Procedure

Example 9.1: Creating Model Specifications

The following code demonstrates various options of the SMSPEC procedure:

```
libname a ".";

proc smspec;
   target transform = log;
   input FanInletPressure BypassTotPressure Enthalpy
         HPTCoolantBleed LPTCoolantBleed / transform = log;
   input CoreSpeed / transform = log required=yes;
   reg name = a.reg_log alpha=0.1 holdout=15;
   arima name = a.arima_log alpha=.15 holdout=20;
run;

proc smspec;
   target transform = logm;
   input FanInletPressure BypassTotPressure Enthalpy
         HPTCoolantBleed LPTCoolantBleed CoreSpeed
         / transform = logm required=yes;
   reg name = a.reg_logm;
run;

proc smspec;
   target tsf = log;
   arima name = a.arima_log_no_x;
```

```
run;
```

The first PROC SMSPEC call creates two model specifications: the REG statement requests an OLS regression model specification named a.reg_log, and the ARIMA statement requests an ARIMA model specification named a.arima_log. The TRANSFORM=LOG option in the TARGET statement and both INPUT statements requests that the target variable and all input variables be log-transformed. Only one input variable (CoreSpeed) is required, as specified in the REQUIRED=YES option in the second INPUT statement. In the REG statement, the ALPHA=0.1 option requests a significance level of 0.1 and the HOLDOUT=15 option specifies 15 observations for holdout analysis. In the ARIMA statement, the ALPHA=0.15 option requests a significance level of 0.15 and the HOLDOUT=20 option specifies 20 observations for holdout analysis.

NOTE: For proper model comparison after a project calibration, it is recommended that you request the same number of holdout observations for all models in one project. This can be enforced in the SMPROJECT procedure.

The REG statement in the second PROC SMSPEC call generates an OLS regression model specification and stores the model specification in a SAS data set named a.reg_logm. All the variables in this model specification are symmetric-log-transformed because the TRANSFORM=LOGM option in specified in both the TARGET statement and the INPUT statement.

The ARIMA statement in the third PROC SMSPEC call generates an ARIMA model specification and stores the model specification in a SAS data set named a.arima_log_no_x. The model specification has no input variables (because no INPUT statements are specified) and a log-transformed target variable (because the TSF=LOG option is specified in the TARGET statement).

References

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

Chapter 10
The TSINFO Procedure

Contents

Overview: TSINFO Procedure

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

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

PROC TSINFO Compared with the TIMEID Procedure

The TSINFO procedure is the next generation of the TIMEID procedure (a SAS/ETS procedure) for time series information analysis. PROC TSINFO was developed specifically for SAS Viya. The syntax of PROC TSINFO is similar to the syntax of PROC TIMEID, from which it borrows the underlying methodology and goals. PROC TSINFO is designed to run on a cluster of machines that distribute the data and computations to multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Services (CAS) server.

Table 10.1 shows the differences between the TSINFO and TIMEID procedures.

Table 10.1 PROC TSINFO versus PROC TIMEID

Feature	TSINFO	TIMEID
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable bottom-up

Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

```
cas mysess terminate;
```

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

Getting Started: TSINFO Procedure

When a data table contains a time ID variable that has corrupted, missing, or duplicate values, PROC TSINFO can help isolate and identify these problematic observations. For larger data tables whose quality is unknown, it can be useful to get a general overview of the relative number of observations that have problematic time ID values. When prior knowledge of the time interval that separates observations is incomplete, PROC TSINFO can be used to infer the interval.

The following example uses the Sashelp.Air data set to illustrate how to analyze some time series information.

The following DATA step loads the Air data set from the Sashelp directory to a table named mycas.air in your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
   set sashelp.air;
do class=1 to 10;
   output;
end;
run;
```

The following statements use the TSINFO procedure to infer the interval of the time series:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval1
            nthreads=2;
   id date align=BEGIN;
   by class;
run;
```

PROC TSINFO requires the following inputs:

- The input table, specified in the DATA= option in the PROC statement
- BY variables (class), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSINFO Procedure

The following statements are available in the TSINFO procedure:

```
PROC TSINFO <options> ;
    BY variables ;
    ID variable <options> ;
```

The PROC TSINFO and ID statements are required. The following sections describe the PROC TSINFO statement and then describe the other statements in alphabetical order.

Functional Summary

Table 10.2 summarizes the statements and options that are used with the TSINFO procedure.

Table 10.2 PROC TSINFO Functional Summary

Description	Statement	Option
Data Options		
Specifies the input data table	PROC TSINFO	DATA=
Specifies the detailed output interval data table	PROC TSINFO	OUTINTERVALDETAILS=
Specifies the number of threads to be used in the procedure	PROC TSINFO	NTHREADS=
Time ID Options		
Specifies the alignment of time ID values	ID	ALIGN=
Specifies the ending time ID value	ID	END=
Specifies the starting time ID value	ID	START=

PROC TSINFO Statement

```
PROC TSINFO options ;
```

The PROC TSINFO statement invokes the TSINFO procedure.

You must specify the following data tables:

DATA=*CAS-libref.data-table*

specifies the SAS input data table that contains the input data for the procedure. If the DATA= option is not specified, the most recently created SAS data set is used. *CAS-libref.data-set* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

OUTINTERVALDETAILS =*CAS-libref.data-table*
names the output data table to contain the time ID interval information for each BY group in the DATA= data set. For more information, see the section "OUTINTERVALDETAILS= Data Set" on page 125. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

You can also specify the following *option*:

NTHREADS=*number*
specifies the number of threads to use in the computation.

By default, NTHREADS=1.

BY Statement

BY *variables* **;**

You can use a BY statement to obtain separate analyses for groups of observations that are defined by the *variables*, which can be either character or numeric.

When a BY statement is included, the procedure expects the input data to be sorted in the order of the BY variables.

ID Statement

ID *variable* < /*options* > **;**

The ID statement names a numeric *variable* that identifies observations in the input and output data. The ID variable values are assumed to be SAS date or datetime values.

You can specify the following *options* to determine how to space the time ID values and align them relative to a SAS date or datetime interval:

ALIGN=BEGINNING | MIDDLE | ENDING | INFER
specifies the alignment of the identifying SAS date or datetime that is used to represent intervals. The value of the ALIGN= option is used in the analysis of the time ID variable. You can specify the following values:

BEGINNING | BEG | B	uses the beginning date in the interval as the identifying date for the interval.
MIDDLE | MID | M	uses the middle date in the interval as the identifying date for the interval.
ENDING | END | E	uses the end date in the interval as the identifying date for the interval.
INFER	infers that the alignment of values within time intervals from the time ID values.

By default, ALIGN=BEGIN.

END=*value*
specifies a SAS date, datetime, or time value that represents when the SAS date or datetime values end. If the largest *variable* value is less than *value*, then this option has no effect.

START=*value*
specifies a SAS date, datetime, or time value that represents the *variable* value at which the SAS date or datetime values begins.

Details: TSINFO Procedure

Time ID Diagnostics

For a specified time interval, PROC TSINFO decomposes the raw time ID values in an input data table into the following three quantities, whose values are represented by nonnegative integers at each unique time ID value in the input series:

intervalcounts
the number of observations that share each time interval in the data set.

offsets
the numerical difference between a time ID value and the aligned value for that time interval. The unit of measure that is used to express this distance is days for date values and seconds for datetime values. The offset is computed for each time ID value, t_i, by using the following SAS expression:

$$\text{offset}_i = t_i - \text{INTNX}(\text{interval}, t_i, 0, \text{alignment})$$

spans
the number of intervals between each time ID value and the previous time ID value. The spans value is equivalent to the number returned by the following SAS expression:

$$\text{spans}_i = \text{INTCK}(\text{interval}, t_{i-1}, t_i)$$

Inferring Time Intervals and Alignments

A time interval is inferred from the time ID values in the input data. The technique that is used to infer a time interval involves searching for the interval that fits the greatest number of time ID values. First, time ID values are sampled from the input data to generate a set of candidate intervals. Then the candidate interval that is consistent with greatest number of time ID values is chosen to represent the time series.

When the ALIGN=INFER option is specified, the convention that is used to specify time interval alignment is inferred from the time ID variable values by using a similar technique. When both the time interval and its alignment are to be inferred, each of the possible alignments (BEGIN, MIDDLE, and END) is considered in the search. Precedence in the search is given to intervals that use the BEGIN alignment.

Data Table Output

The TSINFO procedure creates an ODS output data table. This data table contain the variables that are specified in the BY statement along with variables that characterize the time ID values. The information in this data table summarizes time ID diagnostic information across all BY groups in the table that is specified in the DATA= option.

OUTINTERVALDETAILS= Data Set

The OUTINTERVALDETAILS= data set contains statistics about the time interval that is specified in the ID statement or inferred from the time ID values for each BY group. The following variables represent these statistics:

TIMEID	time ID variable name
START	starting time ID interval
END	ending time ID interval
INTERVAL	time interval that was specified or is recommended
INTNAME	time interval base name that was specified or is recommended
MULTIPLIER	time interval multiplier that was specified or is recommended
SHIFT_INDEX	time interval shift index that was specified or is recommended
ALIGNMENT	time interval alignment that was specified or is recommended
SEASONALITY	seasonality determined from the specified or recommended time interval
FORMAT	format of the time ID variable

The START and END variables are reported using the interval and alignment values that are either specified in the ID statement or inferred from the time ID values.

Printed Tabular Output

The TSINFO procedure produces printed output by using the Output Delivery System (ODS). By default, the procedure produces the overall time series information table.

Table 10.3 ODS Tables Produced in PROC TSINFO

ODS Name	Description
DataSet	Information about the input data table
Decomposition	Time ID counts, offsets, and spans
Interval	Information about the time interval
IntervalCountsComponent	Frequency distribution of interval counts
Align	Time ID alignment information

Table 10.3 *continued*

ODS Table Name	Description
Seasonality	Summary of the number of valid observations

Examples: TSINFO Procedure

Example 10.1: Examining a Weekly Time ID Variable and Inferring a Date Interval

This example illustrates how a time ID variable can be inferred from data when a sufficient number of observations are present. The following DATA step loads the Air data set onto the CAS server. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
   set sashelp.air;
run;
```

The following TSINFO procedure statements generate an ODS display of the time series that characterizes interval counts, alignment, and seasonality in the time ID variable:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;
  id date align=B;
run;
```

There are 144 observations in the mycas.Air data table. The 144 observations are enough to determine that the Date time ID variable is represented by the MONTH interval.

Example 10.2: Examining Multiple BY Groups

This example illustrates how a time ID variable can be examined independently over each BY group and summarized over all observations in the data table that is specified in the DATA= option.

```
data mycas.air(promote=yes);
set sashelp.air;
do class=1 to 10;
   output;
end;
run;
```

The following TSINFO procedure statements generate data tables that summarize data that have four BY groups:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;
  id date align=B;
  by class;
run;
```

The summarized information shows that BY groups 1 to 10 in the mycas.interval2 data table contain the interval fit information of the time ID values and shows the number of counts, alignment, and seasonality for each by group. All BY groups conform exactly to the MONTH interval. The information can be useful in diagnosing time series that contain many time intervals.

Chapter 11
The TSMODEL Procedure

Contents

Overview: TSMODEL Procedure

The TSMODEL procedure is a SAS Viya procedure that executes user-defined programs on time series data. The TSMODEL procedure analyzes timestamped transactional data with respect to time and accumulates the data into a time series format.

The TSMODEL procedure forms time series from input timestamped transactional data and writes the accumulated time series variables to an output table. Time series are delineated based on the distinct values of the variables that are listed in the BY statement.

Timestamped transactional data are recorded at no fixed interval. Analysts often want to use time series analysis techniques that require fixed-time intervals. Therefore, the transactional data must be accumulated to form a fixed-interval time series, such as daily, weekly, or monthly.

The TSMODEL procedure forms time series vectors from timestamped data and then provides these vectors as array variables for subsequent processing by your program statements. Your program statements are processed independently for each BY group. The TSMODEL procedure is similar to the SAS DATA step for time series data. The SAS DATA step processes data by each row, whereas the TSMODEL procedure processes time series vectors for the BY groups.

Because PROC TSMODEL runs on SAS Cloud Analytic Services (CAS), it can process the time series data in parallel. Time series for the BY groups are divided across the nodes of the CAS session, and then threads are used on each node to process the node's BY groups concurrently.

All results of the time series analysis can be stored to CAS tables.

Comparison of the TSMODEL and TIMESERIES Procedures

The TSMODEL procedure has similarities to the TIMESERIES procedure in SAS/ETS. The TIMESERIES procedure enables you to perform a variety of standard time series analysis techniques with its various statements, whereas the TSMODEL procedure enables you to define your own analyses via user-defined program statements that you include in the procedure's statement block. The TSMODEL procedure provides no built-in time series analysis capabilities. You must provide a program to analyze the time series data. For more information about PROC TIMESERIES, see *SAS/ETS User's Guide*.

Comparison of the TSMODEL and TIMEDATA Procedures

The syntax of the TSMODEL procedure is similar to the syntax of the TIMEDATA procedure in SAS/ETS software. For more information about PROC TIMEDATA, see *SAS/ETS User's Guide*.

PROC TSMODEL requires that you specify CAS tables for all input data and all output data. Unlike PROC TIMEDATA, no actual processing of the time series data occurs in PROC TSMODEL; data in PROC TSMODEL are processed in the CAS session that is associated with the CAS librefs that you specify in the PROC TSMODEL statement.

Like PROC TIMEDATA, PROC TSMODEL supports auxiliary input tables, which you specify in the AUXDATA= option. However, PROC TSMODEL requires that either all or none of the BY variables are

present in an AUXDATA= table. No partial BY group matching for the AUXDATA= tables is supported at this time.

A simple example provides insight about the relative ease of moving from PROC TIMEDATA to PROC TSMODEL. Consider the following PROC TIMEDATA example, which rescales the Sale and Price variables for each of the Product BY groups:

```
proc timedata data=mylib.pricex
    outsum=mylib.pricexsum
    outarray=mylib.pricexoa
    outscalar=mylib.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale / accumulate=total;
var Price / accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
  if Sale[i] ne . then do;
    relsale[i] = Sale[i]/sbase;
  end;
  if Price[i] ne . then do;
     relprice[i]=Price[i]/pbase;
  end;
end;
run;
```

When you convert the preceding statements to use PROC TSMODEL with a CAS table, the code looks like this:

```
proc tsmodel data=mycas.pricex
     outsum=mycas.pricexsum
     outarray=mycas.pricexoa
     outscalar=mycas.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale /accumulate=total;
var Price /accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
submit;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
  if Sale[i] ne . then do;
     relsale[i] = Sale[i]/sbase;
  end;
  if Price[i] ne . then do;
     relprice[i]=Price[i]/pbase;
  end;
end;
endsubmit;
quit;
```

Examine these two examples closely to see the differences. At first glance, the differences might not be very obvious. Although the differences in code syntax are subtle, the underlying differences in the procedure processing are profound.

One important difference between PROC TIMEDATA and PROC TSMODEL is that PROC TSMODEL requires you to use the SUBMIT and ENDSUBMIT statements to begin and end the set of programming statements that you want to execute on the time series BY groups. You can include SAS macro statements in the statements in your SUBMIT/ENDSUBMIT block. Table 11.1 shows the differences between the processing of programming statements in PROC TSMODEL compared to PROC TIMEDATA.

Table 11.1 Comparison of PROC TSMODEL and PROC TIMEDATA

Processing Step	PROC TSMODEL	PROC TIMEDATA
Statement parsing	Any parse-time errors are reported only after the ENDSUBMIT statement. Those errors come from the parsing and compilation of your code in your CAS session. SAS line numbers for incorrect statements are not reported in the SAS log.	Statements are parsed as the procedure works its way through its statement block. Any parse-time errors are interspersed with the program statements in the SAS log.
PROC FCMP subroutines and functions	Does not support as many built-in PROC FCMP functions and subroutines.	Supports built-in PROC FCMP functions and subroutines.
RUN_MACRO	Not supported because it cannot be performed by the CAS server when it executes your program statements in the context of your CAS session.	Can call the PROC FCMP subroutine RUN_MACRO to run a SAS macro, which can then call a SAS procedure.
ID statement with a SAS date or datetime variable	An ID variable is required because the value of the ID variable is needed to determine the time index of each row within the BY group. The accumulation frequency that you specify in the INTERVAL= option in the ID statement is required.	An ID statement is not required.
Custom intervals	Custom intervals are not supported at this time.	Custom intervals are supported.
Accumulation modes	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are not supported.	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are supported.

Table 11.1 *continued*

Processing Step	PROC TSMODEL	PROC TIMEDATA
Output data	Can be printed and plotted through other SAS procedures.	Supports PRINT and PLOT options to display data.

Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 7 in Chapter 3, “Shared Concepts.”

Getting Started: TSMODEL Procedure

This section outlines the use of the TSMODEL procedure and describes some of the analysis techniques that you can perform on timestamped transactional data.

Suppose that a bank wants to analyze the transactions that are associated with each of its customers over time. Further, suppose that the CAS table mycas.transactions contains four variables that are related to these transactions: Customer, Date, Withdrawals, and Deposits. The following examples illustrate possible ways to analyze these transactions by using the TSMODEL procedure.

The following statements accumulate the timestamped transactional data to form a daily time series based on the accumulated daily totals of each type of transaction (Withdrawals and Deposits). These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.transactions
             out=mycas.timeseries(replace=yes)
             outarray=mycas.arrays(replace=yes);
   by Customer;
   id Date interval=day accumulate=total;
   var withdrawals deposits;
   outarrays Balance;
   submit;
   Balance[1] = deposits[1] - withdrawals[1];
   do t = 2 to _LENGTH_;
      Balance[t] = Balance[t-1] + (deposits[t] - withdrawals[t]);
   end;
   endsubmit;
quit;
```

The OUT= option requests that the resulting time series data for each customer be stored in the table mycas.timeseries. The OUTARRAY= option requests that the resulting time series data along with a newly created variable, Balance, be stored in the table mycas.arrays. Both tables are created in the CAS session's current caslib.

The INTERVAL=DAY option requests that the transactions be accumulated on a daily basis within each Customer according to the values of the ID variable Date. The ACCUMULATE=TOTAL option requests that the sum of the transactions be calculated. After the transactional data are accumulated into a time series format, the example code computes a daily balance for each customer. Many of the procedures provided with SAS software can be used to perform further processing on the resulting time series data. The tables that are produced by PROC TSMODEL can also be used as input to subsequent PROC TSMODEL steps.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Figure 11.1.

Figure 11.1 Summary of Time Series Processing

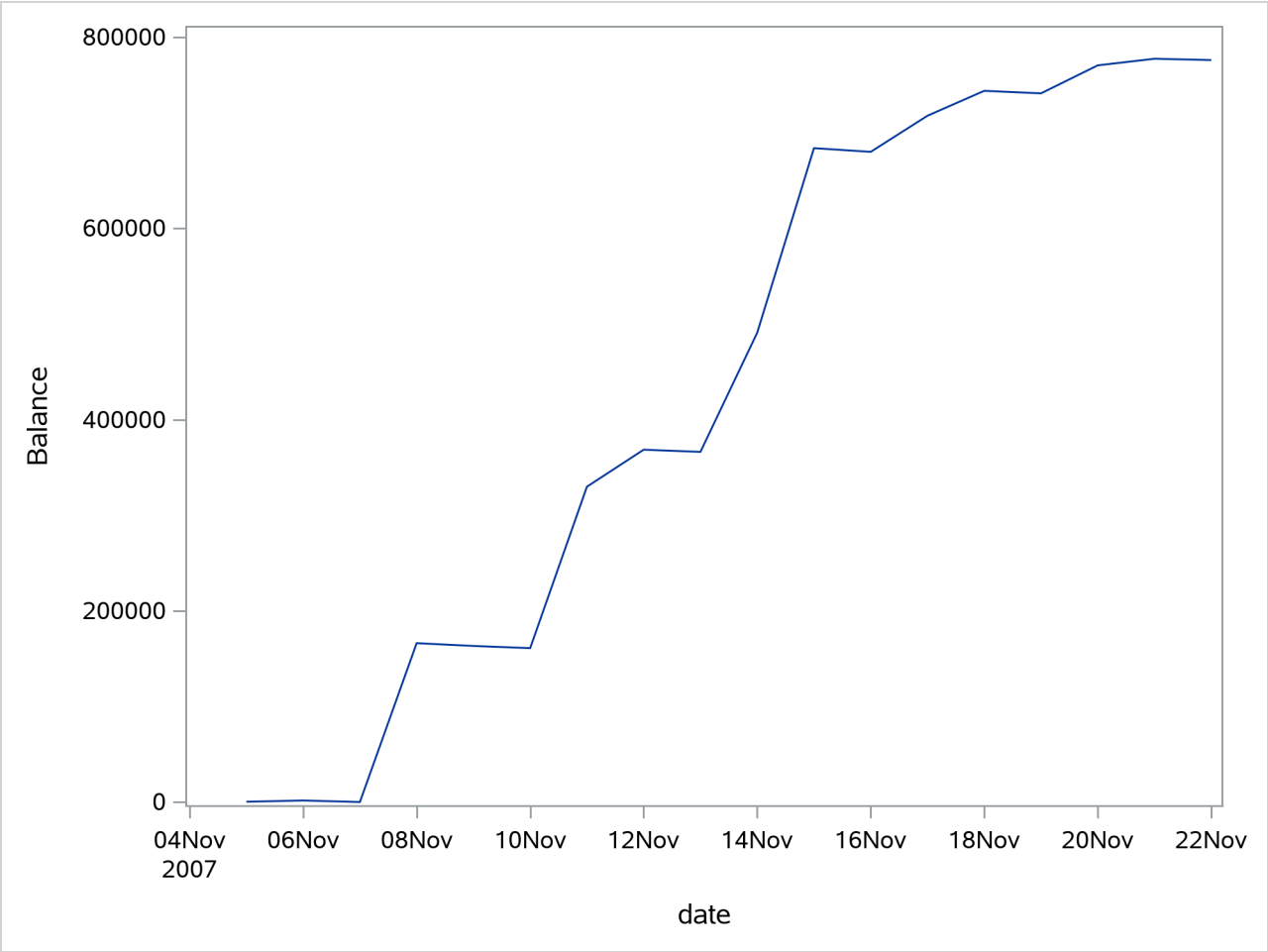
The TSMODEL Procedure

Processing Summary	
CAS table	TRANSACTIONS
Number of analysis variables	2
Number of rows read	600
Number of groups read	3
Memory for group packages (KB)	1
Time to load groups (seconds)	0.0144879818
Minimum time ID	05NOV2007
Maximum time ID	22NOV2007
Minimum time periods	18
Maximum time periods	18
Number of nodes run	4
Number of nodes with data	2
Number of nodes with groups	2
Number of threads budgeted	48
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	48
Maximum threads active	48
Number of groups processed by submitted code	3
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0355381966
Number of output table rows produced	54
Number of array table rows produced	54

You might want to plot the generated Balance series for some particular customer. The following code produces a graph for the customer named 'Bill'; the graph is shown in Figure 11.2.

```
proc sgplot data=mycas.arrays(where=(customer='Bill'));
   series x=Date y=balance;
run;
```

Figure 11.2 Balance for Bill



Syntax: TSMODEL Procedure

The following statements are available in the TSMODEL procedure:

PROC TSMODEL *options* **;**
  BY *variables* **;**
  ID *variable* **INTERVAL=***interval* < *options* > **;**
  OUTARRAYS *array-name-list* **;**
  OUTSCALARS *scalar-name-list* **;**
  INSCALARS *scalar-name-list* **;**
  VAR *variable-list* / *options* **;**
  REQUIRE *package-list* **;**
  PRINT *print-options* **;**
  SUBMIT < **FILE=** *SAS-file-ref* | '*File-path*' > < *submit-options* > **;**
  Programming statements **;**
  ENDSUBMIT **;**

The PROC TSMODEL and ID statements are required.

The following sections present a summary of the statements and options that are used in PROC TSMODEL, a description of the TSMODEL statement, and then descriptions of the other statements.

Functional Summary

Table 11.2 summarizes the statements and options that control the TSMODEL procedure.

Table 11.2 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies variables to analyze	VAR	
Specifies the time ID variable	ID	
Specifies the arrays to output	OUTARRAYS	
Specifies the scalars to output	OUTSCALARS	
Specifies the scalars that are input	INSCALARS	
Specifies the packages to include	REQUIRE	
Specifies the beginning of program statements	SUBMIT	
Specifies the end of program statements	ENDSUBMIT	
Print results from program execution	PRINT	
Table Options		
Specifies the auxiliary input tables	PROC TSMODEL	AUXDATA=
Specifies the input table	PROC TSMODEL	DATA=
Specifies the output table	PROC TSMODEL	OUT=
Specifies the array output table	PROC TSMODEL	OUTARRAY=
Specifies the scalar output table	PROC TSMODEL	OUTSCALAR=
Specifies the summary statistics output table	PROC TSMODEL	OUTSUM=
Specifies the table to contain the BY-group message output log	PROC TSMODEL	OUTLOG=
Specifies whether messages are output to the OUTLOG= table by severity	PROC TSMODEL	LOGCONTROL=
Specifies the input table for scalar variables	PROC TSMODEL	INSCALAR=
Specifies a collector object for an output table	PROC TSMODEL	OUTOBJ=
Specifies a repeater object for an input table	PROC TSMODEL	INOBJ=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of the seasonal cycle	PROC TSMODEL	SEASONALITY=

Table 11.2 *continued*

Description	Statement	Option
Specifies the interval alignment	ID	ALIGN=
Specifies the method for partitioning input tables	ID	DISTRIBUTE=
Specifies the use of the best international format for the time ID values in output tables	ID	NLFORMAT=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, VAR	ACCUMULATE=
Specifies how to interpret missing values	ID, VAR	SETMISSING=
Specifies the method for trimming BY groups	ID	TRIMID=
Miscellaneous Options		
Specifies the forecast horizon or lead used to extend the CAS table	PROC TSMODEL	LEAD=
Specifies the format for the time ID variable	ID	FORMAT=
Specifies the file that contains the user-defined program	SUBMIT	FILE=
Specifies variable values available to the user-defined program	SUBMIT	DYNAMICS=

PROC TSMODEL Statement

PROC TSMODEL *options* **;**

The PROC TSMODEL statement invokes the TSMODEL procedure. You can specify the following *options*:

AUXDATA=*CAS-libref.data-table*
names a table that contains auxiliary input data for the procedure to use for supplying time series variables. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 133. For more information, see the section "Auxiliary Tables" on page 158.

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 133.

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

INOBJ=(*object-name***=***CAS-libref.data-table* **...)**

specifies pairs, each of which binds a repeater object specified by *object-name* with an input table specified by *CAS-libref.data-table*. You can specify one or more object-table pairs as needed to associate the repeater objects that you declare in your user-defined program with their input tables. You must specify a binding for any repeater object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
inobj=(inest=mycas.outest
       inspec=mycas.outspec)
```

This code binds the repeater objects named INEST and INSPEC to the CAS tables mycas.outest and mycas.outspec, respectively. In addition to the columns that are required to satisfy the built-in table schema of a particular repeater object, each specified table must have all or none of the BY variables of the primary DATA= table. When the INOBJ= table has none of the BY variables, all the CAS table rows are input.

Repeater objects are defined in various packages that use PROC TSMODEL as a method to input data that are required for each application. For more information about creating repeater objects for a package, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section "REQUIRE Statement" on page 153.

INSCALAR=*CAS-libref.data-table*

specifies a table to supply scalar dynamic variables to be included and made accessible to your program code as it executes.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 133.

When you specify BY variables for the DATA= table, the INSCALAR= table must contain those BY variables. If you do not specify BY variables, then the INSCALAR= table is read unqualified for the BY variables across the CAS workers in the CAS session. In this case, only a single value for each variable is needed, and only a single row is required in the table. If the table contains multiple rows in this case, an error is generated when the procedure is called. If you specify this option, then you must also specify one or more INSCALARS statements to specify the variables that you want to include for your program to access.

If BY variables are specified in a BY statement, then the table specified in the INSCALAR= option must contain all the specified BY variables or none of them. If BY variables are present in the INSCALAR= table, then the values for the variables specified in the INSCALARS statement are input subject to BY-group processing. If BY variables are not present in the INSCALAR= table, then only a single value of each variable specified in the INSCALARS statement is input for all BY groups. It is an error for the INSCALARS= table to contain more than one value (row) for the variables if the table

is not subject to BY-group processing. If the INSCALARS= table is subject to BY-group processing and multiple values (rows) are present for any BY group, then the value can be input from any row, leading to inconsistent results. For consistent results, you should prepare the INSCALAR= tables such that only a single value is input for each BY group.

LEAD=*n*

specifies the number of periods ahead to extend time series arrays for the variables in both the VAR and OUTARRAYS statements that are output to the CAS table. You can specify this option to accommodate a forecast lead or horizon when you are preparing time series data for forecasting.

The value of *n* is relative to the ending value of the input time ID for each BY group as specified by the TRIMID= option in the ID statement; it is not relative to the last nonmissing observation of a particular series. By default, LEAD=0.

LOGCONTROL=(*severity***=IGNORE | KEEP < *severity*=IGNORE | KEEP*. . .* >)**

specifies pairs that define error severity and associated message disposition for the OUTLOG= option. You can specify multiple LOGCONTROL= options.

You can specify zero or more pairs. In these pairs, *severity* can take one of the following values:

NONE specifies messages that have no severity classification.

NOTE specifies messages whose severity classification is NOTE.

WARNING specifies messages whose severity classification is WARNING.

ERROR specifies messages whose severity classification is ERROR.

You can specify the following values to indicate the associated message disposition:

IGNORE ignores messages of the specified severity.

KEEP retains messages of the specified severity.

This option is applied only when you specify an OUTLOG= option; otherwise, it has no effect. By default, LOGCONTROL=(ERROR=KEEP) when the OUTLOG= option is specified. This default value retains only messages whose severity classification is ERROR and discards all others. However, the default behavior no longer applies when you specify the LOGCONTROL= option. For example, if you specify LOGCONTROL=(NONE=KEEP), then only messages that have no severity classification are retained and all others (i.e., NOTE, WARNING, and ERROR) are discarded.

OUT=*CAS-libref.data-table*

names the output table to contain the time series variables that are specified in the subsequent VAR statements.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 133.

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 133.

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 133.

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 153.

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 133.

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 133.

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 162.

ID Statement

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

The ID statement names a numeric *variable* that identifies the temporal order (time sequence) of observations in the input and output tables. The values of variables are assumed to be SAS date, SAS datetime, or observation values. In addition, the ID statement specifies the frequency to be associated with the time series. The ID statement *options* also specify a global treatment for how the time series variables are accumulated from the BY group's rows and how the time ID values are aligned to form the time series. The specified information affects all variables that are specified in subsequent VAR statements. The ID statement and the INTERVAL= option are required to specify the desired accumulation frequency.

You must specify the following option:

INTERVAL=*interval*

specifies the frequency of the accumulated time series. For example, if the input table consists of quarterly observations, then specify INTERVAL=QTR.

Interval names are constructed from a basic interval type with an optional multiplier and shift. The general form of an interval name is as follows:

type< *multiplier* >.< *shift* >

The interval *type* for SAS date values are summarized as follows:

YEAR

specifies yearly intervals. Abbreviations are YEAR, YEARS, YEARLY, YR, ANNUAL, ANNUALLY, and ANNUALS. The starting subperiod *shift* is in months (MONTH).

YEARV

specifies ISO 8601 yearly intervals. The ISO 8601 year starts on the Monday immediately preceding January 4 (or on January 4 if it is a Monday). Note that it is possible for the ISO 8601 year to start in December of the preceding year. Also, some ISO 8601 years contain a leap week. For more information about ISO weeks, see Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). The starting subperiod *shift* is in ISO 8601 weeks (WEEKV).

R445YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-5-4 months (R454MON). For more information about the retail 4-5-4 calendar, see National Retail Federation (2007).

R544YR

is the same as YEARV except that the starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIYEAR

specifies semiannual intervals (every six months). Abbreviations are SEMIYEAR, SEMIYEARS, SEMIYEARLY, SEMIYR, SEMIANNUAL, and SEMIANN.

The starting subperiod *shift* is in months (MONTH). For example, SEMIYEAR.3 intervals are March–August and September–February.

QTR

specifies quarterly intervals (every three months). Abbreviations are QTR, QUARTER, QUARTERS, QUARTERLY, QTRLY, and QTRS. The starting subperiod *shift* is in months (MONTH).

R445QTR

specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454QTR

specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544QTR

specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

MONTH

specifies monthly intervals. Abbreviations are MONTH, MONTHS, MONTHLY, and MON. The starting subperiod *shift* is in months (MONTH). For example, MONTH2.2 intervals are February–March, April–May, June–July, August–September, October–November, and December–January of the following year.

R445MON

specifies retail 4-4-5 monthly intervals. The 3rd, 6th, 9th, and 12th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. All other months are four ISO 8601 weeks long. R445MON intervals begin with the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th weeks of the ISO year. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454MON

specifies retail 4-5-4 monthly intervals. The 2nd, 5th, 8th, and 11th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R454MON intervals begin with the 1st, 5th, 10th, 14th, 18th, 23rd, 27th, 31st, 36th, 40th, 44th, and 49th weeks of the ISO year. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544MON

specifies retail 5-4-4 monthly intervals. The 1st, 4th, 7th, and 10th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R544MON intervals begin with the 1st, 6th, 10th, 14th, 19th, 23rd, 27th, 32nd, 36th, 40th, 45th, and 49th weeks of the ISO year. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIMONTH

specifies semimonthly intervals. SEMIMONTH breaks each month into two periods, starting on the 1st and 16th days. Abbreviations are SEMIMONTH, SEMIMONTHS, SEMIMONTHLY, and SEMIMON. The starting subperiod *shift* is in SEMIMONTH periods. For example, SEMIMONTH2.2 specifies intervals from the 16th of one month through the 15th of the next month.

TENDAY

specifies 10-day intervals. TENDAY breaks the month into three periods, the 1st through the 10th day of the month, the 11th through the 20th day of the month, and the remainder of the month. (TENDAY is a special interval typically used for reporting automobile sales data.) The starting subperiod *shift* is in TENDAY periods. For example, TENDAY4.2 defines 40-day periods that start at the second TENDAY period.

WEEK

specifies weekly intervals of seven days. Abbreviations are WEEK, WEEKS, and WEEKLY. The starting subperiod *shift* is in days (DAY), with the days of the week numbered as 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, and 7=Saturday. For example, WEEK.7 means weekly with Saturday as the first day of the week.

WEEKV

specifies ISO 8601 weekly intervals of seven days. Each week starts on Monday. The starting subperiod *shift* is in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 starts on Monday, WEEKV.2 starts on Tuesday, and so on.

WEEKDAY
WEEKDAY*d*W
WEEKDAY*dd*W
WEEKDAY*ddd*W

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:

DT*SAS-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 155.

ALIGN=*option*

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

BEGINNING | BEG | B represents each time period using the beginning SAS date or datetime value of the time period.

ENDING | END | E represents each time period using the ending SAS date or datetime value of the time period.

MIDDLE | MID | M represents each time period using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

DISTRIBUTE=*option*

specifies the method that is used to distribute the input time series observations in the input tables across the CAS worker nodes in the CAS session. This process, which is known as "autopartitioning," is performed in order to form complete time series that are suitable for processing. You can specify the following options:

AUTO automatically selects the best approach, either the PREACCUMULATE or SHUFFLE *option*, to distribute the input time series observations across the CAS worker nodes in the CAS session. The selection occurs on the basis of both the accumulation time interval that you specify in the INTERVAL= option and the level of scattering of the time series observations across the CAS worker nodes.

PREACCUMULATE | PREACC generates partially accumulated time series (that is, those that have a fixed length and a fixed interval) in each CAS worker node in the CAS session and then distributes their observations to their destination CAS worker nodes. These partially accumulated time series are accumulated again upon arrival at their destination CAS worker nodes in order to form complete time series. This option can usually reduce the amount of time that is required in order to distribute time series observations across the CAS worker nodes, especially in the case in which you specify an accumulation interval in the INTERVAL= option that is larger than the natural frequency of the input data. However, this option can require additional memory in order to generate the partially accumulated time series in the CAS worker nodes.

SHUFFLE distributes the raw time series observations to their destination CAS worker nodes in the CAS session before accumulating them into complete time series. This option is recommended when you specify an accumulation time interval in the INTERVAL= option that is close to the natural frequency of the input data. This option requires less memory than the PREACCUMULATE option because it does not have to generate partially accumulated time series in the memory of each CAS worker node. However, this option can lead to increased run time compared to the PREACCUMULATE option when it is used outside the recommended setting.

By default, DISTRIBUTE=AUTO.

END=*value*

specifies a SAS date or datetime value that represents the end of the data. If the last time ID variable value is less than *value*, then the series is extended with missing values. If the last time ID variable value is greater than *value*, then the series is truncated. For example, END="&sysdate"D uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. You can specify the START= and END= options to ensure that the data that are associated within each BY group contain the same number of observations.

FORMAT=*format*

specifies the SAS format for the time ID values. If this option is omitted, then the default format is inferred from the INTERVAL= option. You can specify FORMAT=_DATA_ to force the SAS format of the time ID variable to be automatically propagated to the time ID values.

NLFORMAT=*option*

enables the selection of the best international SAS format for the time ID values on the basis of the accumulation time interval that you specify in the INTERVAL= option. This option is ignored when you specify the FORMAT= option. You can specify the following *options*:

NO selects the best English-language-based SAS format for the time ID values.

YES selects the best international SAS format for the time ID values.

By default, NLFORMAT=NO.

SETMISSING=*option*

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

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=*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 11.1 and Output 11.1.1.

INSCALARS Statement

INSCALARS *scalar-name-list* **;**

INSCALAR *scalar-name-list* **;**

The INSCALARS statement specifies which scalar variables to automatically include for your program to use. These variables can be numeric or character. Each variable that you name in an INSCALARS statement must be defined in the table specified in the INSCALAR= option in the PROC TSMODEL statement. You must specify at least one INSCALARS statement if you specify the INSCALAR= option in the PROC TSMODEL statement. You can specify multiple INSCALARS statements. For more information, see the INSCALAR= option.

OUTARRAYS Statement

OUTARRAYS *array-name-list* **;**

OUTARRAY *array-name-list* **;**

The OUTARRAYS statement specifies a list of array names; each name specifies a numeric output array variable to be stored in the table that is specified in the OUTARRAY= option in the PROC TSMODEL statement. You can specify multiple OUTARRAYS statements.

Your program statements can create and use any number of arrays. The array variables that are specified in the OUTARRAYS statement are created automatically for you, and their lengths are predetermined by the length of the time ID vector for the BY group. The arrays are initialized to missing values at the start of each BY group. Although you can define and use many arrays in your program, only arrays that are listed in the OUTARRAYS statement are included in the table that is specified in the OUTARRAY= option.

OUTSCALARS Statement

OUTSCALARS *scalar-name-list* **;**

OUTSCALAR *scalar-name-list* **;**

The OUTSCALARS statement specifies a list of scalar names; each name specifies a numeric or character output scalar variable to be stored in the table that is specified in the OUTSCALAR= option in the PROC TSMODEL statement. You can specify multiple OUTSCALARS statements.

Your program statements can create and use any number of scalars. Only scalars that are listed in the OUTSCALARS statement are predefined and included in the table that is specified in the OUTSCALAR= option. The scalars are initialized to missing values at the start of each BY group. Each scalar name can specify numeric or character scalar variables which are defined as follows:

var-list	defines the variables in the *var-list* as numeric scalars.
var-list $*n*	defines the variables in the *var-list* as character scalars with length *n*.

The following statement defines scalar variables rc, holdout, and nchanges as numeric scalars, and defines selected as character with length 32.

```
outscalar rc selected $32 holdout nchanges;
```

PRINT Statement

PRINT *print-options* **;**

The PRINT statement allows you to display results that are saved to CAS tables that you specify in the PROC TSMODEL statement options. You can include multiple PRINT statements in your PROC TSMODEL statement block. Each PRINT statement is processed independently in the order that you specify them.

The PRINT OUTLOG statement has the following form:

PRINT OUTLOG < **WHERE** *where-clause* > **;**

PRINT OUTLOG displays information from the OUTLOG= table to the SAS LOG. If you specify a BY statement, then the messages that are stored to the OUTLOG table are printed in BY group order followed with a BY line separator. You can specify an optional WHERE clause to select the rows from the OUTLOG table that you want to print. The WHERE clause must reference columns of the OUTLOG= table.

Following are four examples of PRINT OUTLOG statements that you might specify:

```
print outlog;
print outlog where regionName eq 'Region1';
print outlog where _errno_ ne 0;
print outlog where _log_ contains "sale";
```

The first PRINT statement prints all OUTLOG= table rows to the SAS LOG, the second selects rows where the regionName column has the value 'Region1', the third selects rows where the _errno_ column has a non-zero value, and the last selects rows where the _log_ column text contains the word 'sale'.

VAR Statement

VAR *variable-list* < / *options* > **;**

The VAR statement lists the numeric variables in the DATA= table or AUXDATA= tables whose values are to be accumulated to form the time series.

An input table variable can be specified in only one VAR statement. You can specify any number of VAR statements. You can also specify the following *options* in the VAR statements:

ACCUMULATE=*option*

specifies how to accumulate the table observations within each time period for the variables in the *variable-list*. If you do not specify this option, accumulation is determined by the ACCUMULATE= option in the ID statement.

SETMISS=*option* | *number*

SETMISSING=*option* | *number*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series for variables in the *variable-list*. If you do not specify this option, missing values are set based on the SETMISSING= option in the ID statement.

REQUIRE Statement

REQUIRE *package-1* < *package-2* . . . *package-n* > **;**

REQUIRE *package (class-1* . . . *class-n)* **;**

The REQUIRE statement specifies which time series and time frequency analysis packages to make available for your user-defined program. These packages include functions, subroutines, and objects that you can utilize from your program to perform sophisticated time series processing. These packages provide functionality that ranges from a simple function to count missing observations in an array to very sophisticated objects that perform automatic time series modeling and forecasting.

The first form of the REQUIRE statement enables you to specify a list of package names that are available for use. If you use this form, you must specify at least one package name. You can specify multiple packages in the same REQUIRE statement.

The second form of the REQUIRE statement enables you to selectively identify which objects in *package* you intend to use. Packages might contain a number of different objects, and it can be more efficient to register only the ones that you plan to use.

You can specify multiple REQUIRE statements. All packages that are specified in REQUIRE statements are loaded prior to parsing your program statements so that any references are defined at the time your code is parsed. If you specify an invalid package name, then an error is returned prior to parsing your program statements. For more information about the packages that are available in SAS Visual Forecasting, see *SAS Visual Forecasting: Time Series Packages*. For more information about the packages that are available in SAS Econometrics, see *SAS Econometrics: Econometrics Procedures*.

SUBMIT Statement

SUBMIT < **FILE=** *SAS-file-ref* | '*File-path*' > < *submit-options* > **;**

The SUBMIT statement indicates the start of your user-defined program statements or specifies the location of a file that contains your user-defined program statements.

If the SUBMIT statement does not specify a file, then all lines of code that are specified between the SUBMIT statement and the ENDSUBMIT statement are executed on the time series BY groups that arise from your timestamped input data. Compilation errors are reported back to the SAS log, and no further processing occurs.

Notice the placement of the SUBMIT and ENDSUBMIT in the example in the "Comparison of the TSMODEL and TIMEDATA Procedures" on page 130. The use of SUBMIT and ENDSUBMIT statements to delineate of the program block is a major difference between PROC TSMODEL and PROC TIMEDATA.

You can specify the following *submit-options*:

FILE=*SAS-file-ref* | '*File-path*'

specifies a file that contains your user-defined program. If you specify this option, you must not specify the ENDSUBMIT statement.

You can specify the file either by including a *SAS-file-ref* that is defined in a prior SAS FILENAME statement or by including a quoted string that identifies the host-specific file path.

The following rules apply when you include the FILE= option:

- The text file you specify must contain only the program statements that you want to submit and none of the PROC TSMODEL statements. You must not include an ENDSUBMIT statement in the file.
- The text file you specify must not contain any SAS macro statements or macro variable references. The file content is submitted as specified.
- You are free to use any mechanism of your choosing to generate the program statements in the text file. You can use a SAS DATA step with PUT statements, PROC LUA, PROC STREAM, and so on to generate the file from within a SAS program. You can use your favorite editor to generate the text file and then specify it in the SUBMIT statement if the content is static. For more information, see *Getting Started with SAS Viya for Lua*.

DYNAMICS=(*var-1***=[***number* | *"string"***] <** *var-2***=[***number* | *"string"***]. . . >)**

defines variables (constants) outside of the scope of the submitted code. These variables function as global arguments to the submitted code. In conjunction with the INSCALARS statement, these values can be used and evaluated within the submitted code. You can specify both numeric and character variable types. The type of the variable is determined by the right-hand side of the variable's declaration. For example, consider the following DYNAMICS= option:

```
DYNAMICS=(SOF='MAPE' HOLDOUT=10)
```

The variable SOF is defined as a character variable, and its static value is 'MAPE'. The variable HOLDOUT is defined as a numeric variable, and its static value is 10.

ENDSUBMIT Statement

ENDSUBMIT ;

The ENDSUBMIT statement is required to terminate the user-defined program statements in the PROC TSMODEL statement block whenever you specify the SUBMIT statement without the FILE= option. Do not specify the ENDSUBMIT statement if you specify the FILE= option in the SUBMIT statement. If you do not specify the FILE= option and you fail to include an ENDSUBMIT statement to accompany your SUBMIT statement, then the PROC TSMODEL step ends with an error.

SAS Programming Statements

Programming statements ;

You can use all the programming statements that are allowed in PROC FCMP except the following:

- CALL RUN_MACRO subroutine
- CALL READ_ARRAY subroutine
- CALL WRITE_ARRAY subroutine
- Custom intervals in the INTCK and INTNX functions

Details: TSMODEL Procedure

The TSMODEL procedure forms time series data from transactional data, which are analyzed according to the following steps if the relevant option listed on the right is specified:

1. accumulation	ACCUMULATE= option in the ID or VAR statement
2. missing value interpretation	SETMISSING= option in the ID or VAR statement
3. program execution	user-defined program statements
4. descriptive statistics	OUTSUM= option

Accumulation

If you specify the ACCUMULATE= option in the ID or VAR statement, observations in the table that is specified in the DATA= option in the PROC TSMODEL statement are accumulated within each time period. The frequency (width of each time interval) is specified by the INTERVAL= option in the ID statement. The ID variable contains the time ID values. Each time ID value is a SAS date or datetime that identifies the time period that contains the date or datetime. Accumulation is useful when the input table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series, which is used in subsequent analyses.

For example, suppose an input table contains the following observations:

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

If INTERVAL=MONTH is specified in the ID statement, then all the preceding observations fall within a three-month period of time between March 1999 and May 1999. The observations are accumulated within each time period as follows:

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

```
01MAR1999    40
01APR1999    .
01MAY1999    90
```

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

```
01MAR1999    20
01APR1999    .
01MAY1999    30
```

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

```
01MAR1999    10
01APR1999    .
01MAY1999    20
```

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

```
01MAR1999    30
01APR1999    .
01MAY1999    50
```

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

```
01MAR1999    14.14
01APR1999    .
01MAY1999    17.32
```

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

Missing Value Interpretation

Once the data has been accumulated to form a time series based on the INTERVAL= and the ACCUMULATE= options, missing value interpretation is performed. Sometimes missing values should be interpreted as unknown values. In other situations, missing values are known, such as when missing values are created from accumulation and no observation should be interpreted as no value—that is, 0. In the former case, you can specify the SETMISSING= option to interpret how to treat missing values. In the latter case, you can specify SETMISSING=0 in order to treat missing observations as no (zero) values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Summary Statistics

You can compute summary statistics from the working series by specifying the OUTSUM= option in the PROC TSMODEL statement.

SAS Programming Statements

The user-defined program for PROC TSMODEL can contain most of the SAS programming statements and functions available in the DATA step or in the FCMP procedure. However, there are a few differences as noted in "SAS Programming Statements" on page 155. For more information, see the "FCMP Procedure" chapter in the *Base SAS Procedures Guide*.

All variables that are specified in the ID and VAR statements are assigned as predefined arrays for subsequent processing. In addition, all array names that are specified in the OUTARRAYS statements and all the scalars names that are specified in the OUTSCALARS statements are assigned as predefined symbols for subsequent processing.

Predefined Symbols

In addition to the predefined arrays that are specified in the OUTARRAYS statements and the predefined scalars that are specified in the OUTSCALARS statements, the TSMODEL procedure creates predefined symbols that are automatically available for use in the programming statements. The name and description of the predefined symbols are shown in the following subsections; these names must not be used as variable names in any input data set.

Predefined Scalar Values

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
CYCLE	life-cycle index values

Auxiliary Tables

The TSMODEL procedure can use auxiliary tables to contribute input variables to the run of the procedure step. This functionality creates a virtual data source that allows some of the input variables to physically reside in different tables. Some input variables can reside in the primary table, which is specified in the DATA= option, and other input variables can reside in the tables that are specified in one or more AUXDATA= options. This functionality enables sharing of common time series data across multiple projects.

You can specify more than one auxiliary data source to be used to input time series vectors across a particular BY-group hierarchy. To simplify data management, you can isolate variables that have naturally different levels of BY-group qualification into separate tables and use separate AUXDATA= options to supply them.

AUXDATA Functionality

There are two classes of time series table sources:

- a primary table from the DATA= option
- auxiliary data sources from AUXDATA= options

The AUXDATA= option specifies an auxiliary table that provides time series variables that are required for processing but are not included in the table that is specified in the DATA= option.

You can specify multiple AUXDATA= options in the PROC TSMODEL statement. Each AUXDATA= option establishes an auxiliary table source to supply variables that are declared in subsequent statements in the procedure step. If no auxiliary data sources are required, then the AUXDATA= option can be omitted.

Variables referenced in the TSMODEL procedure fall into three classes:

- variables that must be physically present in the primary table, which is specified in the DATA= option
- variables that must be physically present in each auxiliary table that is specified in an AUXDATA= option
- variables that can reside in either the primary or an auxiliary table

The ID variable for PROC TSMODEL must be present in the primary table and all the auxiliary tables that you specify. Variables that you specify in the BY statement must be present in the primary table. The auxiliary tables must contain all of those BY variables or none of them. Partial matching of a leftmost subset of the BY variables is not supported for the auxiliary tables.

The time series variables that you specify in VAR statements can be input from either the primary table or an auxiliary table. Variable resolution proceeds in reverse order from the last AUXDATA= option in the PROC TSMODEL statement to the first. If the variable in question is not found in any of those, the variable must be present in the primary table for the procedure step to be successful.

AUXDATA Alignment across BY Groups

All variables in the BY statement must be physically present in the primary table. However, it is not necessary to have the BY variables present in any of the auxiliary tables. All or none of the BY variables can be present in any auxiliary table.

For example, suppose you have a hierarchy of (REGION, PRODUCT) in the primary table Sales, which holds the time series variables for monthly sales metrics. Suppose you have an auxiliary table called Promotions that has no BY variables and contains analysis variables for promotions, and another table called Returns that contains time series analysis variables for (REGION, PRODUCT) level groupings. In this scenario, each (REGION, PRODUCT) group in the Sales table always includes the time series variables from the Promotions table, and the analysis variables for the matching (REGION, PRODUCT) BY groups from the Returns table. So if (‘SOUTH’,‘EDSEL’) is a BY group from the primary table, any matching rows from the Returns table are used to define the time series variables that are contributed from that table. The time series variables that are contributed by the Promotions table are always included in every BY group from the Sales table.

AUXDATA Alignment over the Time Dimension

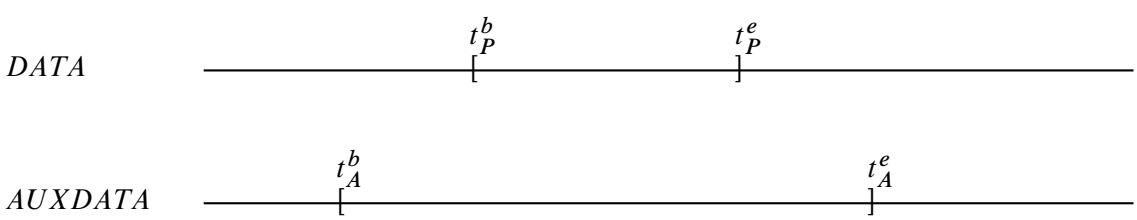
The series from each BY group of the primary table defines a reference time span for the auxiliary tables. Only the intersection of the time span for each auxiliary series with the reference span is input. Leading or trailing missing values are added to the auxiliary series as required to create a time series that has the same span as the reference span. The leading or trailing missing values, if any, are then interpreted according to the value of the SETMISSING= option. For more information, see the SETMISSING= option in the ID statement and the SETMISSING= option in the VAR statement.

When time series are input from a single primary table, the values for all observations of all the time series are contained in the primary table, and no time series needs to be extended with leading or trailing missing values. However, when time series are input from both the primary table and an auxiliary table, the time series from the auxiliary table are truncated or extended as required if the span of the auxiliary series is not identical to the reference span.

For the preceding (REGION, PRODUCT) example, which includes a primary table and two auxiliary tables, consider how differences in the reference span from each BY group affect the time series input from the auxiliary tables in the following cases. In these cases:

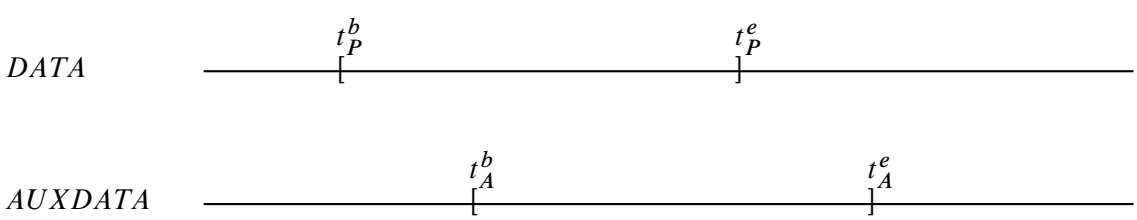
- The DATA line show a series that contains observations in the table specified in the DATA= option.
- The AUXDATA line show a series that contains observations in the table specified in an AUXDATA= option.
- t_P^b denotes the beginning time ID of the primary (DATA) series.
- t_P^e denotes the ending time ID of the primary (DATA) series.
- t_A^b denotes the beginning time ID of the AUXDATA series.
- t_A^e denotes the ending time ID of the AUXDATA series.
- $[t_P^b, t_P^e]$ denotes the time span for the primary (DATA) series (also known as the reference time span).
- $[t_A^b, t_A^e]$ denotes the time span for the AUXDATA series.

Case 1:



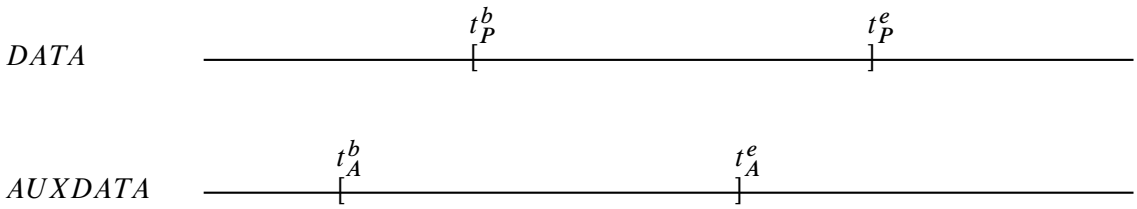
In this case, $[t_P^b, t_P^e] \subseteq [t_A^b, t_A^e]$: the auxiliary time span includes the reference span as a subset. Values in the AUXDATA series to the left of t_P^b and values to the right of t_P^e are truncated from the AUXDATA series. AUXDATA series values in $[t_P^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 2:



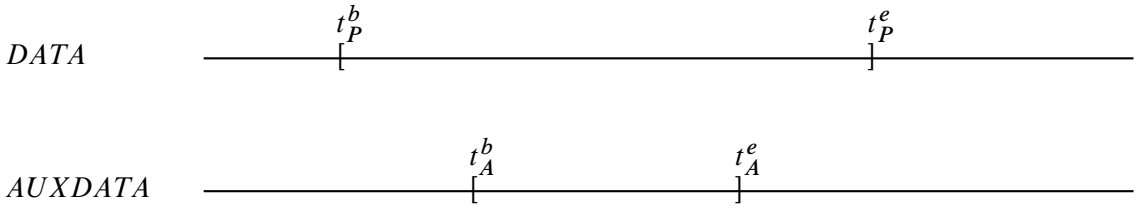
In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^b) \cup [t_A^b, t_P^e]$: the reference time span leads the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^b)$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_P^e, t_A^e]$ are truncated.

Case 3:



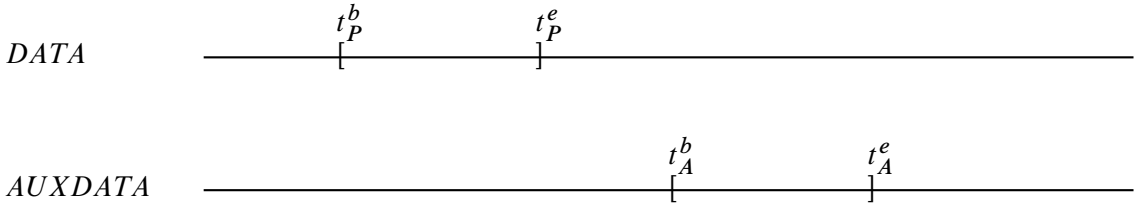
In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^e] \cup (t_A^e, t_P^e]$: the reference time span lags the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^b, t_P^b]$ are truncated.

Case 4:



In this case, $[t_A^b, t_A^e] \subset [t_P^b, t_P^e]$: the auxiliary time span is a subset of the reference time span. AUXDATA series values in $[t_P^b, t_A^b)$ and values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 5:



In this case, $[t_P^b, t_P^e] \cap [t_A^b, t_A^e] = \emptyset$: the auxiliary time span does not intersect the reference time span at all. AUXDATA series values in $[t_P^b, t_P^e]$ are set to missing values, and then those missing values are interpreted according to the value of the SETMISSING= option.

Table Output

The TSMODEL procedure can create the OUT=, OUTARRAY=, OUTSCALAR=, OUTSUM=, OUTLOG=, and OUTOBJ= tables. These tables always contain the variables that are specified in the BY statement. If a BY-group analysis step fails, then the values of this step are not recorded or are set to missing in the related output tables. Appropriate error or warning messages (or both) are recorded in the OUTLOG= table, subject to the value of the LOGCONTROL= option.

It is important to note that output tables created by PROC TSMODEL are naturally partitioned by the BY variables because of the way the BY groups are input into CAS session processes to form the time series data frames. The entire time series data for each BY group is input completely in a particular CAS session process. Consequently, all the time series processing for a particular BY group occurs in the context of a single CAS session node. Subsequent use of PROC TSMODEL to process the output tables makes use of this table-level partitioning.

The BY group partitioning of the output tables should not be confused with BY-group sort order or time ID sort order when the output tables are read by the CAS libname engine back into the SAS DATA step code for subsequent processing. If BY-group order is needed, or if time ID sort order within the BY groups is needed, you must specify PROC SORT steps in your SAS code to order the data set. For example, consider the PROC TSMODEL step in the example "Comparison of the TSMODEL and TIMEDATA Procedures" on page 130. 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.

Examples: TSMODEL Procedure

Example 11.1: Accumulating Transactional Data into Time Series Data

This example uses the TSMODEL procedure to accumulate timestamped transactional data that have been recorded at no particular frequency into time series data at a specific frequency.

Suppose that the input table mycas.retail contains variables Store and Timestamp and numerous other numeric transaction variables. The BY variable Store contains values that divide the transactions into groups (BY groups). The time ID variable Timestamp contains SAS date values that are recorded at no particular frequency. The other variables, Item1–Item8, contain the numeric transaction values to be analyzed. The following statements form monthly time series from the transactional data.

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.retail
     out=mycas.mseries(replace=yes);
   by store;
   id timestamp interval=month
                accumulate=avg
                setmiss=0
                start='01jan1998'd
                end  ='31dec2000'd;
   var item1-item8;
run;
```

The ACCUMULATE=AVG option in the ID statement accumulates the transactions that are recorded with each time period based on the average values. The SETMISS=0 option in the ID statement sets the accumulated time series values for time periods that have no transactions to 0 instead of to missing. The START='01JAN1998'D and END='31DEC2000'D options request that only transactions recorded between the first day of 1998 and the last day of 2000 be considered and, if needed, extended to include this range.

The monthly time series data are stored in the table mycas.mseries. Each BY group that is associated with the BY variable Store contains an observation for each of the 36 months that are associated with the years 1998, 1999, and 2000. Each observation contains the values Store, Timestamp, and each of the analysis variables Item1–Item8 in the input table.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Output 11.1.1.

Output 11.1.1 Summary of Accumulation Processing

The TSMODEL Procedure

Processing Summary	
CAS table	RETAIL
Number of analysis variables	8
Number of rows read	400
Number of groups read	4
Memory for group packages (KB)	1
Time to load groups (seconds)	0.0141348839
Minimum time ID	NOV1998
Maximum time ID	DEC2000
Minimum time periods	36
Maximum time periods	36
Number of nodes run	4
Number of nodes with data	2
Number of nodes with groups	2
Number of threads budgeted	48
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	48
Maximum threads active	48
Number of groups processed by submitted code	4
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0265569687
Number of output table rows produced	144

After each set of transactions has been accumulated to form a corresponding time series, the accumulated time series can be analyzed using various time series analysis techniques. For example, exponentially weighted moving averages can be used to smooth each series.

If the time ID variable Timestamp contains SAS datetime values instead of SAS date values, the INTERVAL=, START=, and END= options must be changed accordingly and the following statements could be used:

```
proc 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 11.2: User-Defined Program Statements

The following SAS macro statements create a user-defined subroutine and a user-defined function. Mylog is a subroutine that log-transforms a time series. Mymean is a function that computes the mean of a time series.

```
%macro fcmpcode;
   subroutine mylog(actual[*], transform[*]);
   outargs transform;
   actlen  = DIM(actual);
   do t = 1 to actlen;
      transform[t] = log(actual[t]);
   end;
   endsub;

   function mymean(z[*]);
   nz  = DIM(z);
   sum = 0;
   nnmiss = 0;
   do t = 1 to nz;
      if z[t] ne . then do;
         sum = sum + z[t];
         nnmiss = nnmiss + 1;
      end;
   end;
   if nnmiss eq 0 then return(.);

   return(sum/nnmiss);
   endsub;
%mend;
```

The following DATA step loads data set Sashelp.Air, which contains airline data into the table mycas.air. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
    set sashelp.air;
run;
```

The input table mycas.air contains the variables Air and Date. The time series is recorded monthly.

The following statements form quarterly time series from the monthly series based on the median value of the total of the transactions recorded within each month. The OUTARRAYS statement specifies the Logair and Logairc arrays as output. The OUTSCALARS statement specifies the Meanlog scalars as output. The other arrays and scalars are not part of the output. The subsequent program statements create the output arrays and scalars.

```
proc tsmodel data=mycas.air
   outarray=mycas.arrays(replace=yes)
   outscalar=mycas.scalars(replace=yes)
   outsum=mycas.airsum(replace=yes);
   id date interval=qtr acc=total;
   var air;
```

```
   outarrays logair logairc;
   outscalars meanlog;
   submit;
      %fcmpcode;
      call mylog(air,logair);
      meanlog = mymean(air);
      do t = 1 to dim(air);
         logairc[t] = logair[t] - meanlog;
      end;
   endsubmit;
quit;
```

The following PROC PRINT step displays the OUTSUM= table, as shown in Output 11.2.1.

```
proc print data=mycas.airsum; run;
```

Output 11.2.1 Summary Statistics for mycas.Air

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV
1	AIR	48	48	0	341	1736	840.89583333	356.43144067
2	logair	48	48	0	5.8318824773	7.4593388952	6.6430769906	0.4395319391
3	logairc	48	48	0	-835.0639509	-833.4364944	-834.2527563	0.4395319393

Obs	_STATUS_	START	END	STARTOBS	ENDOBS
1	0	1949:1	1960:4	1	48
2	0	1949:1	1960:4	1	48
3	0	1949:1	1960:4	1	48

Example 11.3: Using Auxiliary Tables

This example demonstrates the use of the AUXDATA= option in the PROC TSMODEL statement. The data set Sashelp.Gulfoil contains oil and gas production data from the Gulf of Mexico. The following DATA step loads data set Sashelp.Gulfoil into the table mycas.gulfoil.

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.gulfoil(replace=yes);
    set sashelp.gulfoil;
run;
```

The variable RegionName define a time series hierarchy of interest. Suppose you want to generate two new series that contain the region's total share of oil and gas production for each month in the mycas.gulfoil table.

You first use PROC TSMODEL to perform temporal aggregation (accumulation) of the time series:

```
proc 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:

```
data mycas.allreg;
    set mycas.allreg;
    drop oil gas;
run;
```

You can then use PROC TSMODEL with the AUXDATA= option to compute the share of oil and gas production that is contributed by each region for each month. PROC TSMODEL reads a monthly time series for each RegionName group for the variables Oil and Gas from Mycas.Gulfoil. Two new series are produced in the variables Oilshare and Gasshare, which contain the region's share of the oil and gas production, respectively. Those share variables are specified in the OUTARRAY statement for inclusion in the table (mycas.shares), which is specified in the OUTARRAY= option. The time series that are acquired for the variables alloil and allgas are common across all of the RegionName BY groups.

```
proc tsmodel data=mycas.gulfoil
             auxdata=mycas.allreg
             outarray=mycas.shares;
   by regionname;
   outarray oilshare gasshare;
   var oil gas alloil allgas;
   id date interval=month accumulate=total;
   submit;
      do i=1 to _length_;
         oilshare[i] = oil[i] / alloil[i];
         gasshare[i] = gas[i] / allgas[i];
      end;
   endsubmit;
quit;
```

The following code demonstrates that the computed shares sum to 1 for each time index in the resulting Oilshare and Gasshare series. PROC TSMODEL is used to accumulate the shares for these respective variables from the table mycas.shares, and the accumulated share series at the RegionName level are stored to the table mycas.rshares with variable names Oilsum and Gassum, respectively. The tables mycas.shares and mycas.rshares are shown in Output 11.3.1.

```
proc tsmodel data=mycas.shares
     outscalar=mycas.rshares(replace=yes)
     outsum=mycas.rsum(replace=yes);
   id date interval=month accumulate=total;
   var oilshare gasshare;
   outscalar oilsum gassum;
   submit;
      oilsum = 0;
      gassum = 0;
      do t=1 to _length_;
```

```
        oilsum = oilsum + oilshare[t];
        gassum = gassum + gasshare[t];
      end;
   endsubmit;
quit;

proc print data=mycas.rshares; run;
proc print data=mycas.rsum; run;
```

Output 11.3.1 Validation of Oil and Gas Shares by Region

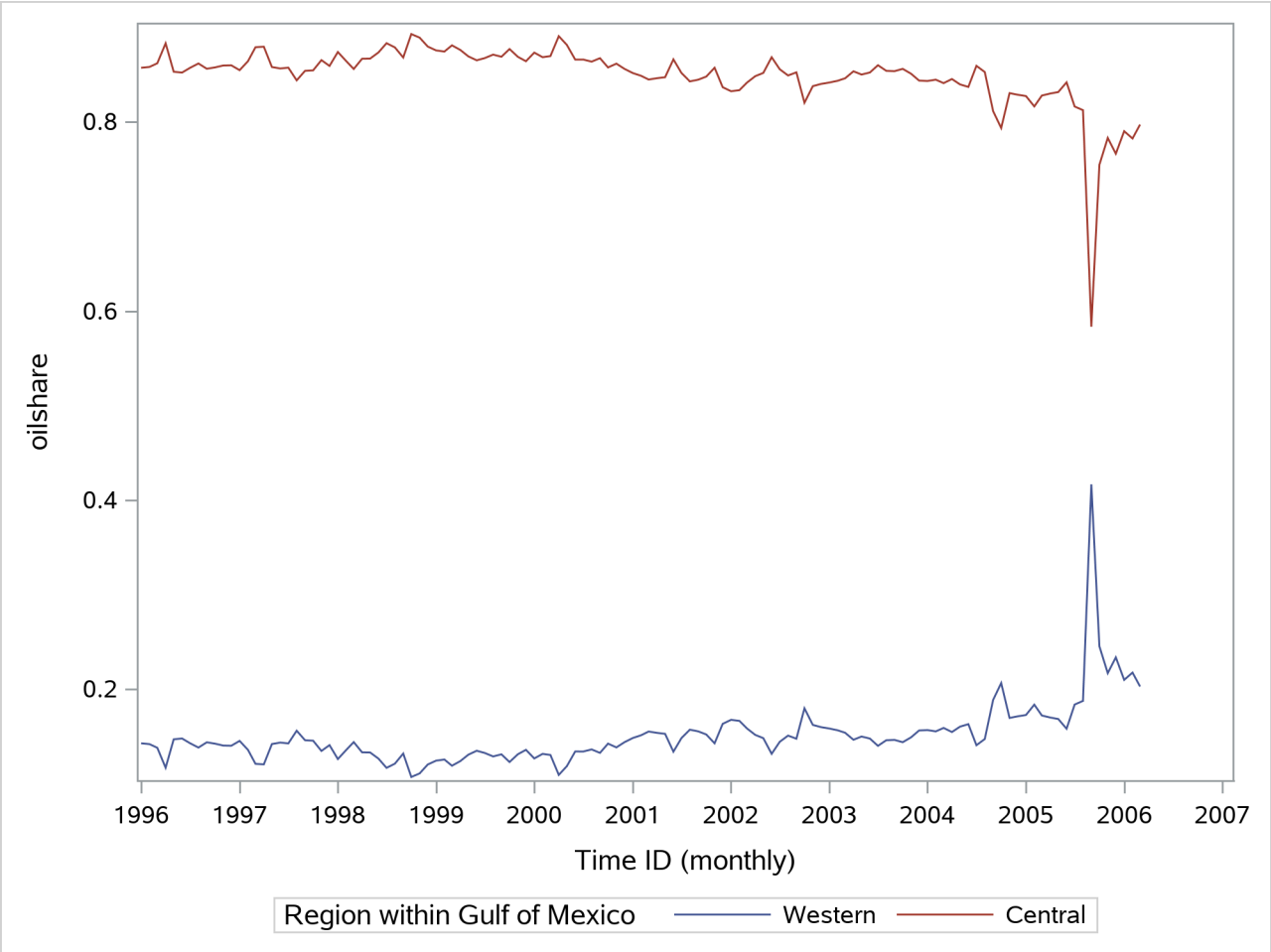
Obs	_STATUS_	oilsum	gassum
1	0	123	123

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV	_STATUS_	START	END	STARTOBS	ENDOBS
1	oilshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123
2	gasshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123

You might also want to plot the share series. The following code produces a graph that overlays the region level share series for oil production as shown in Output 11.3.2.

```
proc sgplot data=mycas.shares;
   series x=Date y=OilShare/group=RegionName;
run;
```

Output 11.3.2 Region Share of Oil Production



References

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2004). "Segmenting Time Series: A Survey and Novel Approach." In *Data Mining in Time Series Databases*, edited by M. Last, A. Kandel, and H. Bunke, 1–22. London: World Scientific. https://www.worldscientific.com/worldscibooks/10.1142/5210.

National Retail Federation (2007). "National Retail Federation 4-5-4 Calendar." http://www.nrf.com/.

Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). *ISO 8601:2004 Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times.* 3rd ed. Technical report, International Organization for Standardization.

Chapter 12
The TSRECONCILE Procedure

Contents

Overview: TSRECONCILE Procedure

When data are organized in a hierarchical fashion, there are often accounting constraints that link the data at different levels of the hierarchy. Typically, for any particular time period, the data in a *parent* node are either the sum or the average of the data of its *child* nodes. For example, the total sales of a product by a retail company are the sum of the sales of the same product in all its stores.

It seems natural to require that the same constraints hold also for the predicted values. Imposing such constraints during the forecasting process can be difficult or impossible. Therefore, the series are often forecast independently at different levels, but this independence results in forecasts do not abide by the constraints that bind the original series. However, you can enforce these constraints by using an after-the-fact process that is called *reconciliation of hierarchical forecasts*.

The TSRECONCILE procedure reconciles forecasts of timestamped data at two different levels of a hierarchy in a top-down fashion for input data that are contained in CAS tables.

PROC TSRECONCILE Compared with the HPFRECONCILE procedure

The TSRECONCILE procedure is the next generation of the HPFRECONCILE procedure (in SAS Forecast Server Procedures) for hierarchical time series forecasts reconciliation. PROC TSRECONCILE was developed specifically for SAS Viya. The syntax of PROC TSRECONCILE is similar to the syntax of PROC HPFRECONCILE, from which it borrows the underlying methodology and goals. PROC TSRECONCILE is designed to run on a cluster of machines that distribute the data and computations in multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Server. Table 12.1 shows the differences between the TSRECONCILE and HPFRECONCILE procedures.

Table 12.1 PROC TSRECONCILE versus PROC HPFRECONCILE

Feature	PROC TSRECONCILE	PROC HPFRECONCILE
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable
Requires equally spaced data?	No, the INTERVAL= option in the ID statement is not supported	Yes, the INTERVAL= option in the ID statement indicates the frequency
Reconciliation direction supported	Top-down	Top-down and bottom-up

Using CAS Sessions and CAS Engine Librefs

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

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

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

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

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

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

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

```
cas mysess terminate;
```

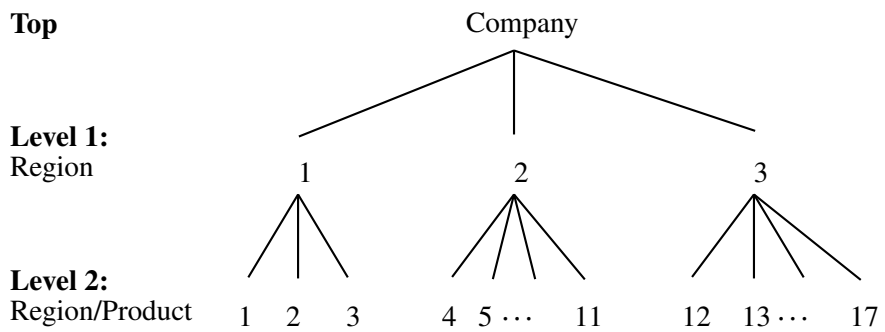
For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: User's Guide*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Getting Started: TSRECONCILE Procedure

This example uses the Sashelp.Pricedata data set to illustrate how to reconcile two levels of a hierarchy of forecasts in a top-down fashion.

The hierarchical structure of data set is shown in Figure 12.1.

Figure 12.1 Hierarchical Structure of Sashelp.Pricedata



The following DATA step loads the Pricedata data set from the Sashelp directory to a table named mycas.pricedata in your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.pricedata(replace=yes);
   set sashelp.pricedata;
run;
```

The following statements use the TSMODEL procedure to generate forecasts for the dependent variable sale first at level 2 (region / product) and then at level 1 (region). For more information about the TSMODEL procedure, see Chapter 11, "The TSMODEL Procedure."

```
/* Define a macro to handle bad return codes of TSModel*/
%macro rccheck;
   if rc < 0 then do; stop; end;
%mend;

/* Automatically select model and generate forecasts at the child level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_region_product)
```

```
        errorstop=yes
        ;
   by region product;
   require atsm;
   id date interval=month;
   var sale;
   submit;
      declare object t(tsdf);
      declare object f(foreng);
      declare object outfor(outfor);
      rc = t.Initialize(); %rccheck;
      rc = t.AddY(sale); %rccheck;
      rc = f.Initialize(t); %rccheck;
      rc = f.SetOption('lead',12); %rccheck
      rc = f.Run(); %rccheck;
      rc = outfor.Collect(f); %rccheck;
   endsubmit;
run;

/* Automatically select model and generate forecasts at the parent level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_region)
       errorstop=yes
       ;
   by region;
   require atsm;
   id date interval=month;
   var sale;
   submit;
      declare object t(tsdf);
      declare object f(foreng);
      declare object outfor(outfor);
      rc = t.Initialize(); %rccheck;
      rc = t.AddY(sale); %rccheck;
      rc = f.Initialize(t); %rccheck;
      rc = f.SetOption('lead',12); %rccheck
      rc = f.Run(); %rccheck;
      rc = outfor.Collect(f); %rccheck;
   endsubmit;
run;
```

The following statements use the TSRECONCILE procedure to reconcile the separate forecasts in a top-down fashion:

```
/* Reconcile child to parent with default options */
proc tsreconcile
      child = mycas.outfor_region_product
      parent = mycas.outfor_region
      outfor = mycas.recfor_region_product
   ;
   by region product;
   id date;
run;
```

PROC TSRECONCILE requires the following inputs:

- The child table at the region / product level, specified in the CHILD= option in the PROC statement
- The parent table at the region level, specified in the PARENT= option in the PROC statement
- An output table, specified in the OUTFOR= option in the PROC statement
- BY variables at the child level (region / product), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSRECONCILE Procedure

The following statements are available in the TSRECONCILE procedure:

```
PROC TSRECONCILE <options> ;
    BY variables ;
    ID variable <options> ;
    CHILDROLES <options> ;
    PARENTROLES <options> ;
```

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

Functional Summary

Table 12.2 summarizes statements and options that are used with the TSRECONCILE procedure.

Table 12.2 PROC TSRECONCILE Functional Summary

Description	Statement	Option
Statements		
Specifies the BY variables	BY	
Specifies the time ID variable	ID	
Names variables in the CHILD= data table	CHILDROLES	
Names variables in the PARENT= data table	PARENTROLES	
Time ID Options		
Specifies the ending time ID value	ID	END=
Specifies the starting time ID value	ID	START=
Data Table Options		
Specifies the child-level input data table	PROC TSRECONCILE	CHILD=
Specifies the parent-level input data table	PROC TSRECONCILE	PARENT=
Specifies the output data table to contain the reconciled forecasts	PROC TSRECONCILE	OUTFOR=

Table 12.2 *continued*

Description	Statement	Option
Specifies the variable that contains the actual values in the CHILD= data table	CHILDROLES	ACTUAL=
Specifies the variable that contains the actual values in the PARENT= data table	PARENTROLES	ACTUAL=
Specifies the variable that contains the predicted values in the CHILD= data table	CHILDROLES	PREDICT=
Specifies the variable that contains the predicted values in the PARENT= data table	PARENTROLES	PREDICT=
Specifies the variable that contains the lower confidence limit in the CHILD= data table	CHILDROLES	LOWER=
Specifies the variable that contains the lower confidence limit in the PARENT= data table	PARENTROLES	LOWER=
Specifies the variable that contains the upper confidence limit in the CHILD= data table	CHILDROLES	UPPER=
Specifies the variable that contains the upper confidence limit in the PARENTROLES= data table	PARENTROLES	UPPER=
Specifies the variable that contains the prediction error in the CHILD= data table	CHILDROLES	ERROR=
Specifies the variable that contains the prediction error in the PARENT= data table	PARENTROLES	ERROR=
Specifies the variable that contains the standard error in the CHILD= data table	CHILDROLES	STD=
Specifies the variable that contains the standard error in the PARENT= data table	PARENTROLES	STD=
Analysis Options		
Specifies the aggregation method	PROC TSRECONCILE	AGGREGATE=
Specifies the confidence level	PROC TSRECONCILE	ALPHA=
Specifies method for confidence limits	PROC TSRECONCILE	CLMETHOD=
Specifies the disaggregation method	PROC TSRECONCILE	DISAGGREGATION=
Leaves zeros forecast unchanged	PROC TSRECONCILE	LOCKZERO
Specifies sign bound on the reconciled series	PROC TSRECONCILE	SIGN=
Requests that the loss function be weighted by the inverse of the prediction variances	PROC TSRECONCILE	WEIGHTED

PROC TSRECONCILE Statement

PROC TSRECONCILE *options* **;**

The PROC TSRECONCILE statement invokes the TSRECONCILE procedure. You can specify the *options* that are described in the following subsections.

Required Options Related to Input Data Tables

You must specify the following data tables:

CHILD=*CAS-libref.data-table*

DISDATA=*CAS-libref.data-table*

DATA=*CAS-libref.data-table*

specifies the input data table that contains the forecast of the time series that represent the children nodes in the hierarchy. Typically, the CHILD= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure. The dimensions of the CHILD= data table are greater than the dimensions of the PARENT= 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, see the section "CHILD= Data Table" on page 186.

PARENT =*CAS-libref.data-table*

AGGDATA=*CAS-libref.data-table*

specifies the input data table that contains the forecasts of the time series that represent the parent nodes in the hierarchy. Typically, the PARENT= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure. *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.

The PARENT= data table must contain a proper subset, possibly empty, of the BY variables that are present in the CHILD= data table.

For more information, see the section "PARENT= Data Table" on page 187.

Required Option Related to the Output Data Table

You must specify the following output data table:

OUTFOR=*CAS-libref.data-table*

specifies the name of the output data table in which to store the reconciled values. *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, see the section "OUTFOR= Data Table" on page 187.

Options Related to the Analysis

AGGREGATE=SUM | AVERAGE

specifies whether the dependent variable in the PARENT= data table is the total sum or the average over the BY groups of the dependent variable in the CHILD= data table. By default, AGGREGATE=SUM.

ALPHA=α

specifies the level of the confidence limits when CLMETHOD=GAUSSIAN, where α must be between 0.0001 and 0.9999. The upper and lower confidence limits will have a $1 - \alpha$ confidence level. By default, ALPHA=0.05, which produces 95% confidence intervals.

CLMETHOD=SHIFT | GAUSSIAN

specifies which method to use to compute confidence limits for the reconciled forecasts.

You can specify the following methods:

GAUSSIAN computes the confidence intervals by assuming that the forecasts are approximately Gaussian.

SHIFT computes the confidence intervals by re-centering the original confidence intervals around the new forecasts.

By default, CLMETHOD=SHIFT.

For more information about the methods of computing confidence intervals, see the section "Details: TSRECONCILE Procedure" on page 183.

DISAGGREGATION=*option*

LOSS=*option*

specifies the type of disaggregation method and type of loss function for top-down reconciliation.

You can specify the following values:

DIFFERENCE bases the loss function on the root mean squared error (RMSE), which results in adjustments that are the (possibly weighted) mean difference of the aggregated child nodes and the parent node.

PROPORTIONS uses a loss function that results in reconciled forecasts that are the (possibly weighted) proportional disaggregation of the parent node.

For more information about the disaggregation methods and associated loss functions, see the section "Top-Down Reconciliation" on page 184.

By default, DISAGGREGATION=PROPORTIONS.

LOCKZERO

specifies that a value of 0 for the Predict variable in the CHILD= data table implies a value of 0 for the corresponding observation in the OUTFOR= data table.

SIGN=*option*
specifies the sign in the reconciled series.

You can specify the following values:

MIXED | NONE	specifies that the output series can have any sign.
NONNEGATIVE | POSITIVE	specify that the output series is nonnegative.
NONPOSITIVE | NEGATIVE	specifies that the output series is nonpositive.

By default, SIGN=MIXED.

WEIGHTED
weights the loss function for top-down reconciliation by the inverse of the variance of the input forecasts.

BY Statement

BY *variables* **;**

The BY statement defines separate groups of observations for the CHILD= data table. BY variables can be either character or numeric.

All BY variables must exist in the CHILD= data table. Conversely, only a strict subset of or none of the BY variables must be present in the PARENT= data table. The BY variables that are present in the PARENT= data table are called AGGBY or PARENTBY variables. Because the AGGBY variables form a proper subset of the BY variables, their number must be less than the number of BY variables. PROC TSRECONCILE finds the AGGBY variables by comparing the variables in the BY statement with the variables in the PARENT= data table.

CHILDROLES Statement

CHILDROLES < *options* > **;**

The CHILDROLES statement enables you to specify names for forecasting variables in the CHILD= data table.

You can specify the following *options*:

ACTUAL=*variable-name*
specifies the name of the variable in theCHILD= data table that contains the actual values. By default, ACTUAL=ACTUAL.

PREDICT=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the predicted values. By default, PREDICT=PREDICT.

LOWER=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the lower confidence limit values. By default, LOWER=LOWER.

UPPER=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the upper confidence limit values. By default, UPPER=UPPER.

ERROR=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the error values. By default, ERROR=ERROR.

STD=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the standard error values. By default, STD=STD.

ID Statement

ID *variable* < /*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, time, or datetime values.

You can specify the following *options* in the ID statement:

END=*option*
specifies a SAS date, datetime, or time value that represents the date at which the reconciliation ends. If the largest *variable* value is less than the END= value, this option has no effect.

START=*option*
specifies a SAS date, datetime, or time value that represents the *variable* value at which the reconciliation begins. This option can be used to limit the reconciliation process only to forecasts that are outside the historical period.

PARENTROLES Statement

PARENTROLES < *options* > **;**

The PARENTROLES statement enables you to specify custom names for forecasting variables in the PARENT= data table.

You can specify the following *options*:

ACTUAL=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the actual values. By default, ACTUAL=ACTUAL.

PREDICT=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the predicted values. By default, PREDICT=PREDICT.

LOWER=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the lower confidence limit values. By default, LOWER=LOWER.

UPPER=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the upper confidence limit values. By default, UPPER=UPPER.

ERROR=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the error values. By default, ERROR=ERROR.

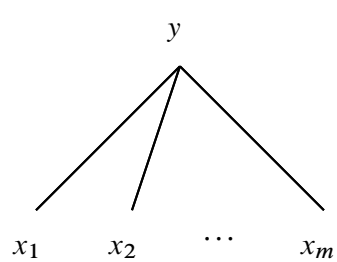
STD=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the standard error values. By default, STD=STD.

Details: TSRECONCILE Procedure

Notation

Assume a two-level hierarchical structure as depicted in Figure 12.2, where y is a parent series and x_i, $i = 1, \ldots, m$, are the child series.

Figure 12.2 Hierarchical Structure



Let y_t be the values of the parent series at time t, and let $\mathbf{x}_t = [x_{1,t}, x_{2,t}, \ldots, x_{m,t}]'$ be the vector of child series at time t, $t = 1, \ldots, T$. As usual, indicate by $\hat{y}_t$ and $\hat{\mathbf{x}}_t$ the prereconciliation forecasts of y_t and $\mathbf{x}_t$, respectively, and denote by $\hat{\boldsymbol{\sigma}}_t = [\hat{\sigma}_{1,t}, \hat{\sigma}_{2,t}, \ldots, \hat{\sigma}_{m,t}]'$ the vector of prediction standard error for $\hat{\mathbf{x}}_t$. Denote by $\hat{\boldsymbol{\Sigma}}$ the diagonal matrix whose main diagonal is $\hat{\boldsymbol{\sigma}}_t^2$. Let a tilde indicate the reconciled values, so that $\tilde{y}_t$ and $\tilde{\mathbf{x}}_t$ indicate the reconciled values of $\hat{y}_t$ and $\hat{\mathbf{x}}_t$, respectively. The number of child series m can vary with t; however, for simplicity, the number of child series is considered fixed in the following discussion.

At each time t, the values of the series $x_{i,t}$, $i = 1, \ldots, m$, and y_t are bound by an aggregation constraint. For example, if the x_i are the sales at store level for a retail company, then y_t can be either the total sales at company level or the average sales per store. If you specify the AGGREGATE=SUM option in the

PROC TSRECONCILE statement, the aggregation constraint is $y_t = \sum_{i=1}^{m} x_{i,t}$. If instead you specify the AGGREGATE=AVERAGE option, the constraint is $y_t = \frac{1}{m}\sum_{i=1}^{m} x_{i,t}$.

If you need to have forecasts at both levels of the hierarchy, it is often more convenient to produce statistical forecasts separately for each series. However, the resulting forecasts do not abide by the aggregation constraint that binds the original series. The after-the-fact process through which the statistical forecasts are modified to enforce the aggregation constraint is called *reconciliation*.

By determining whether the upper-level forecasts or the lower-level forecasts are adjusted to meet the aggregation constraint, you can distinguish between bottom-up (BU) and top-down (TD) reconciliation, respectively. The TSRECONCILE procedure performs top-down reconciliation between two levels of a hierarchy of forecasts.

Top-Down Reconciliation

The goal of top-down (TD) reconciliation is to adjust the statistical forecasts $\hat{x}_{i,t}$ to obtain a new series $\{\tilde{x}_{i,t}\}$ of reconciled forecasts so that the sum of the reconciled forecasts at each fixed time t is equal to $\hat{y}_t$.

The problem can be restated as follows: minimize with respect to $\tilde{\mathbf{x}}$ a quadratic loss function $L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t)$ subject to the following constraints.

- The *top-down constraint* is

$$\sum_{i=1}^{m} \tilde{x}_{i,t} = \hat{y}_t$$

- The lower bounds are

$$\tilde{x}_{i,t} \geq l_{i,t} \qquad i \in L_t \qquad L_t \in \{1, 2, \ldots, m\}$$

- The upper bounds are

$$\tilde{x}_{i,t} \leq u_{i,t} \qquad i \in U_t \qquad U_t \in \{1, 2, \ldots, m\}$$

Bounds are set by the SIGN= option in the TSRECONCILE statement, which specifies nonpositive or nonnegative bounds on all reconciled forecasts.

PROC TSRECONCILE uses an iterative interior point algorithm to solve the constrained quadratic optimization problem. For more information about this algorithm, see Chapter 17, "The Quadratic Programming Solver" (*SAS Optimization: Mathematical Optimization Procedures*).

Choice of Loss Function

The loss function takes either of the following functional forms, where $\mathbf{W}$ is a positive semidefinite matrix of weights independent of $\tilde{\mathbf{x}}_t$, $\hat{\mathbf{X}}^{-\frac{1}{2}}$ is a diagonal matrix with the square root of $\hat{\mathbf{x}}_t$ on the main diagonal, and $\overline{\hat{\mathbf{X}}^{-\frac{1}{2}}}$ is its complex conjugate:

- When DISAGGREGATION=DIFFERENCE, the loss function is defined as the following:

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = (\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)'\mathbf{W}^{-1}(\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)$$

- When DISAGGREGATION=PROPORTIONS, the loss function is defined as the following only when all $\hat{x}_{i,t}$ are different from 0. The solutions can be extended to the zero cases by defining $\tilde{x}_{i,t} := 0$ when $\hat{x}_{i,t} = 0$ if at least one $\hat{x}_{j,t}$ is different from 0. The case where all $\hat{x}_{j,t}$ are 0 is handled by setting $\tilde{x}_{i,t} := \frac{\hat{y}_t}{m}$ when AGGREGATE=TOTAL and $\tilde{x}_{i,t} := \hat{y}_t$ when AGGREGATE=AVERAGE.

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = (\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)' \hat{\mathbf{X}}^{-\frac{1}{2}} \mathbf{W}^{-1} \overline{\hat{\mathbf{X}}^{-\frac{1}{2}}} (\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)$$

If the WEIGHTED option is not specified in the PROC TSRECONCILE statement, $\mathbf{W}$ is the identity matrix $\mathbf{I}$. If the WEIGHTED option is specified, $\mathbf{W} = \hat{\mathbf{\Sigma}}$, the diagonal matrix with the estimated variances $\hat{\sigma}^2_{i,t}$ of $\hat{x}_{i,t}$ on the main diagonal. The standard errors must be strictly positive.

Unconstrained Solutions

When the only constraint is the top-down constraint, no bounds are present, and the WEIGHTED option is not specified (that is, $\mathbf{W} = \mathbf{I}$), the top-down problem admits intuitive solutions.

When DISAGGREGATION=DIFFERENCE, the loss function becomes

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = \sum_{i=1}^{m} (\hat{x}_{i,t} - \tilde{x}_{i,t})^2$$

This leads to the following solution, where $\hat{r}_t$ is the forecasting aggregate error:

$$\tilde{x}_{i,t} = \hat{x}_{i,t} + \frac{\hat{r}_t}{m}$$

When AGGREGATE=TOTAL,

$$\hat{r}_t := \hat{y}_t - \sum_{i=1}^{m} \hat{x}_{i,t}$$

When AGGREGATE=AVERAGE,

$$\hat{r}_t := m\hat{y}_t - \sum_{i=1}^{m} \hat{x}_{i,t}$$

Thus, when DISAGGREGATION=DIFFERENCE, the reconciled forecast $\tilde{x}_{i,t}$ is found by splitting the aggregation error $\hat{r}_t$ equally among the lower-level forecasts $\hat{x}_{i,t}$.

Notice that even if all statistical forecasts $\hat{x}_{i,t}$ are strictly positive, the reconciled forecasts $\tilde{x}_{i,t}$ are not necessarily positive if no bounds are specified in the SIGN= option. In particular, $\hat{x}_{i,t} = 0$ does not imply $\tilde{x}_{i,t} = 0$.

If DISAGGREGATION=PROPORTIONS, the loss function becomes

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = \sum_{i=1}^{m} \frac{\left(\hat{x}_{i,t} - \tilde{x}_{i,t}\right)^2}{|\hat{x}_{i,t}|}$$

This leads to the following solutions:

$$\tilde{x}_{i,t} = \hat{x}_{i,t} + \frac{|\hat{x}_{i,t}|}{\sum_{j=1}^{m} |\hat{x}_{j,t}|} \hat{r}_t$$

When AGGREGATE=TOTAL and all the $\hat{x}_{j,t}$ have the same sign, the solution resolves to

$$\tilde{x}_{i,t} = \frac{\hat{x}_{i,t}}{\sum_{j=1}^{m} \hat{x}_{j,t}} \hat{y}_t$$

When AGGREGATE=AVERAGE and all the $\hat{x}_{j,t}$ have the same sign, the solution resolves to

$$\tilde{x}_{i,t} = \frac{\hat{x}_{i,t}}{\sum_{j=1}^{m} \hat{x}_{j,t}} m \hat{y}_t$$

Thus, the reconciled forecast $\tilde{x}_{i,t}$ is found by disaggregating the upper-level forecasts according to the proportion that $\hat{x}_{i,t}$ represents in the total sum of the lower-level forecasts.

Missing Values

When one or more of the predicted values are missing, the missing values are replaced by the corresponding actual values that are present. This replacement is done in order to prevent bias between the aggregated and reconciled forecasts, which would result from models in which missing values in the predictions are generated because of the presence of lagged variables. If the corresponding actual value is also missing, the series is excluded from the reconciliation process.

When you use the WEIGHTED option and the standard error is missing, the weight is assumed to be equal to 1.

Confidence Limits

When CLMETHOD=SHIFT, the reconciled confidence limits are computed by re-centering the original confidence limits around the reconciled predicted values.

When CLMETHOD=GAUSS, the reconciled confidence limits are computed by assuming that the series is Gaussian with standard error equal to the prediction standard error.

Input and Output Data Tables

CHILD= Data Table

The CHILD= table must contain the variables that are specified in the BY statement, the variable that is specified in the ID statement, and the PREDICT variable, which specifies the predicted values.

The following variables can optionally be present in the CHILD= data table and are used when available. If not present, their value is assumed to be missing for computational purposes. If these variables are present in the CHILD= data table, they will be present also in the OUTFOR= data table. If the _NAME_ variable is present in both the CHILD= and PARENT= data tables, it is automatically used as a BY variable.

NAME	variable name
ACTUAL	actual values
LOWER	lower confidence limits
UPPER	upper confidence limits
ERROR	prediction errors
STD	prediction standard errors

Typically, the CHILD= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure.

You can specify custom names for the variables in the CHILD= data table by using the CHILDROLES statement. For more information, see the section "CHILDROLES Statement" on page 181.

PARENT= Data Table

The PARENT= table must contain a proper subset (possibly empty) of the variables that are specified in the BY statement, the variable specified in the ID statement, and the PREDICT variable, which specifies the predicted values.

Typically, the PARENT= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure.

You can specify custom names for the variables in the PARENT= table by using the PARENTROLES statement. For more information, see the section "PARENTROLES Statement" on page 182.

OUTFOR= Data Table

The OUTFOR= table contains the BY variables that are specified in the BY statement, the variable that is specified in the ID statement, the ERROR variable, which specifies the prediction errors, and the following variables if they are present in the CHILD= data table:

NAME	variable name
ACTUAL	actual values
PREDICT	predicted values
LOWER	lower confidence limits
UPPER	upper confidence limits
STD	prediction standard errors

Examples: TSRECONCILE Procedure

Example 12.1: Reconciling a Hierarchical Tree

The TSRECONCILE procedure reconciles forecasts between two levels of a hierarchy. It can also be used recursively for reconciling the entire hierarchy.

This example continues the example in the section "Getting Started: TSRECONCILE Procedure" on page 175. Consider the hierarchy structure for the Sashelp.Pricedata data set outlined in Figure 12.1. You can reconcile the hierarchy top down, starting from the top level 0 down to the bottom level 2. At each new iteration, the OUTFOR= data table of the previous reconciliation step becomes the PARENT= data table of the current step.

First, you need to compute the statistical forecasts for all levels. The statistical forecasts for level 1 and level 2 were already computed in the section "Getting Started: TSRECONCILE Procedure" on page 175, so only the forecasts at the company levels are left to compute in the following statements. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
 /* Forecast series at company level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_company)
       errorstop=yes
       ;
require atsm;
id date interval=month;
var sale;
submit;
  declare object t(tsdf);
  declare object f(foreng);
  declare object outfor(outfor);
  rc = t.Initialize(); %rccheck;
  rc = t.AddY(sale); %rccheck;
  rc = f.Initialize(t); %rccheck;
  rc = f.SetOption('lead',12); %rccheck
  rc = f.Run(); %rccheck;
  rc = outfor.Collect(f); %rccheck;
endsubmit;
run;
```

First, you reconcile the top and region levels. The output data table recfor_region (from the example in the section "Getting Started: TSRECONCILE Procedure" on page 175) contains the reconciled forecasts at level 1. This data table becomes the PARENT= data table for the next step of reconciliation, which involves level 1 and level 2.

```
 /* Reconcile forecasts top down from company to region */
proc tsreconcile
      parentTable = mycas.outfor_company
      childTable = mycas.outfor_region
      outfor = mycas.recfor_region
      ;
   by region;
   id date;
run;

/* Reconcile forecasts top down from region to region/product */
proc tsreconcile
      parentTable = mycas.recfor_region
      childTable = mycas.outfor_region_product
      outfor = mycas.recfor_region_product
      ;
   by region product;
   id date;
run;
```

The output data tables mycas.recfor_region and mycas.recfor_region_product contain the reconcile forecasts for the hierarchy at levels 1 and 2, respectively.

Subject Index

Syntax Index