SAS® High-Performance Analytics 1.4
User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2012. *SAS® High-Performance Analytics 1.4: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® High-Performance Analytics 1.4: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, March 2012

Contents

Credits and Acknowledgments

Credits

Documentation

Editing	Anne Baxter
Documentation Support	Tim Arnold, Melanie Gratton

Software

The SAS High-Performance Analytics procedures and their foundation were implemented by the following members of the development staff. Program development includes design, programming, debugging, support, and documentation. In the following list, the names of the developers who currently provide primary support are listed first; other developers and previous developers are also listed.

HPCOUNTREG	Richard Potter
HPDMDB	Scott Pope
HPDS2	Mark Freskos, Tom P. Weber, Oliver Schabenberger
HPFOREST	Ashok Savasere, Padraic Neville
HPLMIXED	Tianlin Wang
HPLOGISTIC	Robert E. Derr, Oliver Schabenberger
HPNEURAL	Larry Lewis
HPNLIN	Marc Kessler, Oliver Schabenberger
HPREDUCE	Alan (Zheng) Zhao
HPREG	Robert Cohen
HPSAMPLE	Ye Liu
HPSEVERITY	Mahesh Joshi
HPSUMMARY	Gordon Keener

High-Performance Computing Foundation	Steve E. Krueger
High-Performance Analytics Foundation	Georges H. Guirguis, Oliver Schabenberger
Numerical Routines	Georges H. Guirguis

The following people contribute to SAS High-Performance Analytics with their leadership and support: Chris Bailey, Tanya Balan, David Pope, Oliver Schabenberger, Renee Sciortino.

Testing

Shu An, Leslie Anderson, Jack Berry, Keith Carter, Ming Chun-Chang, Melissa Corn, John Crouch, Enzo D'Andreti, Bruce Elsheimer, Alex Fang, Girija Gavankar, Greg Goodwin, Sanggohn Han, Lisa Hemmerle, Dright Ho, Gerardo Hurtado, Seungho Huh, Nilesh Jakhotiya, Jagruti Kanjia, Cheryl LeSaint, Yu Liang, Jim McKenzie, Jim Metcalf, Huiping Miao, Kelly Miller, Phil Mohr, Bengt Pederson, Jeff Prevatt, Kim Stott, Benita Taylor, Weihua Shi.

Technical Support

Phil Gibbs

Acknowledgments

Many people make significant and continuing contributions to the development of SAS software products.

The final responsibility for the SAS System lies with SAS alone. We hope that you will always let us know your opinions about the SAS System and its documentation. It is through your participation that SAS software is continuously improved.

Please visit us at http://www.sas.com/software/high-performance-computing/.

Chapter 1
Introduction

Contents

Overview of SAS High-Performance Analytics

SAS High-Performance Analytics provides tools for analytic tasks in a high-performance environment that is characterized by massively parallel processing (MPP) and symmetric multiprocessing (SMP) on a distributed database system. With SAS High-Performance Analytics you can perform analyses that range from descriptive statistics and summarization to correlation analysis, generalized linear modeling, variable selection, neural networks, and maximum likelihood in nonlinear models.

SAS High-Performance Analytics uses a computing appliance that houses a massively parallel database system to manage data in distributed form and to perform the computations in parallel. A computing appliance is a dedicated hardware and software environment that provides computing resources in a client-server model. You are connected indirectly to the appliance through the network connection between the client machine and the computing appliance. Software instructions on the client machine are translated into and deferred to software execution on the appliance.

The SAS High-Performance Analytics appliance provides a massively parallel computing environment supported by Message Passing Interface (MPI) combined with a massively parallel distributed database management system (Teradata or EMC Greenplum) on a x64-Linux platform.

You can also operate SAS High-Performance Analytics procedures in a local SMP mode, where software execution does not engage the appliance and multithreaded analytics are executed on the client machine that runs the SAS session.

SAS High-Performance Analytics procedures do not require access to distributed data, but in many situations performance can be improved dramatically when data are read or written in distributed form.

The software is constantly being updated to reflect new methodology and advances in high-performance analytics.

About This Book

Since High-Performance Analytics software is a part of the SAS System, this book assumes that you are familiar with Base SAS software and with the books *SAS Language Reference: Concepts* and the *Base SAS Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts such as creating SAS data sets with the DATA step and manipulating SAS data sets with Base SAS procedures (for example, the PRINT and SORT procedures).

Chapter Organization

This book is organized as follows.

Chapter 1, this chapter, provides an overview of SAS High-Performance Analytics and summarizes related information, products, and services.

Chapter 2, "Shared Concepts and Topics," provides information about topics that are common to all High-Performance Analytics procedures, such as how these procedures differ from other procedures in processing input and output data sets and how to navigate the distributed computing environment. The chapter also covers syntax that is common among many High-Performance Analytics procedures, such as the CLASS, ID, FREQ, SELECTION, VAR, PERFORMANCE, and WEIGHT statements. Any differences in a procedure's implementation of the common syntax elements is noted in the specific procedure chapter.

Subsequent chapters describe the procedures that make up SAS High-Performance Analytics. These chapters appear in alphabetical order by procedure name. Each chapter is organized as follows:

- The "Overview" section provides a brief description of 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 miscellaneous details, such as ODS tables.
- The “Examples” section contains examples that use the procedure.
- The “References” section contains references for the methodology and for examples of the procedure.

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 the 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 for user-supplied values for options in the syntax definitions and in text.
VariableName	is used for the names of variables and data sets when they appear in the text.
bold	is used to refer to matrices and vectors.
italic	is used for terms that are defined in the text, for emphasis, and for references to publications.
`monospace`	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

Output of Examples

Most of the output shown in this book is produced with the following SAS System options:

```
options linesize=80 pagesize=500 nonumber nodate;
```

The HTMLBLUE style is used to create the HTML output and graphs that appear in the online documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. The style template is specified in the ODS HTML statement as follows:

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

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

Where to Turn for More Information

This section describes other sources of information about SAS High-Performance Analytics.

SAS Technical Support Services

As with all SAS products, the SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of SAS High-Performance Analytics. Go to `http://support.sas.com/techsup` for more information.

Related SAS Software

Many features not found in SAS High-Performance Analytics are available in other parts of the SAS System. If you do not find something you need in SAS High-Performance Analytics, try looking for the feature in the following SAS software products.

SAS/IML® Software

SAS/IML software gives you access to a powerful and flexible programming language (interactive matrix language) in a dynamic, interactive environment. The fundamental object of the language is a data matrix. You can use SAS/IML software interactively (at the statement level) to see results immediately, or you can store statements in a module and execute them later. The programming is dynamic because necessary activities such as memory allocation and dimensioning of matrices are done automatically. SAS/IML enables you to program your methods in the SAS System. For more information, see the *SAS/IML User's Guide*.

SAS/IML® Studio

SAS/IML Studio is a tool for data exploration and analysis; it provides a highly flexible programming environment in which you can run SAS High-Performance Analytics or SAS/IML analyses and display the results with dynamically linked graphics and data tables. You can also move seamlessly between interactive analysis and programmatically driven analysis. SAS/IML Studio is intended for data analysts who write SAS programs to solve statistical problems but need more versatility for data exploration and model building.

The programming language in SAS/IML Studio, which is called *IMLPlus*, is an enhanced version of the SAS/IML programming language. IMLPlus extends SAS/IML by providing features such as the ability to create and manipulate dynamically linked graphs and the ability to call SAS procedures.

SAS/IML Studio also includes an interface to the R language. The IMLPlus language provides functions that transfer data between SAS data sets and R data frames, and between SAS/IML matrices and R matrices.

SAS/IML Studio runs on a PC in the Microsoft Windows operating environment. For more information about SAS/IML Studio, see the *SAS/IML Studio: User's Guide* and *SAS/IML Studio for SAS/STAT Users*.

Base SAS® Software

The features provided by SAS High-Performance Analytics are in addition to the features provided by Base SAS software. Many data management and reporting capabilities that you need are part of Base SAS software. Refer to *SAS Language Reference: Concepts* and the *Base SAS Procedures Guide* for documentation of Base SAS software.

Base SAS software provides the following:

ODS Graphics

- The statistical graphics (SG) family of procedures provides a simple syntax for creating stand-alone statistical graphics. These procedures include SGPLOT, SGSCATTER, and SGPANEL, which provide a simple and convenient syntax for producing many types of displays. They are particularly convenient for exploring and presenting data. See the *SAS ODS Graphics: Procedures Guide* for more information.
- The GTL (graph template language) and the SGRENDER procedure provide a powerful syntax for creating customized graphs. See the *SAS Graph Template Language: User's Guide* and the *SAS Graph Template Language: Reference Guide* for more information. You can also use the GTL to modify the SAS templates that are provided for use with SAS/STAT procedures.
- The ODS Graphics Editor enables you to make immediate changes to ODS Graphics by using a point-and-click interface. See the *SAS ODS Graphics Editor: User's Guide* for more information.

SAS DATA Step

The DATA step is your primary tool for reading and processing data in the SAS System. The DATA step provides a powerful general purpose programming language that enables you to perform all kinds of data processing tasks. The DATA step is documented in *SAS Language Reference: Concepts*.

Base SAS Procedures

Base SAS software includes many useful SAS procedures. Base SAS procedures are documented in the *Base SAS Procedures Guide*. The following is a list of Base SAS procedures you might find useful:

CORR	computes correlations.
RANK	computes rankings or order statistics.
STANDARD	standardizes variables to a fixed mean and variance.
MEANS	computes descriptive statistics and summarizes or collapses data over cross sections.
TABULATE	prints descriptive statistics in tabular format.
UNIVARIATE	computes descriptive statistics.

SAS/ETS® Software

SAS/ETS software provides SAS procedures for econometrics and time series analysis. It includes capabilities for forecasting, systems modeling and simulation, seasonal adjustment, and financial analysis and reporting. In addition, SAS/ETS software includes an interactive time series forecasting system. For more information, see the *SAS/ETS User's Guide*.

SAS/GRAPH® Software

SAS/GRAPH software includes procedures that create two- and three-dimensional plots and charts. For more information, see *SAS/GRAPH: Reference*.

SAS/OR® Software

SAS/OR software provides operations research capabilities that include optimization, discrete-event simulation, and project and resource scheduling. Most SAS/OR features are provided via procedures. In addition, SAS Simulation Studio is a Windows-based Java application for discrete-event simulation. SAS/OR capabilities include:

- building and solving optimization models (including linear, mixed integer, quadratic, and general nonlinear optimization) with a powerful algebraic modeling language and solver suite
- building, running, and analyzing the results of discrete-event simulation models in a graphical modeling environment
- creating project and resource schedules
- creating Gantt chart and network diagram views of projects and their schedules
- monitoring the progress of projects via macros for computing and graphically displaying earned value management metrics

- using finite-domain constraint programming to find feasible solutions to constraint satisfaction problems (both general and scheduling-oriented)

SAS High-Performance Analytics users might be particularly interested in the diverse SAS/OR mathematical optimization capabilities, which encompass a broad range of problem types and solution algorithms and include partially and fully customized solution methods. For more information, see *SAS/OR User's Guide: Mathematical Programming*, *SAS/OR User's Guide: Constraint Programming*, *SAS/OR User's Guide: Local Search Optimization*, *SAS/OR User's Guide: Project Management*, and *SAS Simulation Studio User's Guide*.

SAS/QC® Software

SAS/QC software provides a variety of procedures for statistical quality control and quality improvement, including procedures for the following:

- Shewhart control charts
- cumulative sum control charts
- moving average control charts
- process capability analysis
- Ishikawa diagrams
- Pareto charts
- experimental design

For more information, see the *SAS/QC User's Guide*.

SAS/QC software also includes the ADX interface for experimental design. For more information, see *Getting Started with the SAS ADX Interface for Design of Experiments*.

SAS/STAT® Software

SAS/STAT software provides comprehensive statistical tools for a wide range of statistical analyses, including analysis of variance, categorical data analysis, cluster analysis, multiple imputation, multivariate analysis, nonparametric analysis, power and sample size computations, psychometric analysis, regression, survey data analysis, and survival analysis. Other examples include linear and nonlinear mixed models, generalized linear models, correspondence analysis, and robust regression. For more information, see the *SAS/STAT User's Guide*.

The syntax of many SAS High-Performance Analytics procedures closely follows their counterparts in SAS/STAT software. The algorithms in High-Performance Analytics are based to some extent on the statistical algorithms in SAS/STAT procedures. However, the algorithms in High-Performance Analytics procedures are highly threaded for concurrent execution and have been adapted for a distributed computing environment. Differences in numerical operations and choices for defaults, that are motivated by large-data problems can cause numerical differences between High-Performance Analytics and SAS/STAT procedures.

SAS® Enterprise Miner™ Software

SAS Enterprise Miner software provides a comprehensive set of tools for building predictive and descriptive data mining models for deployment to production scoring environments. The bundle includes both a unique set of SAS procedures and an interactive client interface. The procedures cover a broad range of prediction, classification, decision analysis, clustering, and market basket models. The client interface provides a complete set of interactive tools to help users build repeatable processes and reports. The client bundle can be deployed on the desktop or via a network and integrates with the SAS BI Infrastructure. The entire model-scoring process is saved as score code that can be deployed to a wide range of production scoring systems. For more information, see the SAS Enterprise Miner Reference Help.

Chapter 2

Shared Concepts and Topics

Contents

This chapter discusses topics that are common to all SAS High-Performance Analytics procedures. It contains sections about general procedure features and behavior and sections about common syntax elements. Any deviation by a High-Performance Analytics procedure from the common syntax, which is documented in the section "Syntax" on page 20, is documented in the specific chapter for the procedure later in this book.

SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures

High-Performance Analytics procedures behave in many ways like other procedures in the SAS System. However, their ability to use a computing appliance, their focus on high performance, and their ability to perform computations in parallel lead to some important differences in features and behavior.

How a specific procedure differs from a procedure with similar functionality in other SAS products (for example, how the HPREG procedure differs from the REG procedure in SAS/STAT) is discussed in the individual procedure chapters. This section focuses on High-Performance Analytics procedures in general.

SMP and MPP Modes

Symmetric multiprocessing (SMP) is a computing model in which multiple processors or multiple cores are controlled by a single operating system and can access shared resources, such as disks and memory. In this book, SMP mode refers to an application running multiple concurrent threads on a multicore machine in order to take advantage of parallel execution on multiple processing units. Even more simply, SMP mode for High-Performance Analytics applications means multithreading on the client machine.

All High-Performance Analytics procedures are capable of running in SMP mode, and this is the default mode when a procedure executes on the client machine. The number of concurrent threads is determined by the procedure based on the number of CPUs (cores) on the machine. High-Performance Analytics procedures use different methods to map core count to the number of concurrent threads, depending on the analytic task. Using one thread per core is not uncommon for the procedures that implement data-parallel algorithms.

Massively parallel processing (MPP) is a computing model in which several nodes in a distributed computing environment participate in the calculations. In this book, the MPP mode of a High-Performance

Analytics procedure refers to the procedure performing the analytics on the database management system (DBMS) appliance. MPP mode has two variations:

- The input data for the analytic task are not stored on the appliance but are distributed to the distributed computing environment by the High-Performance Analytics framework during execution of the procedure. This is termed the client-data (or local-data) model of MPP execution.
- The data are stored in the distributed database and are read in parallel from the DBMS; this is termed the alongside-the-database model of MPP execution.

Environment Variables

You control the execution mode with environment variables or options in the PERFORMANCE statement in High-Performance Analytics procedures or both.

The important environment variables are the following:

- *grid host*: identifies the domain name system (DNS) or IP address of the appliance node to which the High-Performance Analytics software connects to run in MPP mode.
- *installation location*: identifies the directory in which the SAS High-Performance software is installed on the appliance.
- *data server*: identifies the database server on Teradata appliances as defined in the HOSTS file on the client. This data server is the same entry that you typically specify in the SERVER= entry of a LIBNAME statement for Teradata. For more information about specifying LIBNAME statements for Teradata and other relational databases, see the *SAS/ACCESS Interface* documentation for the specific database.

You can set an environment variable directly from the SAS program by using the OPTION SET= command. For example, the following statements define the three variables for a Teradata appliance:

```
option set=GRIDHOST      ="hpa.sas.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
option set=GRIDDATASERVER="myserver";
```

Alternatively, you can set the parameters in the PERFORMANCE statement in High-Performance Analytics procedures. For example:

```
performance host      ="hpa.sas.com"
            install   ="/opt/TKGrid"
            dataserver="myserver";
```

A specification in the PERFORMANCE statement overrides a specification of an environment variable without resetting its value. An environment variable that is set in the SAS session with an OPTION SET= command remains in effect until it is modified or until the SAS session terminates.

Specifying a data server is necessary only on Teradata systems. Its specification depends on the entries in the (client) HOSTS file. The HOSTS file specifies the server (suffixed by “cop” and a number), and an IP address. For example:

```
 myservercop1  33.44.55.66
```

The key variable that determines whether a High-Performance Analytics procedure executes in SMP or MPP mode is the *grid host*. The installation location and data server are needed to ensure that a connection to the grid host can be made, given that a host is specified. This book assumes that the installation location and data server (if necessary) have been set by your system administrator.

The following sets of SAS statements are functionally equivalent:

```
proc hpreduce;
   var x:;
   performance host="hpa.sas.com";
run;

option set=GRIDHOST="hpa.sas.com";
proc hpreduce;
   var x:;
run;
```

Switching Modes

You can switch between SMP and MPP mode in High-Performance Analytics procedures according to the following rules:

- If no grid host is specified, the analysis is carried out in SMP mode on the client machine that runs the SAS session.
- If a grid host is specified, the default behavior depends on whether the execution is alongside-the-database. If the data are local to the client (that is, not stored in the distributed database on the appliance), you need to specify with the NODES= option in the PERFORMANCE statement the number of nodes on the appliance you want to engage in the analysis. If the procedure executes alongside the database, you do not need to specify the NODES= option.

The following example works through SMP and client-data MPP configurations with a data set of $100,000$ observations simulated from a logistic regression model. The data are generated in the SAS session with this DATA step:

```
data simulate1;
   array _a{8} _temporary_ (0,0,0,1,0,1,1,1);
   array _b{8} _temporary_ (0,0,1,0,1,0,1,1);
   array _c{8} _temporary_ (0,1,0,0,1,1,0,1);
   do obsno=1 to 100000;
      x  = rantbl(1,0.28,0.18,0.14,0.14,0.03,0.09,0.08,0.06);
      a  = _a{x};
      b  = _b{x};
      c  = _c{x};
      x1 = int(ranuni(1)*400);
      x2 = 52 + ranuni(1)*38;
      x3 = ranuni(1)*12;
```

```
         lp = 6. -0.015*(1-a) + 0.7*(1-b) + 0.6*(1-c) + 0.02*x1 -0.05*x2 - 0.1*x3;
         y  = ranbin(1,1,(1/(1+exp(lp))));
         output;
      end;
      drop x lp;
   run;

   proc hplogistic data=simulate1;
      class a b c;
      model y = a b c x1 x2 x3;
   run;
```

Figure 2.1 shows the results from the analysis.

Figure 2.1 Results from Logistic Regression in SMP Mode

```
                          The HPLOGISTIC Procedure

                          Performance Information

                       Execution Mode       On client
                       Number of Threads    4

                             Model Information

            Data Source                 WORK.SIMULATE1
            Response Variable           y
            Class Parameterization      GLM
            Distribution                Binary
            Link Function               Logit
            Optimization Technique      Newton-Raphson with Ridging


                            Parameter Estimates

                                  Standard
      Parameter     Estimate         Error        DF     t Value     Pr > |t|

      Intercept       5.7011        0.2539     Infty       22.45       <.0001
      a 0           -0.01020       0.06627     Infty       -0.15       0.8777
      a 1                  0             .         .           .            .
      b 0             0.7124       0.06558     Infty       10.86       <.0001
      b 1                  0             .         .           .            .
      c 0             0.8036       0.06456     Infty       12.45       <.0001
      c 1                  0             .         .           .            .
      x1             0.01975      0.000614     Infty       32.15       <.0001
      x2            -0.04728      0.003098     Infty      -15.26       <.0001
      x3             -0.1017      0.009470     Infty      -10.74       <.0001
```

The entries in the “Performance Information” table shows that the HPLOGISTIC procedure executes in client (SMP) mode using eight threads which are chosen according to the number of CPUs on the client machine. You can force a certain number of threads on any machine involved in the computations with the NTHREADS option in the PERFORMANCE statement.

The following statements use 10 nodes (in MPP mode) to analyze the data on the appliance; results appear in Figure 2.2.

```
proc hplogistic data=simulate1;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=10;
run;
```

Figure 2.2 Results from Logistic Regression in MPP Mode

```
                         The HPLOGISTIC Procedure

                         Performance Information

                Host Node                         hpa.sas.com
                Execution Mode                    Distributed
                Number of Compute Nodes           10
                Number of Threads per Node        24

                            Model Information

          Data Source                   WORK.SIMULATE1
          Response Variable             y
          Class Parameterization        GLM
          Distribution                  Binary
          Link Function                 Logit
          Optimization Technique        Newton-Raphson with Ridging

                           Parameter Estimates

                                  Standard
       Parameter    Estimate         Error       DF    t Value    Pr > |t|

       Intercept      5.7011        0.2539    Infty      22.45      <.0001
       a 0          -0.01020       0.06627    Infty      -0.15      0.8777
       a 1                 0             .        .          .           .
       b 0            0.7124       0.06558    Infty      10.86      <.0001
       b 1                 0             .        .          .           .
       c 0            0.8036       0.06456    Infty      12.45      <.0001
       c 1                 0             .        .          .           .
       x1            0.01975      0.000614    Infty      32.15      <.0001
       x2           -0.04728      0.003098    Infty     -15.26      <.0001
       x3            -0.1017      0.009470    Infty     -10.74      <.0001
```

The specification of a host causes the “Performance Information” table to display the name of the host node of the appliance. The “Performance Information” table also indicates that the calculations were performed in a distributed environment (MPP on the appliance). Twenty-four threads on each of 10 nodes were used to perform the calculations—for a total of 240 threads.

Another indication of distributed (MPP) execution on the appliance is the following message, which is issued in the SAS Log by all High-Performance Analytics procedures.

```
NOTE: The HPLOGISTIC procedure is executing in the distributed
      computing environment with 10 worker nodes.
```

You can override the presence of a grid host and force the computations into SMP mode by specifying the NODES=0 option in the PERFORMANCE statement:

```
proc hplogistic data=simulate1;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=0;
run;
```

Figure 2.3 shows the results.

Figure 2.3 SMP Mode Despite Host Specification

```
                              The HPLOGISTIC Procedure

                              Performance Information

                          Host Node             hpa.sas.com
                          Execution Mode        On client
                          Number of Threads     4
```

In the analysis shown previously in Figure 2.2, the data set Work.simulate1 is local to the client, and the HPLOGISTIC procedure distributed the data to 10 nodes on the appliance. The High-Performance Analytics framework does not keep these data on the appliance; when the procedure terminates, the in-memory representation of the input data on the appliance is freed.

When the input data set is large, sending client-side data to the appliance is not advisable; data transfer can easily dominate the execution time. In practice, transfer speeds are typically lower than the theoretical limits of the network connection or disk I/O rates. At a transfer rate of 40 megabytes per second, sending a 10-gigabyte data set to the appliance requires more than four minutes. If analytic execution time is in the range of seconds, the "performance" of the process is dominated by data movement.

The alongside-the-database execution model, unique to High-Performance Analytics procedures, enables you to read and write data in distributed form from the database that is installed on the appliance.

Alongside-the-Database Execution

High-Performance Analytics procedures interface with the distributed database management system (DBMS) on the appliance in a unique way. If the input data are stored in the DBMS and the grid host is the appliance that houses the data, High-Performance Analytics procedures create a distributed computing environment in which an analytic process is co-located with the nodes of the DBMS. Data then passes from the DBMS to the analytic process on each node. Instead of data movement across the network and possibly back to the client machine, data passes locally between the processes on each node of the appliance.

Since the analytic processes on the appliance are separate from the database processes, the technique is referred to as the alongside-the-database model in contrast to in-database execution where the analytic code executes in the database process.

In general, with a large amount of input data, you can achieve the best performance characteristics with High-Performance Analytics procedures if execution is alongside the database.

Before you can run alongside the database, you must distribute the data to the appliance. The following statements use the HPDS2 procedure to distribute the data set Work.simulate1 into database mydb on the appliance hpa.sas.com. In this particular example, the appliance houses a Greenplum database.

```
option set=GRIDHOST="hpa.sas.com";
libname applianc greenplm
        server  ="hpa.sas.com"
        user    =XXXXXX
        password=YYYYY
        database=mydb;

proc datasets lib=applianc nolist; delete simulate1;
proc hpds2 data=simulate1
           out =applianc.simulate1(distributed_by='distributed randomly');
  performance commit=10000;
  data DS2GTF.out;
     method run();
        set DS2GTF.in;
     end;
  enddata;
run;
```

If the output table applianc.simulate1 exists, the DATASETS procedure removes the table from the Greenplum database because a DBMS typically does not support replacement operations on tables.

Note that the LIBNAME for the output table points to the appliance. The data set option informs the HPDS2 procedure to distribute the records randomly among the data segments of the appliance. The statements that follow the PERFORMANCE statement are the DS2 program that copies the input data to the output data without further transformations.

The following HPLOGISTIC statements perform the analysis on the appliance in the alongside-the-database mode. These statements are almost identical to the first HPLOGISTIC example in the previous section, which executed in SMP (client) mode.

```
proc hplogistic data=applianc.simulate1;
   class a b c;
   model y = a b c x1 x2 x3;
run;
```

The subtle differences are as follows:

- A grid host has been specified by the OPTION SET command in the previous statements.
- A LIBNAME is used that the High-Performance Analytics procedure identifies as the data source housed on the appliance.

Figure 2.4 shows the results from this analysis. The “Performance Information” table now shows that the execution was not only in distributed (MPP) mode, but also alongside the Greenplum database. The numeric results agree with the previous analyses shown in Figure 2.1 and Figure 2.2.

Figure 2.4 Alongside Greenplum Execution

```
                              The HPLOGISTIC Procedure

                              Performance Information

          Host Node                       hpa.sas.com
          Execution Mode                  Distributed, alongside Greenplum
          Number of Compute Nodes         32
          Number of Threads per Node      24

                                 Model Information

              Data Source                 SIMULATE1
              Response Variable           y
              Class Parameterization      GLM
              Distribution                Binary
              Link Function               Logit
              Optimization Technique      Newton-Raphson with Ridging

                               Parameter Estimates

                                    Standard
        Parameter    Estimate          Error       DF    t Value    Pr > |t|

        Intercept      5.7011         0.2539    Infty      22.45      <.0001
        a 0          -0.01020        0.06627    Infty      -0.15      0.8777
        a 1                 0              .        .          .           .
        b 0            0.7124        0.06558    Infty      10.86      <.0001
        b 1                 0              .        .          .           .
        c 0            0.8036        0.06456    Infty      12.45      <.0001
        c 1                 0              .        .          .           .
        x1            0.01975       0.000614    Infty      32.15      <.0001
        x2           -0.04728       0.003098    Infty     -15.26      <.0001
        x3            -0.1017       0.009470    Infty     -10.74      <.0001
```

If you specify a NODES= option in the PERFORMANCE statement in the alongside-the-database mode, the HPLOGISTIC procedure ignores the NODES= specification. If the data are read alongside the database, the number of compute nodes is determined by the layout of the database and cannot be modified. The particular appliance used in the example consists of 32 nodes (see the “Performance Information” table).

Output Data Sets

In the alongside-the-database mode, the data are read in distributed form, minimizing data movement for best performance. Similarly, when you write output data sets and a High-Performance Analytics procedure executes in MPP mode, the data can be written in parallel into the database.

For example, in the following statements, the HPLOGISTIC procedure executes in distributed mode using eight nodes on the appliance to perform the logistic regression on work.simulate1:

```
proc hplogistic data=simulate1;
   class a b c;
   model y = a b c x1 x2 x3;
   id a;
   output out=applianc.simulate1_out pred=p;
   performance host="hpa.sas.com" nodes=8;
run;
```

The output data set applianc.simulate1_out is written in parallel into the database. Although the data are fed on eight nodes, the database might distribute the data on more nodes.

When a High-Performance Analytics procedure executes in SMP mode, all output objects are created on the client. If the LIBNAME of the output data sets points to the appliance, the data are transferred to the database on the appliance. This can lead to considerable performance degradation compared to execution in MPP mode.

Many procedures in SAS software add the variables from the input data set when an observationwise output data set is created. The assumption of High-Performance Analytics procedures is that the input data sets can be large and contain many variables. For performance reasons, the output data set contains the following:

- variables that are explicitly created by the statement
- variables listed in the ID statement
- distribution keys or hash keys that are transferred from the input data set

Including this information enables you to add to the output data set information necessary for subsequent SQL joins without copying the entire input data set to the output data set.

Working with Formats

You can use SAS formats and user-defined formats with High-Performance Analytics procedures as you can with other procedures in the SAS System. However, because the analytic work is carried out in a distributed environment and might depend on the formatted values of variables, some special handling can improve the efficiency of work with formats.

High-Performance Analytics procedures examine the variables used in an analysis for association with user-defined formats. Any user-defined formats that are found by a procedure are transmitted automatically to the appliance. If you are running multiple High-Performance Analytics procedures in a SAS session and the analysis variables depend on user-defined formats, you can preprocess the formats. This step involves generating an XML stream (a file) of the formats and passing the stream to the High-Performance Analytics procedures.

Suppose that the following formats are defined in your SAS program:

```
proc format;
     value YesNo        1='Yes'        0='No';
     value checkThis    1='ThisisOne'  2='ThisisTwo';
     value $cityChar    1='Portage'    2='Kinston';
run;
```

The next group of SAS statements create the XML stream for the formats in the file *Myfmt.xml*, associate that file with the file reference myxml, and pass the file reference with the FMTLIBXML= option in the PROC HPLOGISTIC statement.

```
filename myxml 'Myfmt.xml';
libname  myxml XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
proc format cntlout=myxml.allfmts;
run;

proc hplogistic data=six fmtlibxml=myxml;
   class wheeze cit age;
   format wheeze best4. cit $cityChar.;
   model wheeze = cit age;
run;
```

Generation and destruction of the stream can be wrapped in convenience macros:

```
%macro Make_XMLStream(name=tempxml);
    filename &name 'fmt.xml';
    libname  &name XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
    proc format cntlout=&name..allfmts;
    run;
%mend;

%macro Delete_XMLStream(fref);
    %let rc=%sysfunc(fdelete(&fref));
%mend;
```

If you do not pass an XML stream to a High-Performance Analytics procedure that supports the FMTLIBXML= option, the procedure generates an XML stream as needed when it is invoked.

Syntax

CLASS Statement

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

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. See the section "Levelization of Classification Variables" on page 33 for details about these mappings.

If a CLASS statement is specified, it must precede the MODEL statement in High-Performance Analytics procedures that support a MODEL statement.

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

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

You can specify the following values for either an *option* or a *global-option*:

DESCENDING

DESC

reverses the sorting order of the classification variable. If both the DESCENDING and ORDER= options are specified, High-Performance Analytics procedures order the categories according to the ORDER= option and then reverse that order.

ORDER=DATA | FORMATTED | INTERNAL

ORDER=FREQ | FREQDATA | FREQFORMATTED | FREQINTERNAL

specifies the sorting order for the levels of classification variables. This ordering determines which parameters in the model correspond to each level in the data. By default, ORDER=FORMATTED. For ORDER=FORMATTED and ORDER=INTERNAL, the sort order is machine-dependent. When ORDER=FORMATTED is in effect for numeric variables for which you have supplied no explicit format, the levels are ordered by their internal values.

The following table shows how High-Performance Analytics procedures interpret values of the ORDER= option.

Value of ORDER=	Levels Sorted By
DATA	Order of appearance in the input data set
FORMATTED	External formatted values, except for numeric variables with no explicit format, which are sorted by their unformatted (internal) values
FREQ	Descending frequency count (levels with more observations come earlier in the order)
FREQDATA	Order of descending frequency count, and within counts by order of appearance in the input data set when counts are tied
FREQFORMATTED	Order of descending frequency count, and within counts by formatted value when counts are tied
FREQINTERNAL	Order of descending frequency count, and within counts by unformatted (internal) value when counts are tied
INTERNAL	Unformatted value

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

REF=*'level'* | *keyword*

REFERENCE=*'level'* | *keyword*

specifies the reference level for PARAM=REFERENCE. For an individual (but not a global) variable REF= *option*, you can specify the *level* of the variable to use as the reference level. Specify the formatted value of the variable if a format is assigned. For a REF= *option* or *global-option*, you can use one of the following *keywords*. The default is REF=LAST.

FIRST designates the first ordered level as reference.

LAST designates the last ordered level as reference.

If you choose a reference level for any CLASS variable, all variables are parameterized in the reference parameterization for computational efficiency. In other words, High-Performance Analytics procedures apply a single parameterization method to all classification variables.

Suppose that the variable temp has three levels with values `'hot'`, `'warm'`, and `'cold'`, and that the variable gender has two levels with values `'M'` and `'F'`. The following statements fit a logistic regression model:

```
proc hplogistic;
   class gender(ref='F') temp;
   model y = gender gender*temp;
run;
```

Both CLASS variables are in reference parameterization in this model. The reference levels are `'F'` for the variable gender and `'warm'` for the variable temp, since the statements are equivalent to the following statements:

```
proc hplogistic;
   class gender(ref='F') temp(ref=last);
   model y = gender gender*temp;
run;
```

SPLIT

requests that the columns of the design matrix that correspond to any effect that contains a split classification variable can be selected to enter or leave a model independently of the other design columns of that effect. This option is specific to the HPREG procedure

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

```
proc hpreg;
   class temp gender / split;
   model y = gender gender*temp;
run;
```

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

```
proc hpreg;
   model y = gender_M gender_F
             gender_M*temp_hot  gender_F*temp_hot
             gender_M*temp_warm gender_F*temp_warm
             gender_M*temp_cold gender_F*temp_cold;
run;
```

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

```
proc hpreg;
   class temp(split) gender;
   model y = gender gender*temp;
run;
```

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

```
proc hpreg;
   class gender;
   model y = gender gender*temp_hot gender*temp_warm gender*temp_cold;
run;
```

The following values can be specified as a *global-option*:

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

If you do not specify the MISSING option, observations with missing values for CLASS variables are removed from the analysis, even if the CLASS variables are not used in the model formulation.

PARAM=*keyword*
specifies the parameterization method for the classification variable or variables. You can specify the following *keywords*:

GLM	a less-than-full-rank reference cell coding. This parameterization is used in, for example, the GLM, MIXED, and GLIMMIX procedures in SAS/STAT.
REFERENCE	a reference cell encoding. You can choose the reference value with a specific variable option in the CLASS statement, or designate the first or last ordered value with a *global-option*. The default is REF=LAST.

The GLM parameterization is the default. See the section "Specification and Parameterization of Model Effects" on page 35 for more information about how parameterization of classification variables affects the construction and interpretation of model effects.

TRUNCATE<**=***n*>
specifies the truncation width of formatted values of CLASS variables when the optional *n* is specified.

If *n* is not specified, the TRUNCATE option requests that classification levels should be determined using no more than the first 16 characters of the formatted values of CLASS variables.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. High-Performance Analytics procedures that support the FREQ statement treat each observation as if it appeared f times, where f is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID *variables* **;**

The ID statement lists one or more variables from the input data set that are transferred to output data sets created by High-Performance Analytics procedures, provided that the output data set contains one (or more) records per input observation. For example, when an OUTPUT statement is used to produce observation-wise scores or prediction statistics, ID variables are added to the output data set.

By default, High-Performance Analytics procedures do not include all variables from the input data set in output data sets. In the following statements, a logistic regression model is fit and then scored. The input and output data are stored in the Greenplum database. The output data set contains three columns (p, account, trans_date) where p is computed during the scoring process and the account and transaction date are transferred from the input data set (High-Performance Analytics procedures also transfer any distribution keys from the input to the output data).

```
libname GPLib greenplm server=gpdca user=XXX password=YYY
              database=ZZZ;
proc hplogistic data=gplib.myData;
   class a b;
   model y = a b x1-x20;
   output out=gplib.scores pred=p;
   id account trans_date;
run;
```

SELECTION Statement

SELECTION < *options* > **;**

High-Performance Analytics procedures that support model selection use the SELECTION statement to control details about the model selection process. This statement is supported in different degrees by the HPREDUCE, HPREG, and HPLOGISTIC procedures. The HPREG procedure supports the most complete set of options.

You can specify the following *options* in the SELECTION statement:

METHOD=NONE | *method*< *method-options* >

specifies the method used to select the model, optionally followed by parentheses enclosing options that are applicable to the specified method. The default selection method (when the METHOD= option is not specified) is METHOD=STEPWISE.

The following *methods* are available and are explained in detail in the section “Methods” on page 44.

NONE	specifies no model selection.
FORWARD	specifies forward selection. This method starts with no effects in the model and adds effects.
BACKWARD	specifies backward elimination. This method starts with all effects in the model and deletes effects.
STEPWISE	specifies stepwise regression. This method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.

FORWARDSWAP specifies forward-swap selection, which is an extension of the forward selection method. Before any addition step, all pairwise swaps of one effect in the model and one effect out of the current model that improve the selection criterion are made. When the selection criterion is R square, this method is the same as the MAXR method in the REG procedure in SAS/STAT software. This method is supported in High-Performance Analytics only by the HPREG procedure.

LAR specifies least angle regression. Like forward selection, this method starts by adding effects to an empty model. The parameter estimates at any step are "shrunk" when compared to the corresponding least squares estimates. If the model contains classification variables, then these classification variables are split. See the SPLIT option in the CLASS statement for details. This method is supported in High-Performance Analytics only by the HPREG procedure.

LASSO adds and deletes parameters based on a version of ordinary least squares where the sum of the absolute regression coefficients is constrained. If the model contains classification variables, then these classification variables are split. See the SPLIT option in the CLASS statement for details. This method is supported in High-Performance Analytics only by the HPREG procedure.

Table 2.1 lists the applicable *method-options* for each of these methods.

Table 2.1 Applicable *method-options* by *method*

method-option	FORWARD	BACKWARD	STEPWISE	FORWARDSWAP	LAR	LASSO
ADAPTIVE					x	x
CHOOSE =	x	x	x		x	x
COMPETITIVE			x			
CRITERION =	x	x	x	x		
FAST		x				
LSCOEFFS					x	x
MAXEFFECTS =	x		x	x	x	x
MAXSTEPS =	x	x	x	x	x	x
MINEFFECTS =		x	x			
SELECT =	x	x	x	x		
SLENTRY =	x		x		x	x
SLSTAY =		x	x			x
STOP =	x	x	x	x	x	x

The syntax of the *method-options* that you can specify in parentheses after the SELECTION= option *method* follows. As described in Table 2.1, not all selection *method-options* are applicable to every SELECTION= *method.*

ADAPTIVE < (GAMMA=*nonnegative number***) >**

requests that adaptive weights be applied to each of the coefficients when METHOD=LASSO. Ordinary least squares estimates of the model parameters are used to form the adaptive weights. You use the GAMMA= option to specify the power transformation that is applied to the parameters in forming the adaptive weights. The default value is GAMMA=1.

CHOOSE=*criterion*

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

COMPETITIVE

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

CRITERION=*criterion*

is an alias for the SELECT option.

FAST

implements the computational algorithm of Lawless and Singhal (1978) to compute a first-order approximation to the remaining slope estimates for each subsequent elimination of a variable from the model. When applied in backward selection, the option essentially leads to approximating the selection process as the selection process of a linear regression model in which the crossproducts matrix equals the Hessian matrix in the full model under consideration. The FAST option is available only when METHOD=BACKWARD in the HPLOGISTIC procedure. It is computationally efficient in logistic regression models because the model is not fit after removal of each effect.

LSCOEFFS

requests a hybrid version of the LAR and LASSO methods, where the sequence of models is determined by the LAR or LASSO algorithm but the coefficients of the parameters for the model at any step are determined by using ordinary least squares.

MAXEFFECTS=*n*

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

MAXSTEPS=*n*

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

MINEFFECTS=*n*

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

SELECT=SL | *criterion*

specifies the criterion that the procedure uses to determine the order in which effects enter or leave at each step of the selection method. The criteria that are supported depend on type of model that is being fit. See the chapter for the relevant High-Performance Analytics procedure for the supported criteria.

The SELECT option is not valid with the LAR and LASSO methods. You can use SELECT=SL to request the traditional approach, where effects enter and leave the model based on the significance level. With other SELECT= criteria, the effect that is selected to enter or leave at any step of the selection process is the effect whose addition to or removal from the current model yields the maximum improvement in the specified criterion.

SLENTRY=*value*

SLE=*value*

specifies the significance level for entry, used when the STOP=SL or SELECT=SL option is in effect. The default is 0.05.

SLSTAY=*value*

SLS=*value*

specifies the significance level for staying in the model, used when the STOP=SL or SELECT=SL option is in effect. The default is 0.05.

STOP=SL | NONE | *criterion*

specifies a criterion that is used to stop the selection process. The criteria that are supported depend on the type of model that is being fit. See the chapter for the relevant High-Performance Analytics procedure for the supported criteria.

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

If you specify STOP=NONE, then the selection process stops if no suitable add or drop candidates can be found or if a size-based limit is achieved. For example, if you specify STOP=NONE MAXEFFECTS=5, then the selection process stops at the first step that produces a model with five effects.

With STOP=SL, selection stops at the step where the significance level of the candidate for entry is greater than the SLENTRY= value for addition steps when METHOD=FORWARD or METHOD=STEPWISE and where the significance level of the candidate for removal is greater than the SLSTAY= value when METHOD=BACKWARD or METHOD=STEPWISE.

If you specify a criterion other than SL with the STOP= option, then the selection process stops if the selection process produces a local extremum of this criterion, or if a size-based limit is achieved. For example, if you specify STOP=AIC MAXSTEPS=5, then the selection process stops before step 5 if the sequence of models has a local minimum of the AIC criterion before step 5. The determination of whether a local minimum is achieved is made on the basis of a stop horizon. The default stop horizon is 3, but you can change this by using the STOPHORIZON= option. If the stop horizon is n and the STOP= criterion at any step is better than the stop criterion at the next n steps, then the selection process terminates.

DETAILS=NONE | SUMMARY | ALL

DETAILS=STEPS< CANDIDATES(ALL | *n*) >

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

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

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

The DETAILS=SUMMARY option produces only the selection summary, stop reason, selection reason, and selected effects tables.

HIERARCHY=NONE | SINGLE | SINGLECLASS

specifies whether and how the model hierarchy requirement is applied. This option also controls whether a single effect or multiple effects are allowed to enter or leave the model in one step. You can specify that only classification effects, or both classification and continuous effects, be subject to the hierarchy requirement. The HIERARCHY= option is ignored unless you also specify one of the following options: METHOD=FORWARD, METHOD=BACKWARD, or METHOD=STEPWISE.

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

The keywords you can specify in the HIERARCHY= option are as follows:

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

SINGLECLASS is the same as HIERARCHY=SINGLE except that only CLASS effects are subject to the hierarchy requirement.

The default value is HIERARCHY=NONE.

SELECTION=NONE | BACKWARD | FORWARD | FORWARDSWAP | STEPWISE | LAR | LASSO
is an alias for the METHOD= option.

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

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a High-Performance Analytics procedure.

With the PERFORMANCE statement, you can also control whether a High-Performance Analytics procedure executes in SMP or MPP mode.

You can specify the following *performance-options* in the PERFORMANCE statement:

COMMIT=*n*
requests that the High-Performance Analytics procedure write periodic updates to the SAS Log when observations are sent from the client to the appliance for distributed processing.

High-Performance Analytics procedures do not have to use input data that are stored in the appliance. You can perform distributed computations regardless of the origin or format of the input data, provided the data are in a format that can be read by the SAS System (for example, because a SAS/ACCESS engine is available).

In the following example, the HPREG procedure performs LASSO variable selection with an input data set stored on the client:

```
proc hpreg data=work.one;
   model y = x1-x500;
   selection method=lasso;
   performance nodes=10 host='mydca' commit=10000;
run;
```

In order to perform the work as requested using 10 nodes on the appliance, the data set Work.One needs to be distributed to the appliance.

High-Performance Analytics procedures send the data in blocks to the appliance. Whenever the number of observations sent exceeds an integer multiple of the COMMIT= size, a SAS Log message is produced. The message indicates the actual number of observations distributed, and not an integer multiple of the COMMIT= size.

CPUCOUNT=ACTUAL | *num*

specifies how many processors the procedure assumes are available on each host in the computing environment. *num* can be any integer from 1 to 256.

CPUCOUNT=ACTUAL sets CPUCOUNT to the number of physical processors available. This number can be less than the physical number of CPUs if the SAS process has been restricted by system administration tools. Setting CPUCOUNT= to a number greater than the actual number of available CPUs might result in reduced performance. This option overrides the CPUCOUNT= SAS system option.

If a High-Performance Analytics procedure executes in SMP mode, this option refers to the client machine of the SAS session. In MPP mode, this option applies to the nodes on the appliance.

DATASERVER=*"name"*

specifies the name of the server on Teradata systems as defined through the *hosts* file and as used in the LIBNAME statement for Teradata. For example, if the *hosts* file defines

```
myservercop1  33.44.55.66
```

as the server for Teradata, then a LIBNAME specification would be as follows:

```
libname TDLib teradata server=myserver user= password= database= ;
```

A PERFORMANCE statement to induce running alongside the Teradata server would specify the following:

```
performance dataserver="myserver";
```

If the DATASERVER= option is specified, it overrides the GRIDDATASERVER environment variable.

DETAILS

requests a table that shows a timing breakdown of the procedure steps.

TIMEOUT=*s*

specifies the timeout in seconds for a High-Performance Analytics procedure to wait for a connection to the appliance and establish a connection back to the client. The default is *s*=120 seconds. If jobs are submitted to the appliance through workload management tools that might suspend access to the appliance for a longer period, you might want to increase the timeout value.

HOST=*"name"*

GRIDHOST=*"name"*

specifies the name of the appliance host in single or double quotes. If the HOST= option is specified, it overrides the value of the GRIDHOST environment variable.

INSTALL=*"name"*

INSTALLLOC=*"name"*

specifies the directory in which the High-Performance Analytics shared libraries are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

NODES=ALL | *n*

NNODES=ALL | *n*

specifies the number of nodes in the distributed computing environment, provided that the data are not processed alongside the database.

If you specify NODES=0, you indicate that you want to process the data in SMP mode on the client machine. If the input data are not alongside the database, this is the default. The High-Performance Analytics procedures then perform the analysis mutlithreaded on the client. For example, the following sets of statements are equivalent:

```
proc hplogistic data=one;
   model y = x;
run;
```

```
proc hplogistic data=one;
   model y = x;
   performance nodes=0;
run;
```

If the data are not read alongside the database, the NODES= option specifies the number of nodes on the appliance involved in the analysis. For example, the following statements perform the analysis in MPP mode using 10 units of work on the appliance identified with the HOST= option:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

If the number of nodes can be modified by the application, you can specify a NODES=*n* option where *n* exceeds the number of physical nodes on the appliance. The High-Performance Analytics software then *oversubscribes* the nodes and associates nodes with multiple units of work. For example, on a system with 16 appliance nodes, the following statements oversubscribe the system by a factor of 3:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=48 host="hpa.sas.com";
run;
```

Generally, it is not advisable to oversubscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depends on the CPU count. You can specify NODES=ALL if you want to use all available nodes on the appliance without oversubscribing the system.

If the data are read alongside the distributed database on the appliance, specifying a nonzero value for the NODES= option has no effect. The number of units of work in the distributed computing environment is then determined by the distribution of the data and cannot be altered. For example, if you are running alongside an appliance with 24 nodes, the NODES= option in the following statements is ignored:

```
libname GPLib greenplm server=gpdca user=XXX password=YYY
              database=ZZZ;
proc hplogistic data=gplib.one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

NTHREADS=*n*

specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, the number of threads are determined based on the number of CPUs on the host on which the analytic computations execute. The algorithm by which a CPU count is converted to a thread count is specific to the High-Performance Analytics procedure. Most procedures create one thread per CPU for the analytic computations.

By default, High-Performance Analytics procedures execute in multiple concurrent threads unless turned off by the NOTHREADS system option or you force single-threaded execution with NTHREADS=1. The largest number that can be specified for *n* is 256. Individual High-Performance Analytics procedures can impose more stringent limits if called for by algorithmic considerations.

You can affect the determination of the CPU count with the CPUCOUNT= option in the PERFORMANCE statement.

NOTE: The SAS system options THREADS | NOTHREADS apply to the client machine on which the SAS High-Performance Analytics procedures execute. They do not apply to the compute nodes in a distributed environment.

VAR Statement

VAR *variable-list* **;**

Some High-Performance Analytics procedures (in particular procedures that do not support a MODEL statement) use a VAR statement to identify numerical variables for the analysis.

WEIGHT Statement

WEIGHT *variable* **;**

The *variable* in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations with nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, all observations used in the analysis are assigned a weight of 1.

Levelization of Classification Variables

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

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

Consider the data on nine observations in Table 2.2. The variable A is integer-valued, and the variable X is a continuous variable with a missing value for the fourth observation. The fourth and fifth columns of Table 2.2 apply two different formats to the variable X.

Table 2.2 Example Data for Levelization

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

By default, levelization of the variables groups the observations by the formatted value of the variable, except for numerical variables for which no explicit format is provided. Numerical variables for which no explicit format is provided are sorted by their internal value. The levelization of the four columns in Table 2.2 leads to the level assignment in Table 2.3.

Table 2.3 Values and Levels

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

The sorting order for the levels of CLASS variables can be affected with the ORDER= option in the CLASS statement.

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

Table 2.4 shows how values of the ORDER= option are interpreted.

Table 2.4 Interpretation of Values of ORDER= Option

Value of ORDER=	Levels Sorted By
DATA	Order of appearance in the input data set
FORMATTED	External formatted value, except for numeric variables with no explicit format, which are sorted by their unformatted (internal) value
FREQ	Descending frequency count (levels with the most observations come first in the order)
INTERNAL	Unformatted value
FREQDATA	Order of descending frequency count, and within counts by order of appearance in the input data set when counts are tied
FREQFORMATTED	Order of descending frequency count, and within counts by formatted value when counts are tied
FREQINTERNAL	Order of descending frequency count, and within counts by unformatted (internal) value when counts are tied

For FORMATTED, FREQFORMATTED, FREQINTERNAL, and INTERNAL values, the sort order is machine-dependent. For more information about sort order, see the chapter on the SORT procedure in the *Base SAS Procedures Guide* and the discussion of BY-group processing in *SAS Language Reference: Concepts*.

When the MISSING option is specified in the CLASS statement, the missing values ('.' for a numeric variable and blanks for a character variable) are included in the levelization and are assigned a level. Table 2.5 displays the results of levelizing the values in Table 2.2 when the MISSING option is in effect.

Table 2.5 Values and Levels with the MISSING Option

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

When the MISSING option is not specified, it is important to understand the implications of missing values for your statistical analysis. When a High-Performance Analytics procedure levelizes the CLASS variables, an observation for which any CLASS variable has a missing value is excluded from the analysis. This is true regardless of whether the variable is used to form the statistical model. Consider, for example, the case in which some observations contain missing values for variable A but the records for these observations are otherwise complete with respect to all other variables in the statistical models. The analysis results from the following statements do not include any observations for which variable A contains missing values, even though A is not specified in the MODEL statement:

```
class A B;
model y = B x B*x;
```

High-Performance Analytics procedures print a "Number of Observations" table that shows the number of observations read from the data set and the number of observations used in the analysis. Pay careful attention to this table—especially when your data set contains missing values—to ensure that no observations are unintentionally excluded from the analysis.

Specification and Parameterization of Model Effects

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

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

or

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

The model matrices **X** and **Z** are formed according to effect construction rules.

Procedures that also have a CLASS statement support the rich set of effects discussed in this section. In order to correctly interpret the results from a statistical analysis, you need to understand how construction (*parameterization*) rules apply to regression-type models, whether these are linear models in the HPREG procedure or generalized linear models in the HPLOGISTIC procedure.

Effects are specified with a special notation that uses variable names and operators. There are two kinds of variables: classification (or CLASS) variables and continuous variables. *Classification variables* can be either numeric or character and are specified in a CLASS statement. See the section "Levelization of Classification Variables" on page 33 for details. An independent variable that is not declared in the CLASS statement is assumed to be *continuous*. For example, the heights and weights of subjects are continuous variables.

Two primary operators (crossing and nesting) are used for combining the variables, and several additional operators are used to simplify effect specification. Operators are discussed in the section "Effect Operators" on page 36.

High-Performance Analytics procedures with a CLASS statement support a general linear model (GLM) parameterization and a reference parameterization for the classification variables. The GLM parameterization is the default for all High-Performance Analytics procedures. See the sections "GLM Parameterization of Classification Variables and Effects" on page 38 and "Reference Parameterization" on page 43 for details.

Effect Operators

Table 2.6 summarizes the operators that are available for selecting and constructing effects. These operators are discussed in the following sections.

Table 2.6 Available Effect Operators

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

Bar and At Sign Operators

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

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

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

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

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

$$\begin{aligned} \mathrm{A \mid B \mid C} &\rightarrow \{\,\mathrm{A \mid B}\,\} \mid \mathrm{C} \\ &\rightarrow \{\,\mathrm{A\ \ B\ \ A{*}B}\,\} \mid \mathrm{C} \\ &\rightarrow \mathrm{A\ \ B\ \ A{*}B\ \ C\ \ A{*}C\ \ B{*}C\ \ A{*}B{*}C} \end{aligned}$$

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

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

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

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

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

More examples of using the bar and at operators follow:

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

Colon, Dash, and Double Dash Operators

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

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

model Y = X1-X9;

model Y = X:;
```

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

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

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

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

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

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

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

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

GLM Parameterization of Classification Variables and Effects

Table 2.7 shows the types of effects that are available in High-Performance Analytics procedures; they are discussed in more detail in the following sections. Let A, B, and C represent classification variables, and let X and Z represent continuous variables.

Table 2.7 Available Types of Effects

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

Table 2.8 shows some examples of MODEL statements that use various kinds of effects.

Table 2.8 Model Statement Effect Examples

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

Intercept

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

Regression Effects

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

Main Effects

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

Table 2.9 is an example where β_0 denotes the intercept and A and B are classification variables with two and three levels, respectively.

Table 2.9 Example of Main Effects

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

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

Interaction Effects

Often a model includes interaction (crossed) effects to account for how the effect of a variable changes with the values of other variables. With an interaction, the terms are first reordered to correspond to the order of the variables in the CLASS statement. Thus, B*A becomes A*B if A precedes B in the CLASS statement. Then, the GLM parameterization generates columns for all combinations of levels that occur in the data. The order of the columns is such that the rightmost variables in the interaction change faster than the leftmost variables (Table 2.10).

In the HPLMIXED procedure, which supports both fixed- and random-effects models, empty columns (that is, columns that would contain all 0s) are not generated for fixed effects, but they are generated for random effects.

Table 2.10 Example of Interaction Effects

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

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

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

Nested Effects

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

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

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

The nesting operator in High-Performance Analytics software is more of a notational convenience than an operation that is distinct from crossing. Nested effects are typically characterized by the property that the nested variables do not appear as main effects. The order of the variables within nesting parentheses is made to correspond to the order of these variables in the CLASS statement. The order of the columns is such that variables outside the parentheses index faster than those inside the parentheses, and the rightmost nested variables index faster than the leftmost variables (Table 2.11).

Table 2.11 Example of Nested Effects

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

Continuous-Nesting-Class Effects

When a continuous variable nests or crosses with a classification variable, the design columns are constructed by multiplying the continuous values into the design columns for the classification effect (Table 2.12).

Table 2.12 Example of Continuous-Nesting-Class Effects

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

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

Continuous-by-Class Effects

Continuous-by-class effects generate the same design columns as continuous-nesting-class effects. Table 2.13 shows the construction of the X*A effect. The two columns for this effect are the same as the columns for the X(A) effect in Table 2.12.

Table 2.13 Example of Continuous-by-Class Effects

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

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

General Effects

An example that combines all the effects is X1*X2*A*B*C(D E). The continuous list comes first, followed by the crossed list, followed by the nested list in parentheses. You should be aware of the sequencing of parameters when you use statements that depend on the ordering of parameters. Such statements include CONTRAST and ESTIMATE statements, which are used in a number of procedures to estimate and test functions of the parameters.

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

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

The sequencing of the parameters generated by an effect can be described by which variables have their levels indexed faster:

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

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

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

then the order of the parameters for the effect B*A(C D), which is renamed A*B(C D), is

$$\begin{array}{llll}
A_1B_1C_1D_1 \rightarrow & A_1B_2C_1D_1 \rightarrow & A_2B_1C_1D_1 \rightarrow & A_2B_2C_1D_1 \rightarrow \\
A_1B_1C_1D_2 \rightarrow & A_1B_2C_1D_2 \rightarrow & A_2B_1C_1D_2 \rightarrow & A_2B_2C_1D_2 \rightarrow \\
A_1B_1C_2D_1 \rightarrow & A_1B_2C_2D_1 \rightarrow & A_2B_1C_2D_1 \rightarrow & A_2B_2C_2D_1 \rightarrow \\
A_1B_1C_2D_2 \rightarrow & A_1B_2C_2D_2 \rightarrow & A_2B_1C_2D_2 \rightarrow & A_2B_2C_2D_2
\end{array}$$

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

Reference Parameterization

Classification variables can be represented in the reference parameterization in High-Performance Analytics procedures. Only one parameterization applies to the variables in the CLASS statement.

To understand the reference representation, consider the classification variable A with four values, 1, 2, 5, and 7. The reference parameterization generates three columns (one less than the number of variable levels). The columns indicate group membership of the nonreference levels. For the reference level, the three dummy variables have a value of 0. If the reference level is 7 (REF='7'), the design columns for variable A are as shown in Table 2.14.

Table 2.14 Reference Coding

A	Design Matrix		
	A1	**A2**	**A5**
1	1	0	0
2	0	1	0
5	0	0	1
7	0	0	0

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

Model Selection

Methods

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

Full Model Fitted

When METHOD=NONE, the complete model specified in the MODEL statement is used to fit the model, and no effect selection is done.

Forward Selection

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

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

An alternative approach to address the critical problem of when to stop the selection process is to assess the quality of the models produced by the forward selection method and choose the model from this sequence that "best" balances goodness of fit against model complexity. There are several criteria that you can use for this purpose. These criteria fall into two groups—information criteria and criteria based on out-of-sample prediction performance.

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

For example, if you specify the following statement, then forward selection terminates at the step where no effect can be added at the 0.2 significance level:

```
selection method=forward(select=SL choose=AIC SLE=0.2);
```

However, the selected model is the first one with the minimal value of Akaike's information criterion. Note that in some cases this minimal value might occur at a step much earlier that the final step, while in other cases the AIC might start increasing only if more steps are done—that is, a larger value is used for the significance level for entry. If you are interested in minimizing AIC, then too many steps are done in the former case and too few in the latter case. To address this issue, the High-Performance Analytics procedures enable you to specify a stopping criterion with the STOP= option. With a stopping criterion specified, forward selection continues until a local extremum of the stopping criterion in the sequence of models generated is reached. To be deemed a local extremum, a criterion value at a given step must be better than its value at the next n steps, where n is known as the "stop horizon." By default, the stop horizon is three steps but you can change this by using the STOPHORIZON= option.

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

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

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

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

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

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

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

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

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

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

It is important to keep in mind that forward selection bases the decision about what effect to add at any step by considering models that differ by one effect from the current model. This search paradigm cannot guarantee reaching a "best" subset model. Furthermore, the add decision is greedy in the sense that the effect deemed most significant is the effect that is added. However, if your goal is to find a model that is best in terms of some selection criterion other than the significance level of the entering effect, then even this one step choice might not be optimal. For example, the effect you would add to get a model with the smallest value of the Mallows' $C(p)$ statistic at the next step is not necessarily the same effect that is most significant based on a hypothesis test. The High-Performance Analytics procedures enable you to specify the criterion to optimize at each step by using the SELECT= option. For example, the following statement requests that at each step the effect that is added be the one that gives a model with the smallest value of the Mallows' $C(p)$ statistic:

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

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

You can use the SELECT= option together with the CHOOSE= and STOP= options. If you specify only the SELECT= criterion, then this criterion is also used as the stopping criterion. In the previous example where only the selection criterion is specified, not only do effects enter based on the Mallows' $C(p)$ statistic, but the selection terminates when the $C(p)$ statistic has a local minimum.

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

Examples of Forward Selection Specifications

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

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

The following statement adds effects based on significance level and stops when all candidate effects for entry at a step have a significance level greater than the default entry significance level of 0.05:

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

The following statement adds effects based on significance level and stops at a step where adding any effect increases the error sum of squares computed on the validation data:

```
selection=forward(select=SL stop=validation) stophorizon=1;
```

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

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

The following statement adds effects that at each step give the largest value of the adjusted R-square statistic and stops at the step where the significance level corresponding to the addition of this effect is greater than 0.2:

```
selection=forward(select=ADJRSQ stop=SL SLE=0.2);
```

Backward Elimination

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

In the traditional implementation of backward selection, the statistic used to determine whether to drop an effect is significance level. At any step, the least significant predictor is dropped and the process continues until all effects that remain in the model are significant at a specified stay significance level (SLS).

Just as with forward selection, you can change the criterion used to assess effect contributions with the SELECT= option. You can also specify a stopping criterion with the STOP= option and use a CHOOSE= option to provide a criterion for selecting among the sequence of models produced. See the discussion in the section "Forward Selection" on page 44 for additional details.

Examples of Backward Selection Specifications

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

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

The following statement removes effects based on significance level and stops when all candidate effects for removal at a step are significant at the default stay significance level of 0.05:

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

The following statement removes effects based on significance level and stops when all effects in the model are significant at the 0.1 level. Finally, from the sequence of models generated, the chosen model is the one that gives the smallest average square error when scored on the validation data:

```
selection method=backward(select=SL choose=validate SLS=0.1);
```

The following statement applies in logistic regression models the fast backward technique of Lawless and Singhal (1978), a first-order approximation that has greater numerical efficiency than full backward selection:

```
selection method=backward(fast);
```

The fast technique fits an initial full logistic model and a reduced model after the candidate effects have been dropped. On the other hand, full backward selection fits a logistic regression model each time an effect is removed from the model.

Stepwise Selection

METHOD=STEPWISE specifies the stepwise method, which is a modification of the forward selection technique that differs in that effects already in the model do not necessarily stay there.

In the traditional implementation of stepwise selection method, the same entry and removal significance levels for the forward selection and backward elimination methods are used to assess contributions of effects as they are added to or removed from a model. If at a step of the stepwise method any effect in the model is not significant at the SLSTAY= level, then the least significant of these effects is removed from the model and the algorithm proceeds to the next step. This ensures that no effect can be added to a model while some effect currently in the model is not deemed significant. Only after all necessary deletions have been accomplished can another effect be added to the model. In this case the effect whose addition is the most significant is added to the model and the algorithm proceeds to the next step. The stepwise process ends when none of the effects outside the model is significant at the SLENTRY= level and every effect in the model is significant at the SLSTAY= level. In some cases, neither of these two conditions for stopping is met and the sequence of models cycles. In this case, the stepwise method terminates at the end of the cycle.

Just as with forward selection and backward elimination, you can use the SELECT= option to change the criterion used to assess effect contributions. You can also specify a stopping criterion with the STOP= option and use a CHOOSE= option to provide a criterion for selecting among the sequence of models produced. See the discussion in the section "Forward Selection" on page 44 for additional details.

For selection criteria other than significance level, the High-Performance Analytics procedures optionally support a further modification in the stepwise method. In the standard stepwise method, no effect can enter the model if removing any effect currently in the model would yield an improved value of the selection criterion. In the modification, you can use the COMPETITIVE option to specify that addition and deletion of effects should be treated competitively. The selection criterion is evaluated for all models obtained by deleting an effect from the current model or by adding an effect to this model. The action that most improves the selection criterion is the action taken.

Examples of Stepwise Selection Specifications

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

```
selection method=stepwise;
```

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

The following statement requests the traditional stepwise method:

```
selection=stepwise(select=SL)
```

First, if the removal of any effect in the model is not significant at the default stay level of 0.05, then the least significant effect is removed and the algorithm proceeds to the next step. Otherwise the effect whose addition is the most significant is added, provided that it is significant at the default entry level of 0.05.

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

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

In this case, the entry and stay significance levels still play a role as they determine whether an effect is deleted from or added to the model. This extra check might result in the selection terminating before a local minimum of the SBC criterion is found.

The following statement selects effects to enter or drop as in the previous example except that the significance level for entry is now 0.1 and the significance level to stay is 0.08. From the sequence of models produced, the selected model is chosen to yield the minimum AIC statistic:

```
selection method=stepwise(select=SL SLE=0.1 SLS=0.08 choose=AIC);
```

The following statement requests stepwise selection based on the AICC criterion with additions and deletions treated competitively:

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

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

The following statement requests stepwise selection based on the SBC criterion with additions and deletions treated competitively and where stopping is based on the average square error over the validation data:

```
selection=stepwise(select=SBC competitive stop=VALIDATE);
```

At any step, SBC statistics that correspond to the removal of any effect from the current model or the addition of any effect to the current model are evaluated. The addition or removal that produces the minimum SBC value is made. The average square error on the validation data for the model with this addition or removal is evaluated. The selection stops when the average square error so produced increases for three consecutive steps.

Forward-Swap Selection

METHOD=FORWARDSWAP specifies the forward-swap selection method, which is an extension of the forward selection method. The forward-swap selection method incorporates steps that improve a model by replacing an effect in the model with an effect that is not in the model. When the model selection criterion is R square, this method is the same as the maximum R-square improvement (MAXR) method that is implemented in the REG procedure in SAS/STAT software. You cannot use the effect significance level as the selection criterion for the forward-swap method.

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

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

Least Angle Regression

METHOD=LAR specifies least angle regression (LAR), which is supported in the HPREG procedure. LAR was introduced by Efron et al. (2004). Not only does this algorithm provide a selection method in its own right, but with one additional modification it can be used to efficiently produce LASSO solutions. Just like the forward selection method, the LAR algorithm produces a sequence of regression models in which one parameter is added at each step, terminating at the full least squares solution when all parameters have entered the model.

The algorithm starts by centering the covariates and response and scaling the covariates so that they all have the same corrected sum of squares. Initially all coefficients are zero, as is the predicted response. The predictor that is most correlated with the current residual is determined, and a step is taken in the direction of this predictor. The length of this step determines the coefficient of this predictor and is chosen so that some other predictor and the current predicted response have the same correlation with the current residual. At this point, the predicted response moves in the direction that is equiangular between these two predictors. Moving in this direction ensures that these two predictors continue to have a common correlation with the current residual. The predicted response moves in this direction until a third predictor has the same correlation with the current residual as the two predictors already in the model. A new direction is determined that is equiangular among these three predictors, and the predicted response moves in this direction until a fourth predictor, which has the same correlation with the current residual, joins the set. This process continues until all predictors are in the model.

As with other selection methods, the issue of when to stop the selection process is crucial. You can specify a criterion for choosing among the models at each step with the CHOOSE= option. You can also specify a stopping criterion with the STOP= option. These formulas use the approximation that at step k of the LAR algorithm, the model has k degrees of freedom. See Efron et al. (2004) for a detailed discussion of this so-called simple approximation.

A modification of LAR selection suggested in Efron et al. (2004) uses the LAR algorithm to select the set of covariates in the model at any step, but it uses ordinary least squares regression with just these covariates to obtain the regression coefficients. You can request this hybrid method by specifying the LSCOEFFS suboption of METHOD=LAR.

Lasso Selection

Method=LASSO specifies the least absolute shrinkage and selection operator (LASSO) method, which is supported in the HPREG procedure. LASSO arises from a constrained form of ordinary least squares regression where the sum of the absolute values of the regression coefficients is constrained to be smaller than a specified parameter. More precisely let $X = (x_1, x_2, \ldots, x_m)$ denote the matrix of covariates and let y denote the response, where the x_is have been centered and scaled to have unit standard deviation and mean zero and y has mean zero. Then for a given parameter t, the LASSO regression coefficients $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ are the solution to the following constrained optimization problem:

$$\text{minimize}||y - X\beta||^2 \qquad \text{subject to} \quad \sum_{j=1}^{m} |\beta_j| \leq t$$

Provided that the LASSO parameter t is small enough, some of the regression coefficients are exactly zero. Hence, you can view the LASSO as selecting a subset of the regression coefficients for each LASSO parameter. By increasing the LASSO parameter in discrete steps, you obtain a sequence of regression coefficients in which the nonzero coefficients at each step correspond to selected parameters.

Early implementations (Tibshirani 1996) of LASSO selection used quadratic programming techniques to solve the constrained least squares problem for each LASSO parameter of interest. Later Osborne, Presnell, and Turlach (2000) developed a "homotopy method" that generates the LASSO solutions for all values of t. Efron et al. (2004) derived a variant of their algorithm for least angle regression that can be used to obtain a sequence of LASSO solutions from which all other LASSO solutions can be obtained by linear interpolation. This algorithm for METHOD=LASSO is used in PROC HPREG. It can be viewed as a stepwise procedure with a single addition to or deletion from the set of nonzero regression coefficients at any step.

As with the other selection methods supported by the High-Performance Analytic procedures, you can specify a criterion to choose among the models at each step of the LASSO algorithm with the CHOOSE= option. You can also specify a stopping criterion with the STOP= option. See the discussion in the section "Forward Selection" on page 44 for additional details. The model degrees of freedom PROC GLMSELECT uses at any step of the LASSO are simply the number of nonzero regression coefficients in the model at that step. Efron et al. (2004) cite empirical evidence for doing this but do not give any mathematical justification for this choice.

A modification of LASSO selection suggested in Efron et al. (2004) uses the LASSO algorithm to select the set of covariates in the model at any step, but it uses ordinary least squares regression with just these covariates to obtain the regression coefficients. You can request this hybrid method by specifying the LSCOEFFS suboption of SELECTION=LASSO.

Adaptive Lasso Selection

Adaptive lasso selection is a modification of lasso selection; in adaptive lasso selection, weights are applied to each of the parameters in forming the lasso constraint (Zou, 2006). More precisely, suppose that the response y has mean zero and the regressors x are scaled to have mean zero and common standard deviation. Furthermore, suppose that you can find a suitable estimator $\hat{\beta}$ of the parameters in the true model and you define a weight vector by $w = 1/|\hat{\beta}|^{\gamma}$, where $\gamma \geq 0$. Then the adaptive lasso regression coefficients $\beta = (\beta_1, \beta_2, \ldots, \beta_m)$ are the solution to the following constrained optimization problem:

$$\text{minimize}||y - X\beta||^2 \qquad \text{subject to} \quad \sum_{j=1}^{m} |w_j \beta_j| \leq t$$

PROC HPREG uses the solution to the unconstrained least squares problem as the estimator $\hat{\beta}$. This is appropriate unless collinearity is a concern. If the regressors are collinear or nearly collinear, then Zou (2006) suggests using a ridge regression estimate to form the adaptive weights.

References

Burnham, K. P., and Anderson, D. R. (2002), *Model Selection and Multimodel Inference,* Second Edition, New York: Springer-Verlag.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004), “Least Angle Regression (with Discussion),” *Annals of Statistics*, 32, 407–499.

Lawless, J. F. and Singhal, K. (1978), “Efficient Screening of Nonnormal Regression Models,” *Biometrics*, 34, 318–327.

Osborne, M., Presnell, B., and Turlach, B. (2000), “A New Approach to Variable Selection in Least Squares Problems,” *IMA Journal of Numerical Analysis*, 20, 389–404.

Searle, S. R. (1971), *Linear Models,* New York: John Wiley & Sons.

Tibshirani, R. (1996), “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society Series B*, 58, 267–288.

Zou, H. (2006), “The Adaptive Lasso and Its Oracle Properties,” *Journal of the American Statistical Association*, 101, 1418–1429.

Chapter 3
The HPCOUNTREG Procedure

Contents

Overview: HPCOUNTREG Procedure

The HPCOUNTREG procedure is a high-performance version of the COUNTREG procedure in SAS/ETS software. Similar to the COUNTREG procedure, the HPCOUNTREG procedure fits regression models in which the dependent variable takes on nonnegative integer or count values. Unlike the COUNTREG procedure which can be run only on an individual workstation, the HPCOUNTREG procedure takes advantage of a computing environment that enables it to distribute the optimization task among one or more nodes. In addition, each node can use one or more threads to carry out the optimization on its subset of the data. When several nodes are employed with each node using several threads to carry out its part of the work, the result is a highly parallel computation that provides a dramatic gain in performance.

With the HPCOUNTREG procedure you can read and write data in distributed form and perform analyses in massively parallel processing (MPP) and symmetric multiprocessing (SMP) mode. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details about how to affect the execution mode of SAS High-Performance Analytics procedures.

The HPCOUNTREG procedure is specifically designed to operate in the high-performance distributed environment. By default, PROC HPCOUNTREG performs computations in multiple threads.

PROC HPCOUNTREG Features

The HPCOUNTREG procedure estimates the parameters of a count regression model by maximum likelihood techniques. The following list summarizes some basic features of the HPCOUNTREG procedure:

- can perform analysis on a massively parallel high-performance appliance
- reads input data in parallel and writes output data in parallel when the data source is the appliance database
- is highly multithreaded during all phases of analytic execution
- performs maximum likelihood estimation
- supports multiple link functions

Currently the HPCOUNTREG procedure does not support BY-group processing or the CLASS statement.

Getting Started: HPCOUNTREG Procedure

Except for its ability to operate in the high-performance distributed environment, the HPCOUNTREG procedure is similar in use to other regression model procedures in the SAS System. For example, the following statements are used to estimate a Poisson regression model:

```
proc hpcountreg data=one ;
   model y = x / dist=poisson ;
run;
```

The response variable *y* is numeric and has nonnegative integer values.

This section illustrates two simple examples that use PROC HPCOUNTREG. The data are taken from Long (1997). This study examines how factors such as gender (fem), marital status (mar), number of young children (kid5), prestige of the graduate program (phd), and number of articles published by a scientist's mentor (ment) affect the number of articles (art) published by the scientist.

The first 10 observations are shown in Figure 3.1.

Figure 3.1 Article Count Data

```
Obs     art     fem     mar     kid5       phd         ment

  1      3       0       1        2      1.38000     8.0000
  2      0       0       0        0      4.29000     7.0000
  3      4       0       0        0      3.85000    47.0000
  4      1       0       1        1      3.59000    19.0000
  5      1       0       1        0      1.81000     0.0000
  6      1       0       1        1      3.59000     6.0000
  7      0       0       1        1      2.12000    10.0000
  8      0       0       1        0      4.29000     2.0000
  9      3       0       1        2      2.58000     2.0000
 10      3       0       1        1      1.80000     4.0000
```

The following SAS statements estimate the Poisson regression model. The model is executed in the distributed computing environment with two threads and four nodes.

```
/*-- Poisson Regression --*/
proc hpcountreg data=long97data;
   model art = fem mar kid5 phd ment / dist=poisson method=quanew;
   performance nthreads=2 nodes=4 details;
run;
```

The "Model Fit Summary" table shown in Figure 3.2, lists several details about the model. By default, the HPCOUNTREG procedure uses the Newton-Raphson optimization technique. The maximum log-likelihood value is shown, in addition to two information measures—Akaike's information criterion (AIC) and Schwarz's Bayesian information criterion (SBC)—which can be used to compare competing Poisson models. Smaller values of these criteria indicate better models.

Figure 3.2 Estimation Summary Table for a Poisson Regression

```
                    The HPCOUNTREG Procedure

                       Model Fit Summary

          Dependent Variable                          art
          Number of Observations                      915
          Data Set                          WORK.LONG97DATA
          Model                                   Poisson
          Log Likelihood                            -1651
          Maximum Absolute Gradient             0.0002080
          Number of Iterations                         13
          Optimization Method                Quasi-Newton
          AIC                                        3314
          SBC                                        3343
```

Figure 3.3 shows the parameter estimates of the model and their standard errors. All covariates are significant predictors of the number of articles, except for the prestige of the program (phd), which has a p-value of 0.6271.

Figure 3.3 Parameter Estimates of Poisson Regression

```
                         Parameter Estimates

                                        Standard
     Effect          DF     Estimate       Error    t Value    Pr > |t|

     Intercept        1       0.3046      0.1030       2.96      0.0031
     fem              1      -0.2246     0.05461      -4.11      <.0001
     mar              1       0.1552     0.06137       2.53      0.0114
     kid5             1      -0.1849     0.04013      -4.61      <.0001
     phd              1      0.01282     0.02640       0.49      0.6271
     ment             1      0.02554    0.002006      12.73      <.0001
```

To allow for variance greater than the mean, you can fit the negative binomial model instead of the Poisson by specifying the DIST=NEGBIN option, as shown in the following statements. While the Poisson model requires that the conditional mean and conditional variance be equal, the negative binomial model allows for overdispersion, in which the conditional variance can exceed the conditional mean.

```
/*-- Negative Binomial Regression --*/
proc hpcountreg data=long97data;
   model art = fem mar kid5 phd ment / dist=negbin(p=2) method=quanew;
   performance nthreads=2 nodes=4 details;
run;
```

Figure 3.4 shows the fit summary and Figure 3.5 shows the parameter estimates.

Figure 3.4 Estimation Summary Table for a Negative Binomial Regression

```
                         The HPCOUNTREG Procedure

                            Model Fit Summary

                   Dependent Variable                        art
                   Number of Observations                    915
                   Data Set                        WORK.LONG97DATA
                   Model                                  NegBin
                   Log Likelihood                          -1561
                   Maximum Absolute Gradient           0.0000666
                   Number of Iterations                       16
                   Optimization Method               Quasi-Newton
                   AIC                                      3136
                   SBC                                      3170
```

Figure 3.5 Parameter Estimates of Negative Binomial Regression

```
                          Parameter Estimates

                                         Standard
             Effect        DF   Estimate      Error   t Value   Pr > |t|

             Intercept      1     0.2561     0.1386      1.85     0.0645
             fem            1    -0.2164    0.07267     -2.98     0.0029
             mar            1     0.1505    0.08211      1.83     0.0668
             kid5           1    -0.1764    0.05306     -3.32     0.0009
             phd            1    0.01527    0.03604      0.42     0.6718
             ment           1    0.02908   0.003470      8.38     <.0001
             _Alpha         1     0.4416    0.05297      8.34     <.0001
```

The parameter estimate for _Alpha of 0.4416 is an estimate of the dispersion parameter in the negative binomial distribution. A t test for the hypothesis $H_0 : \alpha = 0$ is provided. It is highly significant, indicating overdispersion ($p < 0.0001$).

The null hypothesis $H_0 : \alpha = 0$ can be also tested against the alternative $\alpha > 0$ by using the likelihood ratio test, as described by Cameron and Trivedi (1998, pp. 45, 77–78). The likelihood ratio test statistic is equal to $-2(\mathcal{L}_P - \mathcal{L}_{NB}) = -2(-1651 + 1561) = 180$, which is highly significant, providing strong evidence of overdispersion.

Syntax: HPCOUNTREG Procedure

The following statements are available in the HPCOUNTREG procedure. Items within angle brackets (< >) or square brackets ([ ]) are optional.

PROC HPCOUNTREG *<options>* **;**
  BOUNDS *bound1* [*, bound2 . . .*] **;**
  FREQ *freq-variable* **;**
  INIT *initialization1 < , initialization2 . . . >* **;**
  MODEL *dependent-variable* **=** *regressors </ options>* **;**
  OUTPUT *<output-options>* **;**
  PERFORMANCE *performance-options* **;**
  RESTRICT *restriction1* [*, restriction2 . . .*] **;**
  WEIGHT *variable </ option>* **;**
  ZEROMODEL *dependent-variable* ∼ *zero-inflated-regressors </ options>* **;**

There can be only one MODEL statement. The ZEROMODEL statement, if used, must appear after the MODEL statement. If a FREQ or WEIGHT statement is specified more than once, the variable specified in the first instance is used.

Functional Summary

Table 3.1 summarizes the statements and options used with the HPCOUNTREG procedure.

Table 3.1 HPCOUNTREG Functional Summary

Description	Statement	Option
Data Set Options		
Specifies the input data set	HPCOUNTREG	DATA=
Writes parameter estimates to an output data set	HPCOUNTREG	OUTEST=
Writes estimates of $\mathbf{x}_i'\boldsymbol{\beta}$ and $\mathbf{z}_i'\boldsymbol{\gamma}$ to an output data set	OUTPUT	OUT=
Specifies an optional frequency variable	FREQ	
Specifies an optional weight variable	WEIGHT	
Printing Control Options		
Prints the correlation matrix of the estimates	MODEL	CORRB
Prints the covariance matrix of the estimates	MODEL	COVB
Prints a summary iteration listing	MODEL	ITPRINT
Suppresses the normal printed output	HPCOUNTREG	NOPRINT
Requests all printing options	MODEL	PRINTALL
Options to Control the Optimization Process		
Specifies maximum number of iterations allowed	MODEL	MAXITER=
Selects the iterative minimization method to use	HPCOUNTREG	METHOD=
Specifies maximum number of iterations allowed	HPCOUNTREG	MAXITER=
Specifies maximum number of function calls	HPCOUNTREG	MAXFUNC=
Specifies the upper limit of CPU time in seconds	HPCOUNTREG	MAXTIME=
Specifies absolute function convergence criterion	HPCOUNTREG	ABSCONV=
Specifies absolute function convergence criterion	HPCOUNTREG	ABSFCONV=

Description	**Statement**	**Option**
Specifies absolute gradient convergence criterion	HPCOUNTREG	ABSGCONV=
Specifies relative function convergence criterion	HPCOUNTREG	FCONV=
Specifies relative gradient convergence criterion	HPCOUNTREG	GCONV=
Specifies absolute parameter convergence criterion	HPCOUNTREG	ABSXCONV=
Specifies matrix singularity criterion	HPCOUNTREG	SINGULAR=
Sets boundary restrictions on parameters	BOUNDS	
Sets initial values for parameters	INIT	
Sets linear restrictions on parameters	RESTRICT	
Model Estimation Options		
Specifies the type of model	MODEL	DIST=
Specifies the type of model	HPCOUNTREG	DIST=
Specifies the type of covariance matrix	MODEL	COVEST=
Suppresses the intercept parameter	MODEL	NOINT
Specifies the offset variable	MODEL	OFFSET=
Specifies the zero-inflated offset variable	ZEROMODEL	OFFSET=
Specifies the zero-inflated link function	ZEROMODEL	LINK=
Output Control Options		
Includes covariances in the OUTEST= data set	HPCOUNTREG	COVOUT
Outputs probability of the actual value	OUTPUT	PROB=
Outputs expected value of response variable	OUTPUT	PRED=
Outputs estimates of XBeta $= \mathbf{x}_i' \boldsymbol{\beta}$	OUTPUT	XBETA=
Outputs estimates of ZGamma $= \mathbf{z}_i' \boldsymbol{\gamma}$	OUTPUT	ZGAMMA=
Outputs probability of a zero value as a result of the zero-generating process	OUTPUT	PROBZERO=
Performance Options		
Requests a table that shows a timing breakdown	PERFORMANCE	DETAILS
Specifies the number of threads to use	PERFORMANCE	NTHREADS=
Specifies the number of nodes to use on the SAS appliance	PERFORMANCE	NODES=

PROC HPCOUNTREG Statement

PROC HPCOUNTREG <*options*> **;**

The following *options* can be used in the PROC HPCOUNTREG statement:

Input Data Set Option

DATA=*SAS-data-set*

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

Output Data Set Options

OUTEST=*SAS-data-set*

writes the parameter estimates to the specified output data set.

COVOUT

writes the covariance matrix for the parameter estimates to the OUTEST= data set. This option is valid only if the OUTEST= option is specified.

Printing Options

You can specify the following *options* in either the PROC HPCOUNTREG statement or in the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

ITPRINT

prints the objective function and parameter estimates at each iteration. The objective function is the negative log-likelihood function.

NOPRINT

suppresses all printed output.

PRINTALL

requests all printing options.

Estimation Control Options

COVEST=*value*

specifies the type of covariance matrix for the parameter estimates.

The default is COVEST=HESSIAN. You can specify the following *values*:

OP	specifies the covariance from the outer product matrix.
HESSIAN	specifies the covariance from the Hessian matrix.
QML	specifies the covariance from the outer product and Hessian matrices.

Optimization Control Options

PROC HPCOUNTREG uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. You can specify the following *options*:

ABSCONV=*r*

ABSTOL=*r*

specifies an absolute function value convergence criterion. For minimization, termination requires $f(\theta^{(k)}) \leq r$. The default value of *r* is the negative square root of the largest double-precision value, which serves only as a protection against overflows.

ABSFCONV=*r*

ABSFTOL=*r*

specifies an absolute function difference convergence criterion. Termination requires a small change of the function value in successive iterations:

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

The default value is $r = 0$.

ABSGCONV=*r*

ABSGTOL=*r*

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

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

The default value is *r*=1E–5.

ABSXCONV=*r*

ABSXTOL=*r*

specifies an absolute parameter convergence criterion. Termination requires a small Euclidean distance between successive parameter vectors:

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

The default is 0.

FCONV=*r*

FTOL=*r*

specifies a relative function convergence criterion. Termination requires a small relative change of the function value in successive iterations:

$$\frac{|f(\theta^{(k)}) - f(\theta^{(k-1)})|}{|f(\theta^{(k-1)})|} \leq r$$

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

GCONV=*r*

GTOL=*r*

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

$$\frac{g(\theta^{(k)})^T [H^{(k)}]^{-1} g(\theta^{(k)})}{|f(\theta^{(k)})|} \leq r$$

For the CONGRA technique (where a reliable Hessian estimate H is not available), the following criterion is used:

$$\frac{\| g(\theta^{(k)}) \|_2^2 \quad \| s(\theta^{(k)}) \|_2}{\| g(\theta^{(k)}) - g(\theta^{(k-1)}) \|_2 \, |f(\theta^{(k)})|} \leq r$$

The default value is $r = 1E-8$.

MAXFUNC=*i*
MAXFU=*i*

specifies the maximum number of function calls in the optimization process. The default is 1,000.

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that is actually performed can exceed the number that is specified by this option.

MAXITER=*i*
MAXIT=*i*

specifies the maximum number of iterations in the optimization process. The default value is 200.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation of your computer. The time specified by this option is checked only once at the end of each iteration. Therefore, the actual running time can be much longer than that specified by this option. The actual running time includes the remaining time needed to finish the iteration and the time needed to generate the output of the results.

METHOD=*value*

specifies the iterative minimization method to use. The default is METHOD=NEWRAP. You can specify the following *values*:

CONGRA	specifies the conjugate-gradient method.
DBLDOG	specifies the double dogleg method.
NONE	specifies that no optimization be performed beyond using the ordinary least squares method to compute the parameter estimates.
NEWRAP	specifies the Newton-Raphson method (the default).
NRRIDG	specifies the Newton-Raphson Ridge method.
QUANEW	specifies the quasi-Newton method.
TRUREG	specifies the trust region method.

BOUNDS Statement

BOUNDS *bound1* [, *bound2* . . .] **;**

The BOUNDS statement imposes simple boundary constraints on the parameter estimates. You can specify any number of BOUNDS statements.

Each *bound* is composed of parameter names, constants, and inequality operators as follows:

item operator item [*operator item* [*operator item . . .*]]

Each *item* is a constant, a parameter name, or a list of parameter names. Each *operator* is <, >, <=, or >=. Parameter names are as shown in the Effect column of the "Parameter Estimates" table.

You can use both the BOUNDS statement and the RESTRICT statement to impose boundary constraints. However, the BOUNDS statement provides a simpler syntax for specifying these kinds of constraints. See also the section "RESTRICT Statement" on page 66.

The following BOUNDS statement illustrates the use of parameter lists to specify boundary constraints. It constrains the estimates of the parameter for z to be negative, the parameters for x1 through x10 to be between 0 and 1, and the parameter for x1 in the zero-inflation model to be less than 1.

```
bounds z < 0,
       0 < x1-x10 < 1,
       Inf_x1 < 1;
```

FREQ Statement

FREQ *freq-variable* **;**

The FREQ statement identifies a variable (*freq-variable*) that contains the frequency of occurrence of each observation. PROC HPCOUNTREG treats each observation as if it appears n times, where n is the value of *freq-variable* for the observation. If the value for the observation is not an integer, it is truncated to an integer. If the value is less than 1 or missing, the observation is not used in the model fitting. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

INIT Statement

INIT *initialization1* < , *initialization2 . . .* > **;**

The INIT statement sets initial values for parameters in the optimization.

Each *initialization* is written as a parameter or parameter list, followed by an optional equal sign (=), followed by a number:

parameter <=> *number*

Parameter names are as shown in the Effect column of the "Parameter Estimates" table.

MODEL Statement

MODEL *dependent-variable* **=** *regressors* </ *options*> **;**

The MODEL statement specifies the dependent variable and independent regressor variables for the regression model. The dependent count variable should take on only nonnegative integer values in the input data set. PROC HPCOUNTREG rounds any positive noninteger count value to the nearest integer. PROC HPCOUNTREG discards any observation that has a negative count.

Only one MODEL statement can be specified. The following *options* can be used in the MODEL statement after a slash (/).

DIST=*value*

specifies a type of model to be analyzed. You can specify the following *values*:

POISSON | P	specifies the Poisson regression model.
NEGBIN(P=1)	specifies the negative binomial regression model with a linear variance function.
NEGBIN(P=2) | NEGBIN	specifies the negative binomial regression model with a quadratic variance function.
ZIPOISSON | ZIP	specifies zero-inflated Poisson regression.
ZINEGBIN | ZINB	specifies zero-inflated negative binomial regression.

NOINT

suppresses the intercept parameter.

OFFSET=*offset-variable*

specifies a variable in the input data set to be used as an offset variable. The offset variable is used to allow the observational units to vary across observations. For example, when the number of shipping accidents could be measured across different time periods or the number of students who participate in an activity could be reported across different class sizes, the observational units need to be adjusted to a common denominator by using the offset variable. The offset variable appears as a covariate in the model with its parameter restricted to 1. The offset variable cannot be the response variable, the zero-inflation offset variable (if any), or any of the explanatory variables. The "Model Fit Summary" table gives the name of the data set variable used as the offset variable; it is labeled as "Offset."

Printing Options

You can specify the following *options* in either the PROC HPCOUNTREG statement or in the MODEL statement:

CORRB

prints the correlation matrix of the parameter estimates.

COVB

prints the covariance matrix of the parameter estimates.

ITPRINT

prints the objective function and parameter estimates at each iteration. The objective function is the negative log-likelihood function.

NOPRINT

suppresses all printed output.

PRINTALL
requests all printing options.

OUTPUT Statement

OUTPUT < *output-options* > **;**

The OUTPUT statement creates a new SAS data set that includes variables created by the *output-options*. These variables include the estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, the expected value of the response variable, and the probability of the response variable taking on the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data set contain the estimates of $\mathbf{z}_i'\boldsymbol{\gamma}$ and the probability that the response is zero as a result of the zero-generating process. These statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations with missing response values to the input data set, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit.

You can specify only one OUTPUT statement. You can specify the following *output-options*:

OUT=*SAS-data-set*
names the output data set.

XBETA=*name*
names the variable to contain estimates of $\mathbf{x}_i'\boldsymbol{\beta}$.

PRED=*name*
names the variable to contain the predicted value of the response variable.

PROB=*name*
names the variable to contain the probability of the response variable taking the actual value, $\Pr(Y = y_i)$.

PROBCOUNT(*value1* **<** *value2* **. . . >)**
outputs the probability of the response variable taking particular values. Each value should be a nonnegative integer. Nonintegers are rounded to the nearest integer. For *value*, you can also specify a list of the form X TO Y BY Z. For example, PROBCOUNT(0 1 2 TO 10 BY 2 15) requests predicted probabilities for counts 0, 1, 2, 4, 5, 6, 8, 10, and 15. This option is not available for the fixed- and random-effects panel models.

ZGAMMA=*name*
names the variable to contain estimates of $\mathbf{z}_i'\boldsymbol{\gamma}$.

PROBZERO=*name*
names the variable to contain the value of φ_i, which is the probability of the response variable taking on the value of 0 as a result of the zero-generating process. This variable is written to the output file only if the model is zero-inflated.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement specifies options to control the multithreaded and distributed computing environment and requests detailed results about the performance characteristics of the HPCOUNTREG procedure. With the PERFORMANCE statement you can also control whether the HPCOUNTREG procedure executes in SMP or MPP mode. The most commonly used *performance-options* in the PERFORMANCE statement include the following:

DETAILS
requests a table that shows a timing breakdown of the procedure steps.

NODES=*n*
specifies the number of nodes in the distributed computing environment, provided that the data are not processed alongside the database.

NTHREADS=*n*
specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, PROC HPCOUNTREG creates one thread per CPU for the analytic computations.

The PERFORMANCE statement for High-Performance Analytics procedures is documented in more detail in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

RESTRICT Statement

RESTRICT *restriction1* [, *restriction2* . . .] **;**

The RESTRICT statement imposes linear restrictions on the parameter estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written as an expression, followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression, as follows:

expression operator expression

The *operator* can be =, <, >, <=, or >=.

Restriction expressions can be composed of parameter names, constants, and the following operators: times ($*$), plus ($+$), and minus ($-$). Parameter names are as shown in the Effect column of the "Parameter Estimates" table. The restriction expressions must be a linear function of the variables.

Lagrange multipliers are reported in the "Parameter Estimates" table for all the active linear constraints. They are identified with the names Restrict1, Restrict2, and so on. The probabilities of these Lagrange multipliers are computed using a beta distribution (LaMotte 1994). Nonactive (nonbinding) restrictions have no effect on the estimation results and are not noted in the output.

The following RESTRICT statement constrains the negative binomial dispersion parameter α to 1, which restricts the conditional variance to be $\mu + \mu^2$:

```
restrict _Alpha = 1;
```

WEIGHT Statement

WEIGHT *variable* < */ option* > **;**

The WEIGHT statement specifies a variable to supply weighting values to use for each observation in estimating parameters. The log likelihood for each observation is multiplied by the corresponding weight variable value.

If the weight of an observation is nonpositive, that observation is not used in the estimation.

The following *option* can be added to the WEIGHT statement after a slash (/).

NONORMALIZE

does not normalize the weights. (By default, the weights are normalized so that they add up to the actual sample size. Weights w_i are normalized by multiplying them by $\frac{n}{\sum_{i=1}^{n} w_i}$, where n is the sample size.) If the weights are required to be used as they are, then specify the NONORMALIZE option.

ZEROMODEL Statement

ZEROMODEL *dependent-variable* ~ *zero-inflated-regressors* < */ options* > **;**

The ZEROMODEL statement is required if either ZIP or ZINB is specified in the DIST= option in the MODEL statement. If ZIP or ZINB is specified, then the ZEROMODEL statement must follow the MODEL statement. The dependent variable in the ZEROMODEL statement must be the same as the dependent variable in the MODEL statement.

The zero-inflated (ZI) regressors appear in the equation that determines the probability (φ_i) of a zero count. Each of these q variables has a parameter to be estimated in the regression. For example, let $\mathbf{z}_i'$ be the ith observation's $1 \times (q + 1)$ vector of values of the q ZI explanatory variables (w_0 is set to 1 for the intercept term). Then φ_i is a function of $\mathbf{z}_i'\boldsymbol{\gamma}$, where $\boldsymbol{\gamma}$ is the $(q + 1) \times 1$ vector of parameters to be estimated. (The zero-inflated intercept is γ_0; the coefficients for the q zero-inflated covariates are $\gamma_1, \ldots, \gamma_q$.) If q is equal to 0 (no ZI explanatory variables are provided), then only the intercept term γ_0 is estimated. The "Parameter Estimates" table in the displayed output shows the estimates for the ZI intercept and ZI explanatory variables; they are labeled with the prefix "Inf_". For example, the ZI intercept is labeled "Inf_intercept". If you specify Age (a variable in your data set) as a ZI explanatory variable, then the "Parameter Estimates" table labels the corresponding parameter estimate "Inf_Age".

You can specify the following *options* in the ZEROMODEL statement following a slash (/):

LINK=*value*

specifies the distribution function used to compute probability of zeros. The supported distribution functions are as follows:

LOGISTIC	specifies logistic distribution.
NORMAL	specifies standard normal distribution.

If this option is omitted, then the default ZI link function is logistic.

OFFSET=*zero-inflated-offset-variable*

specifies a variable in the input data set to be used as a zero-inflated (ZI) offset variable. The ZI offset variable *zero-inflated-offset-variable* is included as a term, with coefficient restricted to 1, in the equation that determines the probability (φ_i) of a zero count and represents an adjustment to a common observational unit. The ZI offset variable cannot be the response variable, the offset variable (if any), or any of the explanatory variables. The name of the data set variable used as the ZI offset variable is displayed in the "Model Fit Summary" table, where it is labeled as "Inf_offset".

Details: HPCOUNTREG Procedure

Missing Values

Any observation in the input data set with a missing value for one or more of the regressors is ignored by PROC HPCOUNTREG and not used in the model fit. PROC HPCOUNTREG rounds any positive noninteger count values to the nearest integer. PROC HPCOUNTREG ignores any observations with a negative count.

If the input data set contains any observations that have missing response values but nonmissing regressors, PROC HPCOUNTREG can compute several statistics and store them in an output data set by using the OUTPUT statement. For example, you can request that the output data set contain the estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, the expected value of the response variable, and the probability of the response variable taking on the current value. Furthermore, if a zero-inflated model was fit, you can request that the output data set contain the estimates of $\mathbf{z}_i'\boldsymbol{\gamma}$, and the probability that the response is 0 as a result of the zero-generating process. Note that the presence of such observations (with missing response values) does not affect the model fit.

Poisson Regression

The most widely used model for count data analysis is Poisson regression. Poisson regression assumes that y_i, given the vector of covariates $\mathbf{x}_i$, is independently Poisson distributed with

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

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

$$\mu_i = \exp(\mathbf{x}_i'\boldsymbol{\beta})$$

where $\boldsymbol{\beta}$ is a $(k+1) \times 1$ parameter vector. (The intercept is β_0; the coefficients for the k regressors are $\beta_1, \dots, \beta_k$.) Taking the exponential of $\mathbf{x}_i'\boldsymbol{\beta}$ ensures that the mean parameter μ_i is nonnegative. It can be shown that the conditional mean is given by

$$E(y_i|\mathbf{x}_i) = \mu_i = \exp(\mathbf{x}_i'\boldsymbol{\beta})$$

The name *log-linear model* is also used for the Poisson regression model since the logarithm of the conditional mean is linear in the parameters:

$$\ln[E(y_i|\mathbf{x}_i)] = \ln(\mu_i) = \mathbf{x}_i'\boldsymbol{\beta}$$

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

$$V(y_i|\mathbf{x}_i) = E(y_i|\mathbf{x}_i) = \mu_i$$

The equality of the conditional mean and variance of y_i is known as *equidispersion*.

The marginal effect of a regressor is given by

$$\frac{\partial E(y_i|\mathbf{x}_i)}{\partial x_{ji}} = \exp(\mathbf{x}_i'\boldsymbol{\beta})\boldsymbol{\beta}_j = E(y_i|\mathbf{x}_i)\beta_j$$

Thus, a one-unit change in the jth regressor leads to a proportional change in the conditional mean $E(y_i|\mathbf{x}_i)$ of β_j.

The standard estimator for the Poisson model is the maximum likelihood estimator (MLE). Since the observations are independent, the log-likelihood function is written as

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

The gradient and the Hessian are as follows, respectively:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{N}(y_i - \mu_i)\mathbf{x}_i = \sum_{i=1}^{N}(y_i - e^{\mathbf{x}_i'\boldsymbol{\beta}})\mathbf{x}_i$$

$$\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}'} = -\sum_{i=1}^{N}\mu_i\mathbf{x}_i\mathbf{x}_i{}' = -\sum_{i=1}^{N}e^{\mathbf{x}_i'\boldsymbol{\beta}}\mathbf{x}_i\mathbf{x}_i'$$

The Poisson model has been criticized for its restrictive property that the conditional variance equals the conditional mean. Real-life data are often characterized by *overdispersion*—that is, the variance exceeds the mean. Allowing for overdispersion can improve model predictions since the Poisson restriction of equal mean and variance results in the underprediction of zeros when overdispersion exists. The most commonly used model that accounts for overdispersion is the negative binomial model.

Negative Binomial Regression

The Poisson regression model can be generalized by introducing an unobserved heterogeneity term for observation i. Thus, the individuals are assumed to differ randomly in a manner that is not fully accounted for by the observed covariates. This is formulated as

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

where the unobserved heterogeneity term $\tau_i = e^{\epsilon_i}$ is independent of the vector of regressors $\mathbf{x}_i$. Then the distribution of y_i conditional on $\mathbf{x}_i$ and τ_i is Poisson with conditional mean and conditional variance $\mu_i \tau_i$:

$$f(y_i|\mathbf{x}_i, \tau_i) = \frac{\exp(-\mu_i \tau_i)(\mu_i \tau_i)^{y_i}}{y_i!}$$

Let $g(\tau_i)$ be the probability density function of τ_i. Then, the distribution $f(y_i|\mathbf{x}_i)$ (no longer conditional on τ_i) is obtained by integrating $f(y_i|\mathbf{x}_i, \tau_i)$ with respect to τ_i:

$$f(y_i|\mathbf{x}_i) = \int_0^\infty f(y_i|\mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i$$

An analytical solution to this integral exists when τ_i is assumed to follow a gamma distribution. This solution is the negative binomial distribution. When the model contains a constant term, it is necessary to assume that $E(e^{\epsilon_i}) = E(\tau_i) = 1$, in order to identify the mean of the distribution. Thus, it is assumed that τ_i follows a gamma(θ, θ) distribution with $E(\tau_i) = 1$ and $V(\tau_i) = 1/\theta$,

$$g(\tau_i) = \frac{\theta^\theta}{\Gamma(\theta)} \tau_i^{\theta-1} \exp(-\theta \tau_i)$$

where $\Gamma(x) = \int_0^\infty z^{x-1} \exp(-z) dz$ is the gamma function and θ is a positive parameter. Then, the density of y_i given $\mathbf{x}_i$ is derived as

$$\begin{aligned}
f(y_i|\mathbf{x}_i) &= \int_0^\infty f(y_i|\mathbf{x}_i, \tau_i) g(\tau_i) d\tau_i \\
&= \frac{\theta^\theta \mu_i^{y_i}}{y_i! \Gamma(\theta)} \int_0^\infty e^{-(\mu_i + \theta)\tau_i} \tau_i^{\theta + y_i - 1} d\tau_i \\
&= \frac{\theta^\theta \mu_i^{y_i} \Gamma(y_i + \theta)}{y_i! \Gamma(\theta)(\theta + \mu_i)^{\theta + y_i}} \\
&= \frac{\Gamma(y_i + \theta)}{y_i! \Gamma(\theta)} \left(\frac{\theta}{\theta + \mu_i} \right)^\theta \left(\frac{\mu_i}{\theta + \mu_i} \right)^{y_i}
\end{aligned}$$

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

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

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

$$E(y_i|\mathbf{x}_i) = \mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}}$$

and conditional variance

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

The conditional variance of the negative binomial distribution exceeds the conditional mean. Overdispersion results from neglected unobserved heterogeneity. The negative binomial model with variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha\mu_i^2$, which is quadratic in the mean, is referred to as the NEGBIN2 model (Cameron and Trivedi 1986). To estimate this model, specify DIST=NEGBIN(P=2) in the MODEL statement. The Poisson distribution is a special case of the negative binomial distribution where $\alpha = 0$. A test of the Poisson distribution can be carried out by testing the hypothesis that $\alpha = \frac{1}{\theta_i} = 0$. A Wald test of this hypothesis is provided (it is the reported t statistic for the estimated α in the negative binomial model).

The log-likelihood function of the negative binomial regression model (NEGBIN2) is given by

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^{N}\left\{\sum_{j=0}^{y_i-1}\ln(j+\alpha^{-1}) - \ln(y_i!)\right. \\ &\qquad \left. -(y_i+\alpha^{-1})\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) + y_i\ln(\alpha) + y_i\mathbf{x}_i'\boldsymbol{\beta}\right\}\end{aligned}$$

where use of the following fact is made if y is an integer:

$$\Gamma(y+a)/\Gamma(a) = \prod_{j=0}^{y-1}(j+a)$$

The gradient is

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\beta}} = \sum_{i=1}^{N}\frac{y_i-\mu_i}{1+\alpha\mu_i}\mathbf{x}_i$$

and

$$\frac{\partial\mathcal{L}}{\partial\alpha} = \sum_{i=1}^{N}\left\{-\alpha^{-2}\sum_{j=0}^{y_i-1}\frac{1}{(j+\alpha^{-1})} + \alpha^{-2}\ln(1+\alpha\mu_i) + \frac{y_i-\mu_i}{\alpha(1+\alpha\mu_i)}\right\}$$

Cameron and Trivedi (1986) consider a general class of negative binomial models with mean μ_i and variance function $\mu_i + \alpha\mu_i^p$. The NEGBIN2 model, with $p = 2$, is the standard formulation of the negative binomial model. Models with other values of p, $-\infty < p < \infty$, have the same density $f(y_i|\mathbf{x}_i)$ except that α^{-1} is replaced everywhere by $\alpha^{-1}\mu^{2-p}$. The negative binomial model NEGBIN1, which sets $p = 1$, has variance function $V(y_i|\mathbf{x}_i) = \mu_i + \alpha\mu_i$, which is linear in the mean. To estimate this model, specify DIST=NEGBIN(P=1) in the MODEL statement.

The log-likelihood function of the NEGBIN1 regression model is given by

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^{N}\left\{\sum_{j=0}^{y_i-1}\ln\left(j+\alpha^{-1}\exp(\mathbf{x}_i'\boldsymbol{\beta})\right)\right. \\ &\qquad \left. -\ln(y_i!) - \left(y_i+\alpha^{-1}\exp(\mathbf{x}_i'\boldsymbol{\beta})\right)\ln(1+\alpha) + y_i\ln(\alpha)\right\}\end{aligned}$$

The gradient is

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{N} \left\{ \left(\sum_{j=0}^{y_i-1} \frac{\mu_i}{(j\alpha + \mu_i)} \right) \mathbf{x}_i - \alpha^{-1} \ln(1+\alpha)\mu_i \mathbf{x}_i \right\}$$

and

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{i=1}^{N} \left\{ - \left(\sum_{j=0}^{y_i-1} \frac{\alpha^{-1}\mu_i}{(j\alpha + \mu_i)} \right) - \alpha^{-2}\mu_i \ln(1+\alpha) - \frac{(y_i + \alpha^{-1}\mu_i)}{1+\alpha} + \frac{y_i}{\alpha} \right\}$$

Zero-Inflated Count Regression Overview

The main motivation for zero-inflated count models is that real-life data frequently display overdispersion and excess zeros. Zero-inflated count models provide a way of both modeling the excess zeros and allowing for overdispersion. In particular, there are two possible data generation processes for each observation. The result of a Bernoulli trial is used to determine which of the two processes is used. For observation i, Process 1 is chosen with probability φ_i and Process 2 with probability $1 - \varphi_i$. Process 1 generates only zero counts. Process 2 generates counts from either a Poisson or a negative binomial model. In general,

$$y_i \sim \begin{cases} 0 & \text{with probability} \quad \varphi_i \\ g(y_i) & \text{with probability} \quad 1-\varphi_i \end{cases}$$

Therefore, the probability of $\{Y_i = y_i\}$ can be described as

$$\begin{aligned} P(y_i = 0|\mathbf{x}_i) &= \varphi_i + (1-\varphi_i)g(0) \\ P(y_i|\mathbf{x}_i) &= (1-\varphi_i)g(y_i), \quad y_i > 0 \end{aligned}$$

where $g(y_i)$ follows either the Poisson or the negative binomial distribution.

When the probability φ_i depends on the characteristics of observation i, φ_i is written as a function of $\mathbf{z}_i'\boldsymbol{\gamma}$, where $\mathbf{z}_i'$ is the $1 \times (q+1)$ vector of zero-inflated covariates and $\boldsymbol{\gamma}$ is the $(q+1) \times 1$ vector of zero-inflated coefficients to be estimated. (The zero-inflated intercept is γ_0; the coefficients for the q zero-inflated covariates are $\gamma_1, \ldots, \gamma_q$.) The function F that relates the product $\mathbf{z}_i'\boldsymbol{\gamma}$ (which is a scalar) to the probability φ_i is called the zero-inflated link function,

$$\varphi_i = F_i = F(\mathbf{z}_i'\boldsymbol{\gamma})$$

In the HPCOUNTREG procedure, the zero-inflated covariates are indicated in the ZEROMODEL statement. Furthermore, the zero-inflated link function F can be specified as either the logistic function,

$$F(\mathbf{z}_i'\boldsymbol{\gamma}) = \Lambda(\mathbf{z}_i'\boldsymbol{\gamma}) = \frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{1+\exp(\mathbf{z}_i'\boldsymbol{\gamma})}$$

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

$$F(\mathbf{z}_i'\boldsymbol{\gamma}) = \Phi(\mathbf{z}_i'\boldsymbol{\gamma}) = \int_0^{\mathbf{z}_i'\boldsymbol{\gamma}} \frac{1}{\sqrt{2\pi}} \exp(-u^2/2)du$$

The zero-inflated link function is indicated by using the LINK= option in the ZEROMODEL statement. The default ZI link function is the logistic function.

Zero-Inflated Poisson Regression

In the zero-inflated Poisson (ZIP) regression model, the data generation process referred to earlier as Process 2 is

$$g(y_i) = \frac{\exp(-\mu_i)\mu_i^{y_i}}{y_i!}$$

where $\mu_i = e^{\mathbf{x}_i' \boldsymbol{\beta}}$. Thus the ZIP model is defined as

$$\begin{aligned} P(y_i = 0|\mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i)\exp(-\mu_i) \\ P(y_i|\mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i)\frac{\exp(-\mu_i)\mu_i^{y_i}}{y_i!}, \quad y_i > 0 \end{aligned}$$

The conditional expectation and conditional variance of y_i are given by

$$E(y_i|\mathbf{x}_i, \mathbf{z}_i) = \mu_i(1 - F_i)$$

$$V(y_i|\mathbf{x}_i, \mathbf{z}_i) = E(y_i|\mathbf{x}_i, \mathbf{z}_i)(1 + \mu_i F_i)$$

Note that the ZIP model (in addition to the ZINB model) exhibits overdispersion since $V(y_i|\mathbf{x}_i, \mathbf{z}_i) > E(y_i|\mathbf{x}_i, \mathbf{z}_i)$.

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

$$\mathcal{L} = \sum_{i=1}^{N} \ln\left[P(y_i|\mathbf{x}_i, \mathbf{z}_i)\right]$$

Once a specific link function (either logistic or standard normal) for the probability φ_i is chosen, it is possible to write the exact expressions for the log-likelihood function and the gradient.

ZIP Model with Logistic Link Function

First, consider the ZIP model in which the probability φ_i is expressed with a logistic link function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}_i' \boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned} \mathcal{L} = & \sum_{\{i: y_i = 0\}} \ln\left[\exp(\mathbf{z}_i' \boldsymbol{\gamma}) + \exp(-\exp(\mathbf{x}_i' \boldsymbol{\beta}))\right] \\ & + \sum_{\{i: y_i > 0\}} \left[y_i \mathbf{x}_i' \boldsymbol{\beta} - \exp(\mathbf{x}_i' \boldsymbol{\beta}) - \sum_{k=2}^{y_i} \ln(k) \right] \\ & - \sum_{i=1}^{N} \ln\left[1 + \exp(\mathbf{z}_i' \boldsymbol{\gamma})\right] \end{aligned}$$

The gradient for this model is given by

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} = \sum_{\{i:y_i=0\}} \left[\frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{\exp(\mathbf{z}_i'\boldsymbol{\gamma}) + \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \right] \mathbf{z}_i - \sum_{i=1}^{N} \left[\frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma})} \right] \mathbf{z}_i$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{\{i:y_i=0\}} \left[\frac{-\exp(\mathbf{x}_i'\boldsymbol{\beta})\exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))}{\exp(\mathbf{z}_i'\boldsymbol{\gamma}) + \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \right] \mathbf{x}_i + \sum_{\{i:y_i>0\}} \left[y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta}) \right] \mathbf{x}_i$$

ZIP Model with Standard Normal Link Function

Next, consider the ZIP model in which the probability φ_i is expressed with a standard normal link function: $\varphi_i = \Phi(\mathbf{z}_i'\boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned} \mathcal{L} &= \sum_{\{i:y_i=0\}} \ln\left\{ \Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta})) \right\} \\ &+ \sum_{\{i:y_i>0\}} \left\{ \ln\left[\left(1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right)\right] - \exp(\mathbf{x}_i'\boldsymbol{\beta}) + y_i \mathbf{x}_i'\boldsymbol{\beta} - \sum_{k=2}^{y_i} \ln(k) \right\} \end{aligned}$$

The gradient for this model is given by

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} &= \sum_{\{i:y_i=0\}} \frac{\varphi(\mathbf{z}_i'\boldsymbol{\gamma})\left[1 - \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))\right]}{\Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \mathbf{z}_i \\ &- \sum_{\{i:y_i>0\}} \frac{\varphi(\mathbf{z}_i'\boldsymbol{\gamma})}{\left[1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right]} \mathbf{z}_i \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \sum_{\{i:y_i=0\}} \frac{-\left[1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \exp(\mathbf{x}_i'\boldsymbol{\beta}) \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))}{\Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1 - \Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \exp(-\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \mathbf{x}_i \\ &+ \sum_{\{i:y_i>0\}} \left[y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta}) \right] \mathbf{x}_i \end{aligned}$$

Zero-Inflated Negative Binomial Regression

The zero-inflated negative binomial (ZINB) model in PROC HPCOUNTREG is based on the negative binomial model with quadratic variance function (when DIST=NEGBIN in the MODEL or PROC HPCOUNTREG statement). The ZINB model is obtained by specifying a negative binomial distribution for the data generation process referred to earlier as Process 2:

$$g(y_i) = \frac{\Gamma(y_i + \alpha^{-1})}{y_i!\Gamma(\alpha^{-1})} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i} \right)^{\alpha^{-1}} \left(\frac{\mu_i}{\alpha^{-1} + \mu_i} \right)^{y_i}$$

Thus the ZINB model is defined to be

$$\begin{aligned}
P(y_i = 0|\mathbf{x}_i, \mathbf{z}_i) &= F_i + (1 - F_i)\,(1 + \alpha\mu_i)^{-\alpha^{-1}} \\
P(y_i|\mathbf{x}_i, \mathbf{z}_i) &= (1 - F_i)\,\frac{\Gamma(y_i + \alpha^{-1})}{y_i!\Gamma(\alpha^{-1})}\left(\frac{\alpha^{-1}}{\alpha^{-1} + \mu_i}\right)^{\alpha^{-1}} \\
&\times \left(\frac{\mu_i}{\alpha^{-1} + \mu_i}\right)^{y_i}, \quad y_i > 0
\end{aligned}$$

In this case, the conditional expectation and conditional variance of y_i are

$$E(y_i|\mathbf{x}_i, \mathbf{z}_i) = \mu_i(1 - F_i)$$

$$V(y_i|\mathbf{x}_i, \mathbf{z}_i) = E(y_i|\mathbf{x}_i, \mathbf{z}_i)\,[1 + \mu_i(F_i + \alpha)]$$

As with the ZIP model, the ZINB model exhibits overdispersion because the conditional variance exceeds the conditional mean.

ZINB Model with Logistic Link Function

In this model, the probability φ_i is given by the logistic function, namely

$$\varphi_i = \frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma})}$$

The log-likelihood function is

$$\begin{aligned}
\mathcal{L} &= \sum_{\{i:y_i=0\}} \ln\left[\exp(\mathbf{z}_i'\boldsymbol{\gamma}) + (1 + \alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}\right] \\
&+ \sum_{\{i:y_i>0\}} \sum_{j=0}^{y_i-1} \ln(j + \alpha^{-1}) \\
&+ \sum_{\{i:y_i>0\}} \left\{-\ln(y_i!) - (y_i + \alpha^{-1})\ln(1 + \alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) + y_i\ln(\alpha) + y_i\mathbf{x}_i'\boldsymbol{\beta}\right\} \\
&- \sum_{i=1}^{N} \ln\left[1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma})\right]
\end{aligned}$$

The gradient for this model is given by

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\boldsymbol{\gamma}} &= \sum_{\{i:y_i=0\}} \left[\frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{\exp(\mathbf{z}_i'\boldsymbol{\gamma}) + (1 + \alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}}\right]\mathbf{z}_i \\
&- \sum_{i=1}^{N}\left[\frac{\exp(\mathbf{z}_i'\boldsymbol{\gamma})}{1 + \exp(\mathbf{z}_i'\boldsymbol{\gamma})}\right]\mathbf{z}_i
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} &= \sum_{\{i:y_i=0\}} \left[\frac{-\exp(\mathbf{x}_i'\boldsymbol{\beta})(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}-1}}{\exp(\mathbf{z}_i'\boldsymbol{\gamma}) + (1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}} \right] \mathbf{x}_i \\
&+ \sum_{\{i:y_i>0\}} \left[\frac{y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta})}{1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})} \right] \mathbf{x}_i
\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{\{i:y_i=0\}} \frac{\alpha^{-2}\left[(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) - \alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})\right]}{\exp(\mathbf{z}_i'\boldsymbol{\gamma})(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{(1+\alpha)/\alpha} + (1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))}$$

$$+ \sum_{\{i:y_i>0\}} \left\{ -\alpha^{-2} \sum_{j=0}^{y_i-1} \frac{1}{(j+\alpha^{-1})} + \alpha^{-2}\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) + \frac{y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta})}{\alpha(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \right\}$$

ZINB Model with Standard Normal Link Function

For this model, the probability φ_i is specified with the standard normal distribution function (probit function): $\varphi_i = \Phi(\mathbf{z}_i'\boldsymbol{\gamma})$. The log-likelihood function is

$$\begin{aligned}
\mathcal{L} &= \sum_{\{i:y_i=0\}} \ln\left\{ \Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right](1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}} \right\} \\
&+ \sum_{\{i:y_i>0\}} \ln\left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \\
&+ \sum_{\{i:y_i>0\}} \sum_{j=0}^{y_i-1} \{\ln(j+\alpha^{-1})\} \\
&- \sum_{\{i:y_i>0\}} \ln(y_i!) \\
&- \sum_{\{i:y_i>0\}} (y_i+\alpha^{-1})\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) \\
&+ \sum_{\{i:y_i>0\}} y_i\ln(\alpha) \\
&+ \sum_{\{i:y_i>0\}} y_i\mathbf{x}_i'\boldsymbol{\beta}
\end{aligned}$$

The gradient for this model is given by

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\gamma}} = \sum_{\{i:y_i=0\}} \left[\frac{\varphi(\mathbf{z}_i'\boldsymbol{\gamma})\left[1-(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}\right]}{\Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right](1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}} \right] \mathbf{z}_i$$

$$- \sum_{\{i:y_i>0\}} \left[\frac{\varphi(\mathbf{z}_i'\boldsymbol{\gamma})}{1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})} \right] \mathbf{z}_i$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{\{i:y_i=0\}} \frac{-\left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right] \exp(\mathbf{x}_i'\boldsymbol{\beta})(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-(1+\alpha)/\alpha}}{\Phi(\mathbf{z}_i'\boldsymbol{\gamma}) + \left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right](1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{-\alpha^{-1}}} \mathbf{x}_i$$

$$+ \sum_{\{i:y_i>0\}} \left[\frac{y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta})}{1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})} \right] \mathbf{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \sum_{\{i:y_i=0\}} \frac{\left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right]\alpha^{-2}\left[(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) - \alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})\right]}{\Phi(\mathbf{z}_i'\boldsymbol{\gamma})(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))^{(1+\alpha)/\alpha} + \left[1-\Phi(\mathbf{z}_i'\boldsymbol{\gamma})\right](1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))}$$

$$+ \sum_{\{i:y_i>0\}} \left\{ -\alpha^{-2} \sum_{j=0}^{y_i-1} \frac{1}{(j+\alpha^{-1})} + \alpha^{-2}\ln(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta})) + \frac{y_i - \exp(\mathbf{x}_i'\boldsymbol{\beta})}{\alpha(1+\alpha\exp(\mathbf{x}_i'\boldsymbol{\beta}))} \right\}$$

Computational Resources

The time and memory required by PROC HPCOUNTREG are proportional to the number of parameters in the model and the number of observations in the data set being analyzed. Less time and memory are required for smaller models and fewer observations. When PROC HPCOUNTREG is run in the high-performance distributed environment, the amount of time required is also affected by the number of nodes and the number of threads per node as specified in the PERFORMANCE statement.

The method chosen to calculate the variance-covariance matrix and the optimization method also affect the time and memory resources. All optimization methods available through the METHOD= option have similar memory use requirements. The processing time might differ for each method depending on the number of iterations and functional calls needed. The data set is read into memory to save processing time. If not enough memory is available to hold the data, the HPCOUNTREG procedure stores the data in a utility file on disk and rereads the data as needed from this file. When this occurs, the execution time of the procedure increases substantially. The gradient and the variance-covariance matrix must be held in memory. If the model has p parameters including the intercept, then at least $8 * (p + p * (p + 1)/2)$ bytes are needed. The processing time is also a function of the number of iterations needed to converge to a solution for the model parameters. The number of iterations needed cannot be known in advance. You can use the MAXITER= option to limit the number of iterations that PROC HPCOUNTREG executes. You can alter the convergence criteria by using nonlinear optimization options available in the PROC HPCOUNTREG statement. For a list of all the nonlinear optimization options, see “Optimization Control Options” on page 60.

Nonlinear Optimization Options

PROC HPCOUNTREG uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization tasks. In the PROC HPCOUNTREG statement, you can specify nonlinear optimization options that are then passed to the NLO subsystem. For a list of all the nonlinear optimization options, see "Optimization Control Options" on page 60.

Covariance Matrix Types

The COVEST= option in the PROC HPCOUNTREG statement enables you to specify the estimation method for the covariance matrix. COVEST=HESSIAN estimates the covariance matrix based on the inverse of the Hessian matrix; COVEST=OP uses the outer product of gradients; and COVEST=QML produces the covariance matrix based on both the Hessian and outer product matrices. Although all three methods produce asymptotically equivalent results, they differ in computational intensity and produce results that might differ in finite samples. The COVEST=OP option provides the covariance matrix that is typically the easiest to compute. In some cases, the OP approximation is considered more efficient than the Hessian or QML approximations because it contains fewer random elements. The QML approximation is computationally the most complex because both the outer product of gradients and the Hessian matrix are required. In most cases, OP or Hessian approximations are preferred to QML. The need to use QML approximation arises in some cases when the model is misspecified and the information matrix equality does not hold. The default is COVEST=HESSIAN.

Displayed Output

PROC HPCOUNTREG produces the following displayed output.

Iteration History for Parameter Estimates

If you specify the ITPRINT or PRINTALL options in the PROC HPCOUNTREG statement, PROC HPCOUNTREG displays a table that contains the following for each iteration. Some information is specific to the model fitting procedure chosen, as indicated.

- iteration number
- number of restarts since the fitting began
- number of function calls
- number of active constraints at the current solution
- value of the objective function (–1 times the log-likelihood value) at the current solution

- change in the objective function from the previous iteration
- value of the maximum absolute gradient element
- step size (for Newton-Raphson and quasi-Newton methods)
- slope of the current search direction (for Newton-Raphson and quasi-Newton methods)
- lambda (for trust region method)
- radius value at current iteration (for trust region method)

Model Fit Summary

The "Model Fit Summary" table contains the following information:

- dependent (count) variable name
- number of observations used
- number of missing values in data set, if any
- data set name
- type of model that was fit
- offset variable name, if any
- zero-inflated link function, if any
- zero-inflated offset variable name, if any
- log-likelihood value at solution
- maximum absolute gradient at solution
- number of iterations
- AIC value at solution (smaller value indicates better fit)
- SBC value at solution (smaller value indicates better fit)

A line in the "Model Fit Summary" table indicates whether the algorithm successfully converged.

Parameter Estimates

The "Parameter Estimates" table in the displayed output gives the estimates for the ZI intercept and ZI explanatory variables; they are labeled with the prefix "Inf_". For example, the ZI intercept is labeled "Inf_intercept". If you specify "Age" (a variable in your data set) as a ZI explanatory variable, then the "Parameter Estimates" table labels the corresponding parameter estimate "Inf_Age". If you do not list any ZI explanatory variables (for the ZI option VAR=), then only the intercept term is estimated.

"_Alpha" is the negative binomial dispersion parameter. The t statistic given for "_Alpha" is a test of overdispersion.

Last Evaluation of the Gradient

If you specify the ITPRINT option in the PROC HPCOUNTREG or MODEL statement, the HPCOUNTREG procedure displays the last evaluation of the gradient vector.

Covariance of Parameter Estimates

If you specify the COVB option in the PROC HPCOUNTREG or MODEL statement, the HPCOUNTREG procedure displays the estimated covariance matrix, which is defined as the inverse of the information matrix at the final iteration.

Correlation of Parameter Estimates

If you specify the CORRB option in the PROC HPCOUNTREG or MODEL statement, the HPCOUNTREG procedure displays the estimated correlation matrix, which is based on the Hessian matrix used at the final iteration.

OUTPUT OUT= Data Set

The OUTPUT statement creates a new SAS data set that contains various estimates that you specify. You can request that the output data set contain the estimates of $\mathbf{x}_i'\boldsymbol{\beta}$, the expected value of the response variable, and the probability of the response variable taking on the current value. Furthermore, if a zero-inflated model is fit, you can request that the output data set contain the estimates of $\mathbf{z}_i'\boldsymbol{\gamma}$ and the probability that the response is 0 as a result of the zero-generating process. These statistics can be computed for all observations in which the regressors are not missing, even if the response is missing. By adding observations with missing response values to the input data set, you can compute these statistics for new observations or for settings of the regressors that are not present in the data without affecting the model fit. Due to potential space limitations on the client workstation, the data set created by the OUTPUT statement does not contain the variables in the input data set.

OUTEST= Data Set

The OUTEST= data set is made up of at least two rows: the first row (with _TYPE_='PARM') contains each of the parameter estimates in the model, and the second row (with _TYPE_='STD') contains the standard errors for the parameter estimates in the model.

If you use the COVOUT option in the PROC HPCOUNTREG statement, the OUTEST= data set also contains the covariance matrix for the parameter estimates. The covariance matrix appears in the observations with _TYPE_='COV', and the _NAME_ variable labels the rows with the parameter names.

ODS Table Names

PROC HPCOUNTREG assigns a name to each table it creates. You can use these names to denote the table when you use the Output Delivery System (ODS) to select tables and create output data sets. These names are listed in Table 3.2.

Table 3.2 ODS Tables Produced in PROC HPCOUNTREG

ODS Table Name	Description	Option
ODS Tables Created by the MODEL Statement		
FitSummary	Summary of nonlinear estimation	Default
ConvergenceStatus	Convergence status	Default
ParameterEstimates	Parameter estimates	Default
CovB	Covariance of parameter estimates	COVB
CorrB	Correlation of parameter estimates	CORRB
InputOptions	Input options	ITPRINT
IterStart	Optimization start	ITPRINT
IterHist	Iteration history	ITPRINT
IterStop	Optimization results	ITPRINT
ParameterEstimatesResults	Parameter estimates	ITPRINT
ParameterEstimatesStart	Parameter estimates	ITPRINT
ProblemDescription	Problem description	ITPRINT

Examples: The HPCOUNTREG Procedure

Example 3.1: High-Performance Zero-Inflated Poisson Model

This example shows the use of the HPCOUNTREG procedure with an emphasis on large data set processing and the performance improvements that are achieved by executing in the high-performance distributed environment.

The following DATA step generates one million replicates from the zero-inflated Poisson (ZIP) model. The model contains seven variables and three variables that correspond to the zero-inflated process.

```
data simulate;
   call streaminit(12345);
   array vars x1-x7;
   array zero_vars z1-z3;
```

```
         array parms{7}  (.3 .4 .2 .4 -.3 -.5 -.3);
         array zero_parms{3} (-.6 .3 .2);

         intercept=2;
         z_intercept=-1;
         theta=0.5;

         do i=1 to 1000000;
            sum_xb=0;
            sum_gz=0;
            do j=1 to 7;
               vars[j]=rand('NORMAL',0,1);
               sum_xb=sum_xb+parms[j]*vars[j];
            end;
            mu=exp(intercept+sum_xb);
            y_p=rand('POISSON', mu);

            do j=1 to 3;
               zero_vars[j]=rand('NORMAL',0,1);
               sum_gz = sum_gz+zero_parms[j]*zero_vars[j];
            end;
            z_gamma = z_intercept+sum_gz;
            pzero = cdf('LOGISTIC',z_gamma);
            cut=rand('UNIFORM');
            if cut<pzero then y_p=0;
            output;
         end;
      keep y_p x1-x7 z1-z3;
      run;
```

The following statements estimate a zero-inflated Poisson model. The model is executed in the distributed computing environmet with two threads and only one node. These settings are used to obtain a hypothetical environment that might resemble running the HPCOUNTREG procedure on a desktop workstation with a dual core CPU. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to the macro variables in the example with the appropriate values.

```
      option set=GRIDHOST="&GRIDHOST";
      option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";

   proc hpcountreg data=simulate dist=zip;
      performance nthreads=2 nodes=1 details;
      model y_p=x1-x7;
      zeromodel y_p ~ z1-z3;
   run;
```

Output 3.1.1 shows the results for the zero-inflated Poisson model. The "Performance Information" table shows that the model was estimated on the grid defined in a macro variable named GRIDHOST in a distributed environment on only one node and that two threads were employed. The "Model Fit Summary" table shows detailed information about the model and indicates that all one million observations were used to fit the model. All parameter estimates in the "Parameter Estimates" table are highly significant and correspond to their theoretical values set during the data generating process. The optimization of the model with one million observations took 81.94 seconds.

Output 3.1.1 Zero-Inflated Poisson Model with One Node and Two Threads

```
                          The HPCOUNTREG Procedure

                          Performance Information

              Host Node                      << your grid host >>
              Execution Mode                 Distributed
              Number of Compute Nodes        1
              Number of Threads per Node     2

                             Model Fit Summary

                  Dependent Variable                     y_p
                  Number of Observations             1000000
                  Data Set                     WORK.SIMULATE
                  Model                                  ZIP
                  ZI Link Function                  Logistic
                  Log Likelihood                    -2215238
                  Maximum Absolute Gradient       2.04436E-8
                  Number of Iterations                     7
                  Optimization Method         Newton-Raphson
                  AIC                                4430500
                  SBC                                4430642

             Convergence criterion (FCONV=2.220446E-16) satisfied.

                            Parameter Estimates

                                            Standard
       Effect              DF    Estimate      Error    t Value    Pr > |t|

       Intercept            1      2.0005   0.000492    4069.80      <.0001
       x1                   1      0.2995   0.000352     850.17      <.0001
       x2                   1      0.3998   0.000353    1132.23      <.0001
       x3                   1      0.2008   0.000352     570.27      <.0001
       x4                   1      0.3994   0.000353    1132.85      <.0001
       x5                   1     -0.2995   0.000353    -848.95      <.0001
       x6                   1     -0.5000   0.000353    -1414.9      <.0001
       x7                   1     -0.3002   0.000352    -852.14      <.0001
       Inf_Intercept        1     -0.9993   0.002521    -396.45      <.0001
       Inf_z1               1     -0.6024   0.002585    -233.02      <.0001
       Inf_z2               1      0.2976   0.002454     121.25      <.0001
       Inf_z3               1      0.1974   0.002430      81.20      <.0001

                           Procedure Task Timing

                                                          Time
          Task                                           (sec.)

          Data read and variable levelization              0.67     0.81%
          Communication to client                          0.03     0.04%
          Optimization                                    81.94     98.8%
          Post-optimization                                0.30     0.37%
```

In the following statements, the PERFORMANCE statement is modified to use a grid with 10 nodes, with each node capable of spawning eight threads:

```
proc hpcountreg data=simulate dist=zip;
   performance nthreads=8 nodes=10 details;
   model y_p=x1-x7;
   zeromodel y_p ~ z1-z3;
run;
```

Since the two models being estimated are identical, it is reasonable to expect that Output 3.1.1 and Output 3.1.2 would show the same results. However, you can see a significant difference in performance. The second model which was run on a grid using 10 nodes with eight threads each, took only 3.28 seconds instead of 81.94 seconds to optimize.

In certain circumstances, you might observe slight numerical differences in the results, depending on the number of nodes and threads involved. This is a consequence of the fact that the order in which partial results are accumulated can make a difference in the final result, owing to the limits of numerical precision and the propagation of error in numerical computations.

Output 3.1.2 Zero-Inflated Poisson Model on 10 Nodes with Eight Threads Each

```
                          The HPCOUNTREG Procedure

                          Performance Information

              Host Node                      << your grid host >>
              Execution Mode                 Distributed
              Number of Compute Nodes        10
              Number of Threads per Node     8

                             Model Fit Summary

                  Dependent Variable                        y_p
                  Number of Observations                1000000
                  Data Set                        WORK.SIMULATE
                  Model                                     ZIP
                  ZI Link Function                     Logistic
                  Log Likelihood                       -2215238
                  Maximum Absolute Gradient          2.06085E-8
                  Number of Iterations                        7
                  Optimization Method            Newton-Raphson
                  AIC                                   4430500
                  SBC                                   4430642

             Convergence criterion (FCONV=2.220446E-16) satisfied.
```

Output 3.1.2 *continued*

```
                         Parameter Estimates

                                            Standard
Effect               DF     Estimate          Error    t Value    Pr > |t|

Intercept             1       2.0005       0.000492    4069.80      <.0001
x1                    1       0.2995       0.000352     850.17      <.0001
x2                    1       0.3998       0.000353    1132.23      <.0001
x3                    1       0.2008       0.000352     570.27      <.0001
x4                    1       0.3994       0.000353    1132.85      <.0001
x5                    1      -0.2995       0.000353    -848.95      <.0001
x6                    1      -0.5000       0.000353    -1414.9      <.0001
x7                    1      -0.3002       0.000352    -852.14      <.0001
Inf_Intercept         1      -0.9993       0.002521    -396.45      <.0001
Inf_z1                1      -0.6024       0.002585    -233.02      <.0001
Inf_z2                1       0.2976       0.002454     121.25      <.0001
Inf_z3                1       0.1974       0.002430      81.20      <.0001

                        Procedure Task Timing

                                                          Time
   Task                                                  (sec.)

   Data read and variable levelization                    0.05     1.46%
   Communication to client                                0.03     0.82%
   Optimization                                           3.28     88.1%
   Post-optimization                                      0.36     9.58%
```

As this example suggests, increasing the number of nodes and the number of threads per node improves performance significantly. When you use the parallelism afforded by a high-performance distributed environment, you can see an even more dramatic reduction in the time required for the optimization as the number of observations in the data set increases. When the data set is extremely large, the computations might not even be possible in some cases with the typical memory resources and computational constraints of a desktop computer. Under such circumstances the high-performance distributed environment becomes a necessity.

References

Cameron, A. C. and Trivedi, P. K. (1986), "Econometric Models Based on Count Data: Comparisons and Applications of Some Estimators and Some Tests," *Journal of Applied Econometrics*, 1, 29–53.

Cameron, A. C. and Trivedi, P. K. (1998), *Regression Analysis of Count Data*, Cambridge: Cambridge University Press.

LaMotte, L. R. (1994), "A Note on the Role of Independence in t Statistics Constructed from Linear Statistics in Regression Models," *The American Statistician*, 48, 238–240.

Long, J. S. (1997), *Regression Models for Categorical and Limited Dependent Variables*, Thousand Oaks, CA: Sage Publications.

Chapter 4
The HPDMDB Procedure

Contents

Overview: HPDMDB Procedure

The HPDMDB procedure is a high-performance version of the DMDB procedure, which creates summaries of the input data source. PROC HPDMDB creates two output data sets: the VAROUT data set, which contains a summary of the numeric variables, and the CLASSOUT data set, which contains a summary of the classification variables.

PROC HPDMDB is high-performance in that it takes advantage of distributed and multicore computing environments when the input data are stored on the SAS Appliance.

You can use PROC HPDMDB to create a data mining database (DMDB) that is compatible with the DMDB from PROC DMDB, although this feature of PROC HPDMDB might not be supported in future versions.

Getting Started: HPDMDB Procedure

The HPDMDB procedure summarizes data. The following example uses the Sampsio.Hmeq data set, which includes information about 5,960 fictitious mortgages. Each case represents an applicant for a home equity loan, and all applicants have an existing mortgage. The binary target variable BAD indicates whether an applicant eventually defaulted or was ever seriously delinquent. There are 10 numeric input variables and three classification input variables.

```
proc hpdmdb data=Sampsio.Hmeq
    classout=cout varout=vout;

    var loan derog mortdue value yoj delinq
        clage ninq clno debtinc;
    class bad(desc) reason(ascending) job;
run;

proc print data=cout;run;
proc print data=vout;run;
```

The data set cout (shown in Figure 4.1) contains the class summary table with levels sorted according to the sort option in the CLASS statement of PROC HPDMDB. You can see that the levels for BAD are in descending order and the levels for REASON are in ascending order. The levels for JOB are in the default ascending order.

Figure 4.1 Summaries of Classification Variables in Sampsio.Hmeq Data Set

```
Obs NAME   LEVEL     CODE  FREQUENCY TYPE CRAW     NRAW FREQPERCENT NMISSPERCENT

  1 BAD    1            1       1189  N               1     19.9497      19.9497
  2 BAD    0            0       4771  N               0     80.0503      80.0503
  3 REASON              0        252  C               .      4.2282        .
  4 REASON DEBTCON      1       3928  C    DebtCon    .     65.9060      68.8157
  5 REASON HOMEIMP      2       1780  C    HomeImp    .     29.8658      31.1843
  6 JOB                 0        279  C               .      4.6812        .
  7 JOB    MGR          1        767  C    Mgr        .     12.8691      13.5011
  8 JOB    OFFICE       2        948  C    Office     .     15.9060      16.6872
  9 JOB    OTHER        3       2388  C    Other      .     40.0671      42.0349
 10 JOB    PROFEXE      4       1276  C    ProfExe    .     21.4094      22.4608
 11 JOB    SALES        5        109  C    Sales      .      1.8289       1.9187
 12 JOB    SELF         6        193  C    Self       .      3.2383       3.3973
```

Numeric summaries are in the data set **vout**, shown in Figure 4.2.

Figure 4.2 Summaries of Numeric Variables in Sampsio.Hmeq Data Set

```
 Obs   NAME         NMISS          N      MIN        MAX        MEAN       STD

   1   LOAN             0       5960  1100.00   89900.00    18607.97  11207.48
   2   DEROG          708       5252     0.00      10.00        0.25      0.85
   3   MORTDUE        518       5442  2063.00  399550.00    73760.82  44457.61
   4   VALUE          112       5848  8000.00  855909.00   101776.05  57385.78
   5   YOJ            515       5445     0.00      41.00        8.92      7.57
   6   DELINQ         580       5380     0.00      15.00        0.45      1.13
   7   CLAGE          308       5652     0.00    1168.23      179.77     85.81
   8   NINQ           510       5450     0.00      17.00        1.19      1.73
   9   CLNO           222       5738     0.00      71.00       21.30     10.14
  10   DEBTINC       1267       4693     0.52     203.31       33.78      8.60

 Obs   SKEWNESS    KURTOSIS            SUM            USS                 CSS

   1    2.02378      6.9326   110903500.00   2.8121848E12     748495791434.56
   2    5.32087     36.8728        1337.00        4099.00             3758.64
   3    1.81448      6.4819   401406367.20   4.0362084E13      10754022449877
   4    3.05334     24.3628   595186333.04   7.9830628E13      19254914800672
   5    0.98846      0.3721       48581.75      745755.59           312296.19
   6    4.02315     23.5654        2418.00        7922.00             6835.25
   7    1.34341      7.5995     1016038.99   224259958.52         41610414.32
   8    2.62198      9.7865        6464.00       23950.00            16283.34
   9    0.77505      1.1577      122197.00     3192071.00           589751.93
  10    2.85235     50.5040      158529.14     5702262.28           347161.26
```

Syntax: HPDMDB Procedure

The following statements are available in the HPDMDB procedure:

> **PROC HPDMDB DATA=** < *libref.* >*SAS-data-set* < *options* > **;**
>   **CLASS** *variable* (< *order-option* >) *variable* (< *order-option* >) . . . **;**
>   **FREQ** *variable* **;**
>   **PERFORMANCE** *performance-options* **;**
>   **VAR** *variables* **;**
>   **WEIGHT** *variable* **;**

PROC HPDMDB Statement

PROC HPDMDB DATA= < *libref.* >*SAS-data-set* < *options* > **;**

The PROC HPDMDB statement invokes the procedure.

Required Arguments

DATA=< *libref.* **>***SAS-data-set*

names the SAS data set that contains the information that you want added to the data mining database. If the data set resides on the SAS Appliance, then the SAS Appliance is used during summarization.

Optional Arguments

DMDBCAT=< *libref.* **>***SAS-catalog*

names the metadata catalog to be created by PROC HPDMDB.

CLASSOUT=< *libref.* **>***SAS-data-set*

names the data set to contain the summaries of classification variables that are specified in the CLASS statement.

VAROUT=< *libref.* **>***SAS-data-set*

names the data set to contain the summaries of analysis variables that are specified in the VAR statement.

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. Table 4.1 shows the possible values for VARDEF.

Table 4.1 Values for VARDEF=

Value	Divisor
N	Number of observations (n)
DF (default)	Degrees of freedom ($n - 1$)

PRINT

prints the class-level information to all open ODS destinations.

SPECIALMISSING

enables special missing values to be treated as separate levels.

MAXLEVEL=*max*

specifies the maximum number of levels to be reported for each class variable. If more than *max* levels of a class variable exist, PROC HPDMDB reports the frequency of unreported observations in a level named *_OTHER_*.

CLASS Statement

CLASS *variable (< order-option>) < variable (< order-option>)> . . .* **;**

The CLASS statement specifies the variables whose values define subgroup combinations for the analysis.

The CLASS and VAR statements are mutually exclusive.

Required Argument

variable

specifies a classification variable to be used in the analysis. For each level of a CLASS variable, the CLASSOUT data set contains information about each of the following: the level value, its frequency, and the type of the variable (numeric or character).

Optional Argument

order-option

specifies the order to use when considering the levels of CLASS variables to be sorted. The value of *order-option* can be one of the following:

ASCENDING | ASC

arranges class levels in lowest-to-highest order of unformatted values.

DESCENDING | DESC

arranges class levels in highest-to-lowest order of unformatted values.

ASCFORMATTED | ASCFMT

arranges class levels in ascending order by their formatted values.

DESFORMATTED | DESFMT

arranges class levels in descending order by their formatted values.

DSORDER | DATA

arranges class levels according to the order of their appearance in the input data set.

NOTE: The DSORDER sort option is not supported for input data sets that are stored on the SAS Appliance.

FREQ Statement

FREQ *variable* **;**

The FREQ statement specifies a numeric *variable* that contains the frequency of each observation.

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. For observations where *variable* is 0 or missing, the observation is omitted in the CLASSOUT data set and is not included in statistical calculations.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

With the PERFORMANCE statement you can also control whether a SAS High-Performance Analytics procedure executes in symmetric multiprocessor (SMP) or massively parallel processor (MPP) mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section “PERFORMANCE Statement” on page 29 of Chapter 2, “Shared Concepts and Topics.”

VAR Statement

VAR *variables* **;**

The VAR statement specifies one or more numeric *variables* to analyze. The CLASS and VAR statements are mutually exclusive.

Required Argument

variables

specifies one or more numeric *variables* to be used in the analysis. The *variables* must be numeric. For each *variable*, the VAROUT data set and the metadata contain the following statistics:

Table 4.2 Statistics Recorded for VAR Variables

Statistic	Meaning
N	Number of observations
NMISS	Number of observations that contain a missing value
MIN	Minimum observed value
MAX	Maximum observed value
MEAN	Mean of observed values
STD	Standard deviation
SKEWNESS	Measure of asymmetry
KURTOSIS	Measure of the “heaviness of the tails”
SUM	Sum of all nonmissing observations
CSS	Corrected sum of squares
USS	Sum of squares

(See Appendix 1, “SAS Elementary Statistics Procedures” in *Base SAS Procedures Guide*, for formulas and other details.)

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement specifies a numeric *variable* that contains a weight for each observation. The *variable* is used in the computation of means, standard deviation, skewness, and kurtosis.

Required Argument

variable
represents how the observation should be weighted in statistical calculations. For observations where *variable* is 0 or missing, the observation is still included in the CLASSOUT data set but the value is not used in statistical calculations.

Details: HPDMDB Procedure

The statistics recorded for numeric variables are detailed in the section "VAR Statement" on page 92.

For classification variables, a *level* is a distinct observed value after formatting, removal of beginning and ending white space, and capitalization. For example, the values **`MyLevel`** and **`MYLEVEL`** are treated as a single level in the data set. Classification variables can be numeric, and the same levelization rules apply. For example, `3.000002` and `3.0000001` are treated as the same level if they are formatted using **`BEST3`**.

Frequencies should be integers. If a noninteger frequency is specified, it is rounded to the nearest integer for calculations. Weights do not need to be integers. Negative frequencies and weights are treated as 0.

Examples: HPDMDB Procedure

Example 4.1: Running PROC HPDMDB on the Client

This example demonstrates how to run the HPDMDB procedure on the following data set on the client:

```
data ex;
input x1 x2 x3 x4 y$ w f y2;
cards;
1  2 1 1  m .90 1 0
1  2 1 2  m .91 2 1
1  2 1 3  x .89 1 4
1  2 1 4  x .90 2 4
```

```
1  3 1 1  m .91 1 1
1  3 1 2  m .89 2 1
2  3 1 3  x .90 1 5
2  3 1 4  x .89 2 5
3  1 2 1  z .90 1 2
3  1 2 2  z .89 2 2
3  1 2 3  y .90 1 7
3  1 2 4  y .89 2 7
3  4 2 1  z .90 1 3
3  4 2 2  z .89 2 3
4  4 2 3  y .90 1 6
4  4 2 4  y .89 2 6
;
run;
```

When the input data set resides on the client and no PERFORMANCE statement is specified, as in the following example, the client performs all computations:

```
proc hpdmdb data=ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
run;
```

Output 4.1.1 shows the summaries of the numeric variables in the data set ex.

Output 4.1.1 Summaries of Numeric Variables in ex Data Set

```
Obs    NAME        NMISS           N    MIN    MAX      MEAN        STD

 1      x4           0            24     1      4      2.66326    1.06938
 2      y2           0            24     0      7      3.57721    2.10658

Obs    SKEWNESS    KURTOSIS     SUM       USS       CSS

 1     -0.05848    -1.19599    57.26    178.80     26.302
 2      0.12214    -1.04243    76.91    377.19    102.067
```

Output 4.1.2 shows the summaries of the classification variables in the data set ex.

Output 4.1.2 Summaries of Classification Variables in ex Data Set

```
Obs NAME LEVEL   CODE  FREQUENCY TYPE CRAW NRAW FREQPERCENT NMISSPERCENT

  1  x1    1        0          9  N          1      37.5        37.5
  2  x1    2        1          3  N          2      12.5        12.5
  3  x1    3        2          9  N          3      37.5        37.5
  4  x1    4        3          3  N          4      12.5        12.5
  5  x2    1        0          6  N          1      25.0        25.0
  6  x2    2        1          6  N          2      25.0        25.0
  7  x2    3        2          6  N          3      25.0        25.0
  8  x2    4        3          6  N          4      25.0        25.0
  9  x3    1        0         12  N          1      50.0        50.0
 10  x3    2        1         12  N          2      50.0        50.0
```

Example 4.2: Running with Client Data on the SAS Appliance

This example uses the same data set as is used in Example 4.1.

When the input data set resides on the client and a PERFORMANCE statement with a NODES= option is specified, as in the following example, PROC HPDMDB copies the data set to the SAS Appliance, which does the computations. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the macro variable references in the example with the appropriate values.

```
option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";
/*Perform the computation on the SAS Appliance using 5 nodes*/
proc hpdmdb data=ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
    performance nodes=5 details;
run;
```

The results are the same as those shown in Output 4.1.1 and Output 4.1.2.

Example 4.3: Running with Data on the SAS Appliance

This example uses the same data set as is used in Example 4.1.

When the input data set resides on the SAS Appliance, the SAS Appliance performs all computations and reports the results back to the client. In the following example, the input data resides in the MyLib library, which is a distributed data source. To run these statements successfully, you need to set the macro variables to resolve to appropriate values, or you can replace the macro variable references with the appropriate values.

```
option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";

libname MyLib &LIBTYPE
        server  ="&GRIDDATASERVER"
        user    =&USER
        password=&PASSWORD
        database=&DATABASE;

/*MyLib is a libname for a distributed data source
  In this case, the computation is automatically done
  on the SAS Appliance.*/
proc hpdmdb data=MyLib.ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
    performance details;
run;
```

The results are the same as those shown in Output 4.1.1 and Output 4.1.2.

Chapter 5
The HPDS2 Procedure

Contents

Overview: HPDS2 Procedure

The HPDS2 procedure enables you to submit DS2 language statements from a Base SAS session to a SAS High-Performance Analytics grid for parallel execution. PROC HPDS2 verifies the syntactic correctness of the DS2 source on the client machine before submitting it to the grid for execution. The output data created by the DS2 DATA statement can be output in either of the following ways: it can be written in parallel back to the grid data store or it can be returned to the client machine and directed to any data store that is supported by SAS.

Because the DS2 code is executed in parallel on separate grid nodes with single data partitions, there is separate output from each node that is the result of processing only the local data partition. As a result, it might be necessary to use a second-stage program to aggregate the results from each node. The second stage can be executed on the SAS client by using the DS2 procedure, where the SET statement reads all rows created by the preceding parallel stage.

The syntax of DS2 is similar to that of the DATA step, but it does not include several key statements such as INPUT and MERGE. In addition, using DS2 in the High-Performance Analytics environment limits the PROC DS2 SET statement to a single input stream. The use of BY processing within the SET statement is also not supported. Therefore, many of the traditional DATA step data preparation features are not available in the HPDS2 procedure. PROC HPDS2 is most useful when significant amounts of computationally intensive, row-independent logic must be applied to the data. The DSTRANS procedure, which is documented in the *SAS DS2 Language Reference*, converts a DATA step to DS2 and in the process identifies DATA step syntax that is not compatible with PROC HPDS2.

For more information about the DS2 language, see the *SAS DS2 Language Reference* which is available at `http://support.sas.com/documentation/solutions/ds2/DS2Ref.pdf`.

PROC HPDS2 Features

The HPDS2 procedure provides a vehicle for the parallel execution of DS2 code in a distributed computing environment. The following list summarizes the basic features of the HPDS2 procedure:

- provides the ability to execute DS2 code in parallel
- enables DS2 code to be executed on a local client machine or on the High-Performance Analytics grid
- enables control of the level of parallelism per execution node and the number of nodes to engage
- performs a syntax check of the DS2 code on the local client machine before sending it to the grid for execution
- manages data migration to the location of execution and movement back to the client machine as needed

Client and Grid Execution Modes

The HPDS2 procedure controls the execution of DS2 language statements in two dimensions. You can control both the number of parallel threads per execution node and also the number of compute nodes to engage. The threading provided by PROC HPDS2 operates outside the syntax of the language in contrast to the THREADS PACKAGE DS2 syntax which provides single-node scalability as part of the DS2 syntax.

Alternatively, the HPDS2 procedure can be executed on the High-Performance Analytics grid. In grid mode, one or more copies of the DS2 program are executed in parallel on each grid node.

The grid mode of execution has two variations:

- In the client-data (local-data) model of grid execution, the input data are not stored on the appliance but are distibuted to the distributed computing environment by the High-Performance Analytics framework during execution of the HPDS2 procedure.
- In the alongside-the-database model of grid execution, the data source is the database on the appliance. The data are stored in the distributed database, and the DS2 program running on each node is able to read and write the data in parallel during execution of the procedure. Instead of data being moved across the network and possibly back to the client machine, data are passed locally between the processes on each node of the appliance. In general, especially with large data sets, the best HPDS2 performance can be achieved if execution is alongside the database.

By default, the number of copies of the DS2 program that are executed in parallel on a given host (that is, client machine or grid node) is determined by the HPDS2 procedure based on the number of CPUs (cores) available on the host machine. The default is to execute one instance of the DS2 program in a dedicated thread per CPU. This can be changed by means of the NTHREADS= option in the PERFORMANCE statement. For example, if NTHREADS=*n* is specified, then the HPDS2 procedure runs *n* copies of the DS2 program in parallel on each grid node.

The HPDS2 procedure can run in either symmetric multiprocessing (SMP) mode or in massively parallel processing (MPP) mode. For information about the available modes of execution of SAS High-Performance Analytics procedures and how to switch between them, see the section "SMP and MPP Modes" on page 10 of Chapter 2, "Shared Concepts and Topics."

Getting Started: HPDS2 Procedure

This example illustrates a simple HPDS2 procedure. In this case, the DS2 source statements are executed alongside the database in MPP mode. The DS2 code that is submitted to the grid is contained within the DATA and ENDDATA statements. The following DATA step creates a data set that consists of fictitious daily temperatures collected from a number of U.S. airports during a period of one week:

```
data daily_temps;
   input city $ mon tue wed thu fri;
datalines;
lax 88 92 94 97 86
sfo 65 60 75 72 74
nyc 99 95 94 95 90
phl 92 89 91 93 94
atl 95 99 92 98 94
den 85 87 89 72 73
pit 68 70 72 73 77
rdu 98 95 99 95 96
dtt 88 90 90 87 92
anc 51 56 60 64 62
sea 72 78 77 80 79
msy 98 97 99 98 99
mia 95 92 98 94 96
ord 87 85 84 80 79
```

```
dfw 95 96 97 95 97
hou 98 99 98 97 92
las 104 105 102 99 101
pdx 78 82 76 74 80
san 80 81 78 82 80
phx 95 98 95 97 100
cle 75 72 76 80 78
ont 78 80 78 81 72
tpa 94 94 92 90 92
bos 80 78 77 75 79
clt 83 80 79 80 81
;
run;
```

The HPDS2 procedure reads this data set and calculates a daily average temperature in Fahrenheit and Celsius for each airport and then provides a synopsis of the weekly temperature average.

In the following statements, the driver DS2GTF.out in the DATA statement names the input data set, and the SET DS2GTF.in statement names the output data set:

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;

proc hpds2 data=daily_temps
           out=applianc.avg_temps;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
      dcl double avgf avgc;
      dcl char(5) synopsis;
      method run();
         set DS2GTF.in;
         avgf = mean(mon, tue, wed, thu, fri);
         avgc = round((avgf - 32.0) * 5.0/9.0, .1);
         if avgf >= 95.0 then synopsis = 'Hot';
         else if avgf > 80.0 then synopsis = 'Warm';
         else if avgf > 60.0 then synopsis = 'Mild';
         else synopsis = 'Cold';
      end;
   enddata;
run;
```

The following PRINT procedure displays the table of average temperatures that are produced by the HPDS2 procedure:

```
proc print data=applianc.avg_temps;
   title1 'Average Temperatures';
   var city synopsis avgf avgc;
run;
```

Figure 5.1 displays the output of the PRINT procedure.

Figure 5.1 Average Temperatures

```
                    Average Temperatures

 Obs    city    synopsis     avgf     avgc

   1    lax       Warm       91.4     33.0
   2    sfo       Mild       69.2     20.7
   3    nyc       Warm       94.6     34.8
   4    phl       Warm       91.8     33.2
   5    atl       Hot        95.6     35.3
   6    den       Warm       81.2     27.3
   7    pit       Mild       72.0     22.2
   8    rdu       Hot        96.6     35.9
   9    dtt       Warm       89.4     31.9
  10    anc       Cold       58.6     14.8
  11    sea       Mild       77.2     25.1
  12    msy       Hot        98.2     36.8
  13    mia       Hot        95.0     35.0
  14    ord       Warm       83.0     28.3
  15    dfw       Hot        96.0     35.6
  16    hou       Hot        96.8     36.0
  17    las       Hot       102.2     39.0
  18    pdx       Mild       78.0     25.6
  19    san       Warm       80.2     26.8
  20    phx       Hot        97.0     36.1
  21    cle       Mild       76.2     24.6
  22    ont       Mild       77.8     25.4
  23    tpa       Warm       92.4     33.6
  24    bos       Mild       77.8     25.4
  25    clt       Warm       80.6     27.0
```

Syntax: HPDS2 Procedure

The following statements are available in the HPDS2 procedure:

```
PROC HPDS2 < options > ;
    PERFORMANCE performance-options ;
    DATA DS2GTF.out ;
          DS2 statements ;
          METHOD RUN() ;
              SET DS2GTF.in ;
          END ;
    ENDDATA ;
    RUN ;
    RUN CANCEL ;
    QUIT ;
```

PROC HPDS2 Statement

PROC HPDS2 < *options* > **;**

The PROC HPDS2 statement invokes the procedure.

You can specify the following options in the PROC HPDS2 statement:

DATA=*SAS-data-set*

IN=*data-set*

names the SAS data set or database table to be used by PROC HPDS2. The default is the most recently created data set.

OUTPUT=*data-set*

OUT=*data-set*

names the SAS data set or database table created by PROC HPDS2.

DATA Statement

DATA DS2GTF.out ;

The DATA statement indicates the beginning of the DS2 code block. The code block terminates with the ENDDATA statement.

A reference to the DS2 Grid Table Function driver (DS2GTF.out) must be included as part of the DATA statement. If an input data set is specified in the PROC HPDS2 statement, then a **`run()`** method should be included in the DS2 code block. The first statement after the METHOD RUN() statement must be the SET DS2GTF.in statement for this case. DS2GTF.out and DS2GTF.in refer to the input and output data sets, respectively.

ENDDATA Statement

ENDDATA ;

The ENDDATA statement terminates the DS2 code block. The statements between the DATA and ENDDATA statement are submitted to the grid for execution. The DS2 run, init, and term methods are specified between the DATA and ENDDATA statements.

PERFORMANCE Statement

PERFORMANCE *performance-options* **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a High-Performance Analytics procedure.

With the PERFORMANCE statement, you can also control whether a High-Performance Analytics procedure executes in SMP or MPP mode.

It is important to remember the distinction between the NODES= and NTHREADS= options. The NODES= option specifies the number of separate grid nodes that participate in the DS2 execution, while the NTHREADS= option determines how many independent copies of the DS2 program are run in parallel on each node. If the data are located on the grid, then all nodes must be engaged; therefore, the NODES= option might be overridden. Setting NODES=0 causes the DS2 code to execute on the client side only. Setting the NTHREADS= option to a value that is greater than the CPU count on each grid node is not likely to improve overall throughput.

The PERFORMANCE statement for High-Performance Analytics procedures is documented in section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

QUIT Statement

QUIT ;

The QUIT statement stops the procedure. PROC HPDS2 statements that have not been submitted with a RUN statement are terminated.

RUN Statement

RUN ;

The RUN statement submits the preceding PROC HPDS2 statements for execution.

The procedure requires the RUN statement to submit the statements. That is, SAS reads the program statements that are associated with one task until it reaches a RUN statement.

RUN CANCEL Statement

RUN CANCEL ;

The RUN CANCEL statement cancels the preceding PROC HPDS2 statements. RUN CANCEL is useful if you enter a typographical error.

Details: HPDS2 Procedure

Parallel Execution of DS2 Code

An important characteristic of multithreaded or distributed applications is that they might produce nondeterministic or unpredictable results. The exact behavior of a DS2 program running in parallel on the grid is influenced by a number of factors, including the pattern of data distribution that is used, the execution mode that is chosen, the number of compute nodes and threads that are used, and so on. The HPDS2 procedure does not examine whether the DS2 code that is submitted produces meaningful and reproducible results. It simply executes the DS2 code that is provided on each of the units of work, whether these are multiple threads on a single machine or multiple threads on separate grid nodes. Each instance of the DS2 program operates on a subset of the data. The results produced by each unit of work are then gathered, without further aggregation, into the output data set.

Because the DS2 code instances are executed in parallel, consideration must be given to the DS2 language elements that are included in the DS2 code block of an HPDS2 procedure. Not all DS2 language elements can be meaningfully used in multithreaded or distributed applications. For example, lagging or retaining of variables can imply ordering of observations. A deterministic order of observations does not exist in distributed applications, and enforcing data order might have a negative impact on performance.

Optimal performance is achieved when the input data are stored in the distributed database and the grid host is the appliance that houses the data. With the data distributed in this manner, the different instances of the DS2 code running on the grid nodes can read the input data and write the output data in parallel from the local database management system (DBMS).

Limitations and Issues

The current release of the HPDS2 procedure does not support all of the features of the DS2 language. The following subsections summarizes the known limitations and issues for PROC HPDS2.

Packages

DS2 packages are collections of variables and methods that can be used in DS2 programs and threads. The HPDS2 procedure does not support DS2 packages at this time. Use of the PACKAGE and ENDPACKAGE constructs within an HPDS2 procedure results in an error. Similarly, existing packages cannot be referenced within an HPDS2 procedure.

PERFORMANCE Statement Options

The maximum allowed value for the CPUCOUNT= option in the PERFORMANCE statement is 256. However, setting CPUCOUNT= to very high values, including values that substantially exceed the actual number of available CPUs, can result in unpredictable errors, including DS2 program instances that unexpectedly produce no observations in the output data set.

Setting the NTHREADS= option in the PERFORMANCE statement to very high values can cause out-of-memory errors. For example, out-of-memory errors have been seen with NTHREADS=100.

Data Input/Output

If an input data set is specified, then a SET DS2GTF.in statement must be included in the METHOD RUN() statement. If either the SET DS2GTF.in or the SET DS2GTF.out driver reference is missing, then the SAS session stops responding.

The use of BY groups within the SET statement of an HPDS2 procedure is not supported at this time.

The use of nested SQL within the SET statement of an HPDS2 procedure is not supported at this time.

When used within an HPDS2 procedure, the PUT statement does not currently write any data to the client log.

The OVERWRITE= option is not supported in PROC HPDS2.

Attempting to drop the only variable in a one-variable input data set from within an HPDS2 procedure might cause SAS to stop responding or might result in an exception.

Data Types and Declarations

The HPDS2 procedure does not support the following data types: REAL, TINYINT, NCHAR, TIMESTAMP, DATE, and TIME. If any of these data types are declared within an HPDS2 procedure, then an error is displayed.

User-defined formats are not currently supported in PROC HPDS2.

Formats, informats, and labels specified in the HAVING clause in the DECLARE statement are ignored by the HPDS2 procedure.

Delimited identifiers (for example, dcl double "a%& b") are not currently supported in PROC HPDS2.

No warning or error messages are output when assignments involving out-of-bounds arrays are used within an HPDS2 procedure.

Error Messages

Incorrect source line numbers are reported when there is an error in an HPDS2 procedure. In addition, the ordering of error messages displayed is reversed for PROC HPDS2 from the order of error messages that is output for DS2.

Examples: HPDS2 Procedure

Example 5.1: Compute Mandelbrot Set

This example computes and plots a Mandelbrot set. The DS2 source statements that compute the set of coordinates that comprise the Mandelbrot set are submitted to the grid and executed alongside the database in MPP mode. Note that Mandelbrot set computation is perfectly scalable in that each point can be computed independently of every other point.

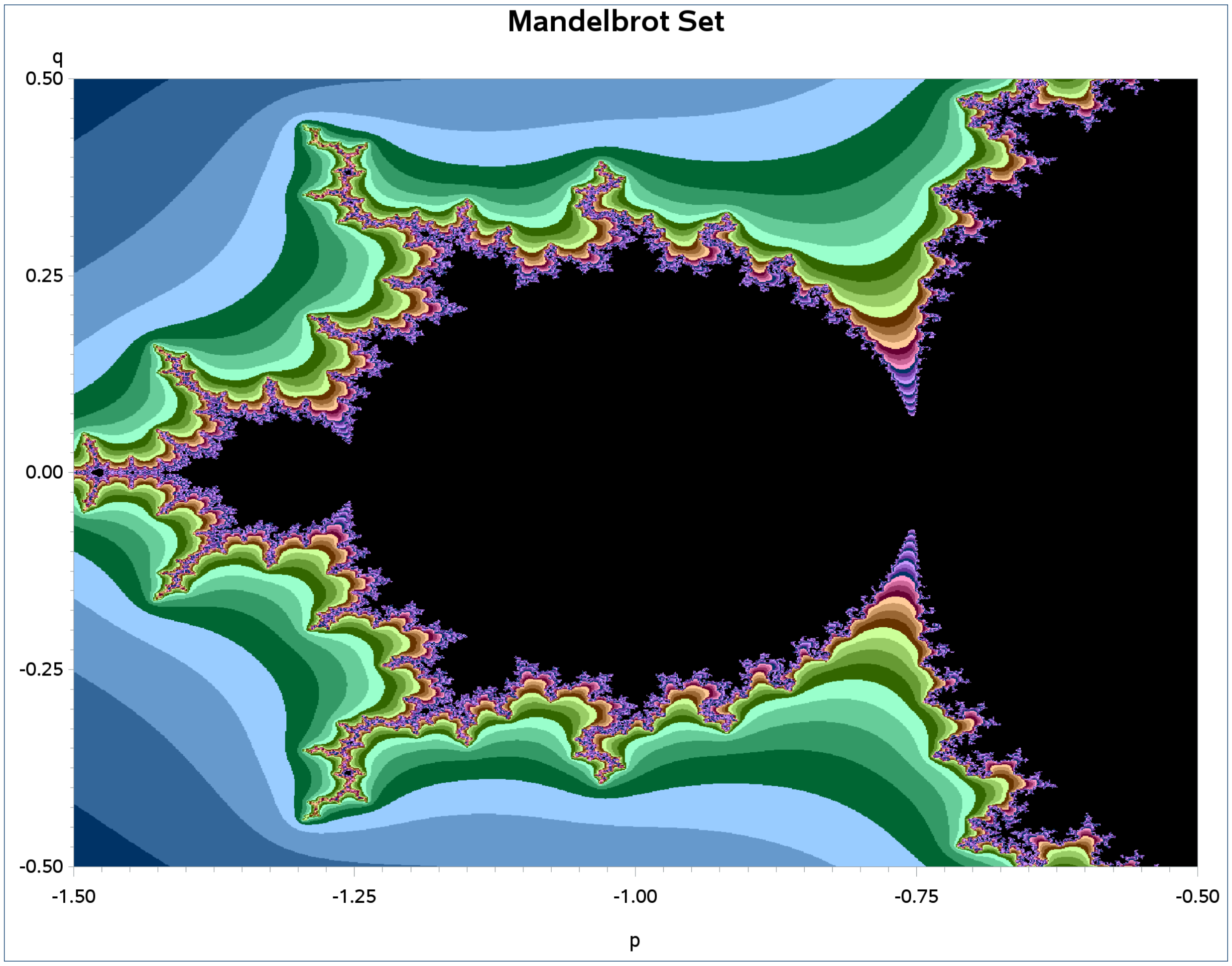
This example uses a DS2 procedure to create a data set that consists of one row for each Mandelbrot coordinate to be computed. The HPDS2 procedure reads this data set and computes the coordinates. The Mandelbrot set is then graphed via the GCONTOUR procedure.

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;

/* Set up the table that contains one row for each coordinate to compute */
proc ds2;
   data inp(overwrite=yes);
      dcl double p q r;
      dcl integer maxiterate;
      method init();
         dcl int n m;
         dcl int i j k;
         dcl double pmin pmax qmin qmax;
         n = 1024;
         m = 1024;
         pmin = -1.5; pmax = -0.5;
         qmin = -0.5; qmax =  0.5;
         r = 100.0;
         maxiterate = 50;
         do k = 1 to n*m;
            i = k/m;
            j = mod(k,m);
            p = i*(pmax-pmin)/(n-1)+pmin;
            q = j*(qmax-qmin)/(m-1)+qmin;
            output;
         end;
      end;
   enddata;
run;
quit;

/* Compute the coordinates */
proc hpds2 data=inp out=applianc.mandelbrot;
```

```
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
      dcl int mesh;
      dcl double x y rr nx ny;
      keep p q mesh;
      method run();
         set DS2GTF.in;
         x = p;
         y = q;
         rr = r**2;
         mesh = 0;
         do while (mesh < maxiterate and (x**2+y**2 < rr));
            nx = x**2 - y**2 + p;
            ny = 2.0*x*y + q;
            x = nx;
            y = ny;
            mesh = mesh+1;
         end;
      end;
   enddata;
run;

/* Plot the results */
goptions colors= (
CX003366 CX336699 CX6699CC CX99CCFF CX006633 CX339966 CX66CC99 CX99FFCC
CX336600 CX669933 CX99CC66 CXCCFF99 CX663300 CX996633 CXCC9966 CXFFCC99
CX660033 CX993366 CXCC6699 CXFF99CC CX003366 CX663399 CX9966CC CXCC99FF
CX003366 CX663399 CX9966CC CXCC99FF CX003366 CX663399 CX9966CC CXCC99FF
CX003366 CX663399 CX9966CC CXCC99FF CX003366 CX663399 CX9966CC CXCC99FF
black
)
;

proc gcontour data=applianc.mandelbrot;
   Title 'Mandelbrot Set';
   plot q*p=mesh /
   nolegend
   pattern
   join
   levels = 5 to 45
  ;
run;
```

Output 5.1.1 shows the graphic representation of the Mandelbrot set computed by the HPDS2 procedure.

Output 5.1.1 Computed Mandelbrot Set



Example 5.2: Aggregate Result Data Set

This example illustrates how the intermediate result data that are generated from the DS2 code running in parallel on separate grid nodes can be aggregated into a final result data set. In this case, the aggregation is done by a second-stage PROC DS2 program that executes on the SAS client.

This example uses a DATA step program running on the SAS client to generate a sample data set that consists of dimensional information for each of 200 objects (closed cylinders). These data are used by the HPDS2 procedure to calculate the volume and surface area of each object. The second-stage DS2 procedure aggregates these results, summing the total volume and surface area for all objects and computing the average volume and surface area. Notice in this example that the DS2 code running in parallel on the grid is used to perform the row-independent and computationally intensive portion of the processing, while the work done by the second-stage DS2 procedure is limited to the final result aggregation and summary.

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;

data obj_dims;
   do id=1 to 200;
      radius = ranuni(1) * 10;
      height = ranuni(1) * 20;
      output;
   end;
run;

%let pi=3.14159;
proc hpds2 data=obj_dims
           out=applianc.obj_comps;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
      method run();
         set DS2GTF.in;
         volume = &pi * radius**2 * height;
         area = (2 * &pi * radius**2) + (2 * &pi * radius * height);
      end;
   enddata;
run;

proc print data=applianc.obj_comps (obs=20);
   title1 'Volumes and Areas';
run;

data obj_comps;
   set applianc.obj_comps;
run;

proc ds2;
   data obj_totals (keep = (ncount vsum asum vmean amean));
      dcl double ncount vsum asum vmean amean;
      method init();
         ncount = 0;
         vsum = 0;
         asum = 0;
      end;
      method run();
         set {select volume, area from obj_comps};
         ncount + 1;
         vsum + volume;
         asum + area;
      end;
      method term();
         if ncount ne 0 then do;
            vmean = vsum/ncount;
            amean = asum/ncount;
         end;
         output;
      end;
   enddata;
```

```
run;
quit;

proc print data=obj_totals;
   title1 'Total Volume and Area';
run;
```

Output 5.2.1 shows a subset of the volumes and areas computed by the HPDS2 procedure.

Output 5.2.1 Computed Volumes and Areas

```
                        Volumes and Areas

        Obs    id     radius     height     volume       area

          1     1    1.84963    19.4018     208.53     246.97
          2     2    3.99824     5.1880     260.55     230.77
          3     3    9.21603    19.3855    5172.67    1656.20
          4     4    5.42979    10.6338     984.93     548.03
          5     5    0.49794     1.3313       1.04       5.72
          6     6    8.19319    10.4774    2209.58     961.15
          7     7    8.53394     1.3437     307.43     529.64
          8     8    9.57024     5.9439    1710.27     932.89
          9     9    2.72612    13.7986     322.16     283.05
         10    10    9.76765     4.5302    1357.82     877.48
         11    11    6.88237     8.2553    1228.45     654.60
         12    12    5.58554     5.7445     563.03     397.63
         13    13    4.75789    16.8997    1201.87     647.45
         14    14    6.34524    11.8073    1493.47     723.71
         15    15    5.82582     7.5403     803.99     489.26
         16    16    7.28362    10.1321    1688.66     797.02
         17    17    9.31214    18.5824    5062.32    1632.10
         18    18    5.89660     5.9445     649.33     438.70
         19    19    3.91042     9.4486     453.90     328.23
         20    20    6.79526     3.3618     487.67     433.66
```

Output 5.2.2 shows the aggregated results produced by the second-stage DS2 program.

Output 5.2.2 Computed Total Volume and Area

```
                      Total Volume and Area

    Obs    NCOUNT       VSUM         ASUM       VMEAN      AMEAN

     1       200     209883.99    104680.26    1049.42    523.401
```

Chapter 6
The HPFOREST Procedure (Experimental)

Contents

Overview: HPFOREST Procedure

The HPFOREST procedure is a high-performance procedure that creates a predictive model called a *forest* that consists of several decision trees. A *predictive model* defines a relationship between input variables and a target variable. The purpose of a predictive model is to predict a target value from inputs. The HPFOREST procedure *trains* the model; that is, it creates the model, using *training data* in which the target values are known. The model can then be applied to observations in which the target is unknown. If the predictions fit the new data well, the model is said to *generalize* well. Good generalization is the primary goal for predictive tasks. A predictive model might fit the training data well but generalize poorly.

A *decision tree* is a type of predictive model that has been developed independently in the statistics and artificial intelligence communities. The HPFOREST procedure creates a tree recursively. An input variable is chosen and used to create a rule to split the data into two segments. The process is then repeated in each segment, and then again in each new segment, and so on until some constraint is met. In the terminology of the tree metaphor, the segments are *nodes*, the original data set is the *root* node, and the final unpartitioned segments are *leaves* or *terminal nodes*. A node is an *internal node* if it is not a leaf. The data in a leaf determine the estimates of the value of the target variable. These estimates are subsequently applied to predict the target of a new observation assigned to the leaf.

The HPFOREST procedure creates decision trees that differ from each other in two ways. First, the training data for a tree is a sample, without replacement, from the original training data of the forest. Second, the input variables considered for splitting a node are randomly selected from all available inputs. Among these variables, the HPFOREST procedure considers only a single variable when forming a splitting rule. The chosen variable is the one that is most associated with the target.

Training a forest can require training hundreds of decision trees. The HPFOREST procedure can exploit computer grids by training trees in parallel independently on different grid nodes. See the section “SMP and MPP Modes” on page 10 in Chapter 2, “Shared Concepts and Topics,” for details about the execution of SAS High-Performance Analytics procedures on a grid of computers.

PROC HPFOREST Features

The HPFOREST procedure creates an ensemble of hundreds of decision trees to predict a single target of either interval or nominal measurement level. An input variable can have an interval, ordinal, or nominal measurement level. The procedure can run on a grid of computers like other SAS High-Performance Analytics procedures.

The HPFOREST procedure deletes from the training data any observation that has a missing value or a FREQ variable whose value is less than or equal to 0.

PROC HPFOREST Contrasted with Other SAS Procedures

No other SAS procedure creates a forest of decision trees for predictive modeling. SAS® Enterprise Miner™ has had three procedures (the ARBOR, DMSPLIT, and SPLIT procedures) during its history that create a single decision tree for predictive modeling. These three procedures search for a split on every variable in every node. The HPFOREST procedure searches for a split on only one variable in a node: the variable with the largest association with the target among candidates randomly selected in that node. Consequently, the HPFOREST procedure creates different trees than the other three procedures.

The syntax for the HPFOREST procedure follows the syntax of the ARBORETUM procedure in SAS Enterprise Miner, except for the PERFORMANCE statement: The HPFOREST procedure uses the same syntax for the PERFORMANCE statement used in other SAS High-Performance Analytics procedures, which is different from the syntax in the ARBORETUM procedure. The values of a variable in a FREQ statement in PROC HPFOREST are adjusted to the nearest integer, as is done in other SAS High-Performance Analytics procedures, and not as is done in the ARBORETUM procedure.

The HPFOREST procedure can run on a grid of computers as can other SAS High-Performance Analytics procedures. For general contrasts between SAS High-Performance Analytics procedures and other SAS procedures, see the section "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10.

Getting Started: HPFOREST Procedure

The HPFOREST procedure can help identify the right mix of ingredients for a good bowl of punch, as this hypothetical example illustrates. The punch contains a mix of watermelon, pineapple, and orange fruit. Six different proportions of the fruits are tasted by three different volunteers who give a numerical rating to each mix. The following DATA step creates the SAS data set Punch with the fruit proportions and ratings:

```
data punch;
   input watermelon pineapple orange rating;
   datalines;
   1.0 0.0 0.0 4.3
   1.0 0.0 0.0 4.7
   1.0 0.0 0.0 4.8
   0.0 1.0 0.0 6.2
   0.0 1.0 0.0 6.5
   0.0 1.0 0.0 6.3
   0.5 0.5 0.0 6.3
   0.5 0.5 0.0 6.1
   0.5 0.5 0.0 5.8
   0.0 0.0 1.0 7.0
   0.0 0.0 1.0 6.9
   0.0 0.0 1.0 7.4
   0.5 0.0 0.5 6.1
   0.5 0.0 0.5 6.5
   0.5 0.0 0.5 5.9
   0.0 0.5 0.5 6.2
   0.0 0.5 0.5 6.1
   0.0 0.5 0.5 6.2
   ;
 run;
```

Use the INPUT and TARGET statements in the HPFOREST procedure to specify the fruit and rating variables, respectively. The procedure assumes that the variables have an interval level of measurement because the variables are numeric. The following statements use MAXTREES=10 to limit the number of trees in the forest, and thereby reduce the output listing. More trees are appropriate for larger data sets.

```
proc hpforest data=Punch maxtrees=10;
   input watermelon pineapple orange;
   target rating;
run;
```

Figure 6.1 HPFOREST Getting Started Example Output

```
                         Number of Observations

           Type                                        N

           Number of Observations Read                18
           Number of Observations Used                18

                            Model Information

      Parameter                       Value

      Category Bins                      30    (Default)
      Leaf Size                           5    (Default)
      Maximum Depth                      50    (Default)
      Maximum Trees                      10
      Minimum Category Size               5    (Default)
      Variables to Try                    2    (Default)
      Alpha                            0.05    (Default)
      Exhaustive                       5000    (Default)
      Leaf Fraction                   0.001    (Default)
      Train Fraction                    0.6    (Default)
      Split Criterion                     3    Variance
      Missing Value Handling              1    Valid value

                              Fit Statistics

                                Average         Average
                                 Square          Square
                                  Error           Error
        NTrees    NLeaves    (Full Data)           (OOB)

             1          2       0.389070        0.293265
             2          4       0.148367        0.238791
             3          6       0.130264        0.221248
             4          8       0.121850        0.181604
             5         10       0.116800        0.164337
             6         12       0.119184        0.153470
             7         14       0.120291        0.151718
             8         16       0.124535        0.151397
             9         18       0.120974        0.197850
            10         20       0.123476        0.176035

                                 Timing

                 Task                  Time(sec.)

                 Reading Data                0.02
                 Training Forest             0.00
```

The listing shows the number of observations used (18), the training options, and then the average square error for several forests. The forests differ by the number of trees they contain. Forest models provide an alternative estimate of average square error, called the *out-of-bag* (OOB) estimate. For more information, see the section "Bagging the Data" on page 119. The output shows that the out-of-bag error estimate is worse (larger) than the estimate that evaluates all observations on all trees. This is usual. The output shows also that a forest with several trees has a better (smaller) error estimate than a single tree. This is good. Additional trees help. In this example, the minimum OOB error occurs with eight trees, indicating that the forest with eight trees is appropriate for predicting new data.

Syntax: HPFOREST Procedure

The following statements are available in the HPFOREST procedure:

> **PROC HPFOREST** < *option(s)* > **;**
>   **INPUT** *variable(s)* < *option(s)* > **;**
>   **TARGET** *variable* < *option(s)* > **;**
>   **FREQ** *variable* **;**
>   **PERFORMANCE** *performance-options* **;**

The PROC HPFOREST statement, INPUT, and TARGET statements are required. The INPUT statement can appear multiple times.

PROC HPFOREST Statement

> **PROC HPFOREST** < *options* > **;**

The PROC HPFOREST statement invokes the procedure. You can specify one or more of the following optional arguments.

DATA=< *libref.* **>***SAS-data-set*

names the SAS data set to be used by PROC HPFOREST for training the model. The default is the most recently created data set.

If the data are already distributed, the procedure reads the data alongside the distributed database. See the section "SMP and MPP Modes" on page 10 for the various execution modes and the section "Alongside-the-Database Execution" on page 15 for the alongside-the-database model. Data from all the computer grid nodes are combined into a structure that is optimized for model training and redistributed to the nodes. The different nodes then proceed independently with identical data to create decision trees.

ALPHA=*number*

specifies a threshold p-value for the significance level of a test of association of a candidate variable with the target. If no association meets this threshold, the node is not split. The default value is 0.05.

CATBINS=*k*

specifies the maximum number of categories of a nominal candidate variable to use in the association test. *k* refers only to the categories that are present in the training data in the node and that satisfy the MINCATSIZE= option. The categories are counted independently in each node. If more than *k* categories are present, then the least frequent categories are removed from the association test. Many infrequent categories can dilute a strong predictive ability of common categories. The search for a splitting rule uses all categories that satisfy the MINCATSIZE= options. The value of *k* must be a positive integer. The default value is 30.

EXHAUSTIVE=*number*

specifies the maximum number of splits to examine in a complete enumeration of all possible splits when the input variable is nominal and the target has more than two nominal categories. The exhaustive method of searching for a split examines all possible splits. If the number of possible splits is greater than *number*, then a heuristic search is done instead of an exhaustive search. The default value of *number* is 5,000.

LEAFFRACTION=*f*

specifies the smallest number of training observations that a new branch can have, expressed as the fraction of the number *N* of available observations in the DATA= data set. *N* might be less than the total number of observations in the data set because observations with a missing target value or nonpositive value of the variable specified in the FREQ statement are excluded from *N*. If you specify a number in the LEAFSIZE= option that implies a larger number than that specified in the LEAFFRACTION= option, *f* is ignored. The value *f* must be larger than 0 and less than 1. The default value is 0.001.

LEAFSIZE=*n*

specifies the smallest number of training observations a new branch can have. If you specify a value for the LEAFFRACTION= option that implies a larger value than *n*, the LEAFSIZE= option is ignored. The default value is 5.

MAXDEPTH=*d*

specifies the maximum depth of a node in any tree that PROC HPFOREST creates. The depth of a node equals the number of splitting rules needed to define the node. The root node has depth 0. The children of the root have depth 1, and on. The smallest acceptable value of *d* is 1. The default value of *d* is 50.

MAXTREES=*n*

specifies the number of trees in the forest. *n* is a positive integer. The number of trees in the resulting forest can be less than *n* when the HPFOREST procedure fails to split the training data for a tree. Up to two times *n* trees are attempted. If the procedure fails to split the training data for more than *n* trees, then less than *n* trees are created. The ALPHA=, LEAFSIZE=, and MINCATSIZE= options constrain the split search to form trees that are more likely to predict well using new data. Setting all of these options to 1 generally frees the search algorithm to find a split and train a tree, although the tree might not help the forest predict well. The default value of *n* is 50.

MINCATSIZE=*n*

specifies the minimum number of observations that a given nominal input category must have in order to use the category in a split search. Categorical values that appear in fewer than *n* observations are handled as if they were missing. The categories that occur in fewer than *n* observations are merged into the pseudo category for missing values for the purpose of finding a split. The policy for assigning such observations to a branch is the same as the policy for assigning missing values to a branch. The default value of *n* is 5.

MINUSEINSEARCH=*n*

specifies the minimum number of observations with a missing value in a node to initiate the USEINSEARCH policy for missing values. See the section "Missing Values" on page 122 for a more complete explanation. The default value of *n* is 1.

MISSING=*policy*

specifies how the training procedure handles an observation with missing values. *policy* may equal USEINSEARCH or DISTRIBUTE. If *policy* equals USEINSEARCH and enough training observations in the node are missing the value of the candidate variable, then the missing value is used as a separate, legitimate value in the test of association and the split search. If *policy* equals DISTRIBUTE, observations with a missing value of the candidate variable are omitted from the test of association and split search in that node. A splitting rule distributes such an observation to all branches. See the section "Missing Values" on page 122 for a more complete explanation. The default value of *policy* is USEINSEARCH.

SEED=*n*

specifies the seed for generating random numbers. The HPFOREST procedure uses random numbers to select training observations for each tree and to select candidate variables in each node to split on. *n* is a nonnegative integer. Set *n* to 0 to use the internal default. The default value of the seed is 8,976,153.

SPLITSIZE=*n*

specifies the requisite number of training observations a node must have for the HPFOREST procedure to consider splitting it. By default, *n* is twice the value of the LEAFSIZE= option (or *n* is the value implied by LEAFFRACTION= option if the procedure ignores the LEAFSIZE= option). The procedure counts the number of observations in a node without adjusting the number with the values of the variable specified in the FREQ statement when it interprets the value specified in the LEAFFRACTION=, LEAFSIZE=, MINCATSIZE=, and SPLITSIZE= options.

TRAINFRACTION=*f*

specifies the fraction of training observations to train a tree with. Using less than all the available data often improves the generalization error. A different training sample is taken for each tree. *f* can be any number greater than 0 and at most 1. The default value of *f* is 0.6. *f* must be large enough to ensure at least three training observations in each tree, and large enough to accommodate the LEAFSIZE=, LEAFFRACTION=, and SPLITSIZE options. The TRAINN= option accepts an absolute number instead of a fraction to specify the same quantity. Specifying both the TRAINN= and TRAINFRACTION= is an error.

TRAINN=*n*

specifies how many observations to use to train each tree. The observations are counted without regard to the variable specified in the FREQ statement. Using less than all the available data often improves the generalization error. A different training sample is taken for each tree. *n* can be any positive integer. If *n* is greater than the number of observations in the data set specified in the DATA= option, then all the available data are used. *n* must be at least 3 and large enough to accommodate the values of the LEAFSIZE=, LEAFFRACTION=, and SPLITSIZE options. The default value is 0.6 times the number of available observations in DATA= data set. The TRAINFRACTION= option accepts a fraction instead of an absolute number to specify the same quantity as the TRAINN= option. Specifying both the TRAINN= and TRAINFRACTION= is an error.

VARS_TO_TRY=*m*

specifies the number of input variables to consider splitting on in a node. *m* ranges from 1 to the number of input variables, v. The default value of *m* is $\sqrt{v}$.

INPUT Statement

INPUT *variable(s) < option(s) >* **;**

The INPUT statement names input variables with common options. The INPUT statement can be repeated. You can specify the following *options*:

LEVEL=*level*

specifies the level of measurement of the variables. Accepted values of *level* are: BINARY, NOMINAL, ORDINAL, and INTERVAL.

ORDER=*order*

specifies the sorting order of the values of an ordinal input variable. Table 6.1 provides recognized values of *order*.

Table 6.1 ORDER= Option Values

Value of ORDER=	Variable Values Sorted By
ASCENDING	Ascending order of unformatted values (default)
ASCFORMATTED	Ascending order of formatted values
DESCENDING	Descending order of unformatted values
DESFORMATTED	Descending order of formatted values
DSORDER	Order of appearance in the input data set

NOTE: The DSORDER sort option is not supported for input data sets stored on the SAS appliance.

TARGET Statement

TARGET *variable* < **LEVEL=***level* > **;**

The TARGET statement names the variable whose values PROC HPFOREST tries to predict. You can specify the following optional argument:

LEVEL=*level*

specifies the level of measurement. Accepted values of *level* are: BINARY, NOMINAL, and INTERVAL. Note that *level* cannot be ORDINAL.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. SAS High-Performance Analytics procedures that support the FREQ statement treat each observation as if it appeared f times, where f is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

PERFORMANCE Statement

PERFORMANCE < *performance-options*> **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of the HPFOREST procedure.

With the PERFORMANCE statement, you can also control whether the HPFOREST procedure executes in symmetric multiprocessing or massively parallel mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

Details: HPFOREST Procedure

Bagging the Data

A decision tree in a forest trains on new training data that are derived from the original training data presented to the HPFOREST procedure. Training different trees with different training data reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the forest.

The HPFOREST procedure samples the original data *without* replacement to create the training data for an individual tree. Most forest algorithms sample *with* replacement. The convention of sampling with replacement originated with Leo Breiman's *bagging* algorithm (Breiman 1996, 2001). The word *bagging* stems from "bootstrap aggregating," where "bootstrap" refers to a procedure that uses sampling with replacement. Breiman refers to the observations that are excluded from the sample as *out-of-bag* (OOB) observations. Therefore, observations in the training sample are called the *bagged* observations, and the training data for a specific decision tree are called the *bagged data*. Subsequently, Freedman and Popescu (2003) argued that sampling *without* replacement can provide more variability between the trees, especially with larger training sets.

The TRAINN= and TRAINFRACTION= options in the PROC HPFOREST statement specify the number of observations to sample without replacement into a bagged data set.

Estimating the goodness-of-fit of the model by using the training data is usually too optimistic; the fit of the model to new data is usually worse than the fit to the training data. Estimating the goodness-of-fit by using the out-of-bag data is usually too pessimistic at first. With enough trees, the out-of-bag estimates are an unbiased estimate of the generalization fit.

Training a Decision Tree

The HPFOREST procedure trains a decision tree by forming a binary split of the bagged data, then forming a binary split of each of the segments, and so on recursively until some constraint is met.

Creating a binary split involves a few subtasks:

1. selecting candidate inputs
2. reducing the number of nominal input categories
3. computing the association of each input with the target
4. searching for the best split using the most highly associated input

The procedure selects candidate inputs independently in every node. The purpose of preselecting candidate inputs is to increase the differences between the trees, thereby decreasing the correlation and theoretically increasing the quality of the forest predictions. The selection is random. Each input has the same chance. The VARS_TO_TRY= option specifies the number of candidates to select. The quality of the forest often depends on the number of candidates. Unfortunately, a good value for the VARS_TO_TRY= option is generally not known in advance. Data with more irrelevant variables generally warrant a larger value.

The reason for searching only one input variable for a splitting rule instead of searching all inputs and choosing the best split is to improve prediction on new data. An input that offers more splitting possibilities provides the search routine more chances to find a spurious split. Loh and Shih (1997) demonstrate the bias towards spurious splits that result. They also demonstrate that preselecting the input variable and then searching only on that one input reduces the bias. The HPFOREST procedure preselects the input with the largest p-value of an asymptotic permutation distribution of an association statistic. Hothorn, Hornik, and Zeileis (2006) originated the idea and describe the statistic.

The procedure sometimes reduces the number of categories of a nominal input. Nominal inputs with fewer categories in the node than the number specified in the CATBINS= option are not modified. For nominal inputs with more categories, the procedure ignores observations with the least frequent category values. Limiting the number of categories in a nominal input can strengthen the association of that input with the target by eliminating categories of less predictive potential. The procedure reduces the categories independently in every node.

The split search seeks to maximize the reduction in the Gini index for a nominal target and the reduction in variance of an interval target.

Predicting an Observation

To predict an observation, the HPFOREST procedure first assigns the observation to a single leaf in each decision tree in the forest, then uses that leaf to make a prediction based on the tree that contains the leaf, and finally simply averages the predictions over the trees. For an interval target, the prediction in a leaf equals the average of the target values among the bagged training observations in that leaf. For a nominal target, the *posterior probability* of a target category equals the proportion of that category among the bagged training observations in that leaf. The predicted nominal target category is the category with the largest posterior probability. In case of a tie, the first category that occurs in the training data is the prediction.

The HPFOREST procedure also computes *out-of-bag predictions*. The out-of-bag prediction of an observation uses only trees for which the observation is out-of-bag (that is, is not selected as part of the training data for that tree).

A model is worthless if its predictions are no better than predictions without a model. For an interval target, the *no-model* prediction of an observation is the average of the target among training observations. For a nominal target, the no-model posterior probabilities are the class proportions in the training data. The no-model predictions are the same for every observation.

Computing the Average Square Error and Misclassification Rate

The HPFOREST procedure computes the average square error and the misclassification rate to assess the goodness of fit of the model. The average square error applies to all types of targets. The misclassification rate applies only to nominal targets.

The average square errors for an interval and a nominal target, and the misclassification rate for a nominal target are defined as

$$\mathrm{ASE}_{\mathrm{int}} = \sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{N}$$

$$\mathrm{ASE}_{\mathrm{cat}} = \sum_{i=1}^{N} \sum_{j=1}^{J} \frac{(\delta_{ij} - \hat{p}_{ij})^2}{JN}$$

$$\mathrm{MISC} = \sum_{i=1}^{N} \frac{1(y_i \neq \hat{y}_i)}{N}$$

where $\hat{y}_i$ is the target prediction of observation i, δ_{ij} equals 1 or 0 if the nominal target value j does or does not occur in observation i, respectively, and $\hat{p}_{ij}$ is the predicted probability of nominal target value j for observation i. N is the number of observations, and J is the number of nominal target values (classes).

The definitions are valid whether $\hat{y}_i$ is the usual model prediction, the out-of-bag prediction, or the no-model prediction, resulting in three different estimates of ASE_{int}. The model has some predictive ability if the out-of-bag estimate of fit is smaller than the no-model estimate. The ASE_{int} based on the usual model predictions of the original training data is usually optimistic, smaller than what its value will be on future data.

Missing Values

If the value of a target variable is missing, the observation is excluded from training and evaluating the tree.

If the value of an input variable X is missing, the training procedure either uses the missing value as a special, legitimate value, or the procedure ignores the observation when computing the association of the X with the target and when searching for a splitting rule using X.

The procedure will use the missing value as a legitimate value if the MISSING= option equals USEINSEARCH and the MINUSEINSEACH= option is not larger than the number of observations with X missing in the training data in the node. Such is the default situation, if X has any missing values.

Assuming the split search uses the missing values, the search will find a rule that associates missing values with a branch that maximizes the worth of the split. For a nominal input variable, a new nominal category representing missing values is created for the duration of the split search. For an ordinal or interval input variable, a rule preserves the ordering of the nonmissing values when assigning them to branches, but may assign missing values to any single branch. Specifying MISSING=USEINSEARCH may produce a branch exclusively for missing values. This is desirable when the existence of a missing value is predictive of a target value.

If the split search does not use missing values, the resulting rule will distribute observations for which X is missing to the branches. The observation is in effect copied, one copy for each branch. The copy assigned to a branch is given a fractional frequency proportional to the number of training observations assigned to the branch.

The prediction of an observation distributed to the branches is the same as the prediction of the observation in the parent of the branches. To proof this for an interval target, let $p_{distributed}$ denote the prediction of an observation distributed to branches. By definition,

$$p_{distributed} = \sum_b (N_b/N)p_b$$

where N_b denotes the number of observations in branch b, $N = \sum_b N_b$, and p_b denotes the prediction in branch b. For an interval target, the prediction in a node is the average of the target values of the training data assigned to the node:

$$p_b = \sum_i w_{bi}y_{bi}/N_b$$

where y_{bi} denotes the target values in branch b, and w_{bi} the weight of observation bi. $\sum_i w_{bi} = N_b$. w_{bi} typically equals 1.

Substituting the formula for p_b into the previous formula,

$$\begin{aligned} p_{distributed} &= \sum_b (N_b/N)p_b \\ &= \sum_b \sum_i (N_b/N) w_{bi} y_{bi}/N_b \\ &= \sum_b \sum_i w_{bi} y_{bi}/N \end{aligned}$$

which equals the average target value in the parent node.

Specifying MISSING=DISTRIBUTE forces every splitting rule to distribute an observation to the branches when the value of the splitting variable is missing. Specifying MISSING=USEINSEARCH will also produce rules that distribute observations if the splitting variable has no missing values in the training data, or when the number of observations with missing values is less than the value specified in the MINUSEINSEACH= option.

Observations distributed into multiple branches might slow down training noticeably. Values of distributed observations in a leaf are stored in a linked list and passed to the association and split-search routines individually. Values of non-distributed observations, observations residing entirely within one leaf, are passed as a single vector. Processing a single vector of values is much faster than plodding through a linked list and calling an accumulation routine separately for each value.

The logic described in this section applies independently to each candidate variable and each node. For example, the test of association using X might use all observations, and the test using input variable Z might ignore some observations because of missing values. Or, the test using X might use all observations in one node but not all in another node.

Unseen Categorical Values

A splitting rule using a categorical variable might not recognize all possible values of the variable. Some categories might not have been in the training data. Others might have been so infrequent in the within-node training sample that the procedure excluded them. The MINCATSIZE= option specifies the minimum number of occurrences required for a categorical value to participate in the search for a splitting rule. Splitting rules handle unseen categorical values as they do missing values.

Displaying the Output

The HPFOREST procedure displays the parameters used to train the model, fit statistics of the trained model, and the time it took to train. If PROC HPFOREST is not run on a grid of computers, the evolution of fit statistics as the number of trees increases is displayed.

ODS Table Names

Table 6.2 lists the names of the data tables created by the HPFOREST procedure. Use these names in ODS statements.

Table 6.2 ODS Tables Produced by PROC HPFOREST

Table Name	Description	Required Statement / Option
PerformanceInfo	Performance Information	Default output
NObs	Number of Observations	Default output
FitStatistics	Fit statistics	Default output
ModelInfo	Model information	Default output
Timing	Absolute and relative times for tasks performed by the procedure	Default output

Examples: HPFOREST Procedure

The following examples illustrate the basic options in the HPFOREST procedure. The three examples use the spambase data available from the UCI Machine Learning Repository (Asuncion and Newman 2007) `http://archive.ics.uci.edu/ml/datasets/Spambase`.

Example 6.1: Out-Of-Bag Estimate of Misclassification Rate

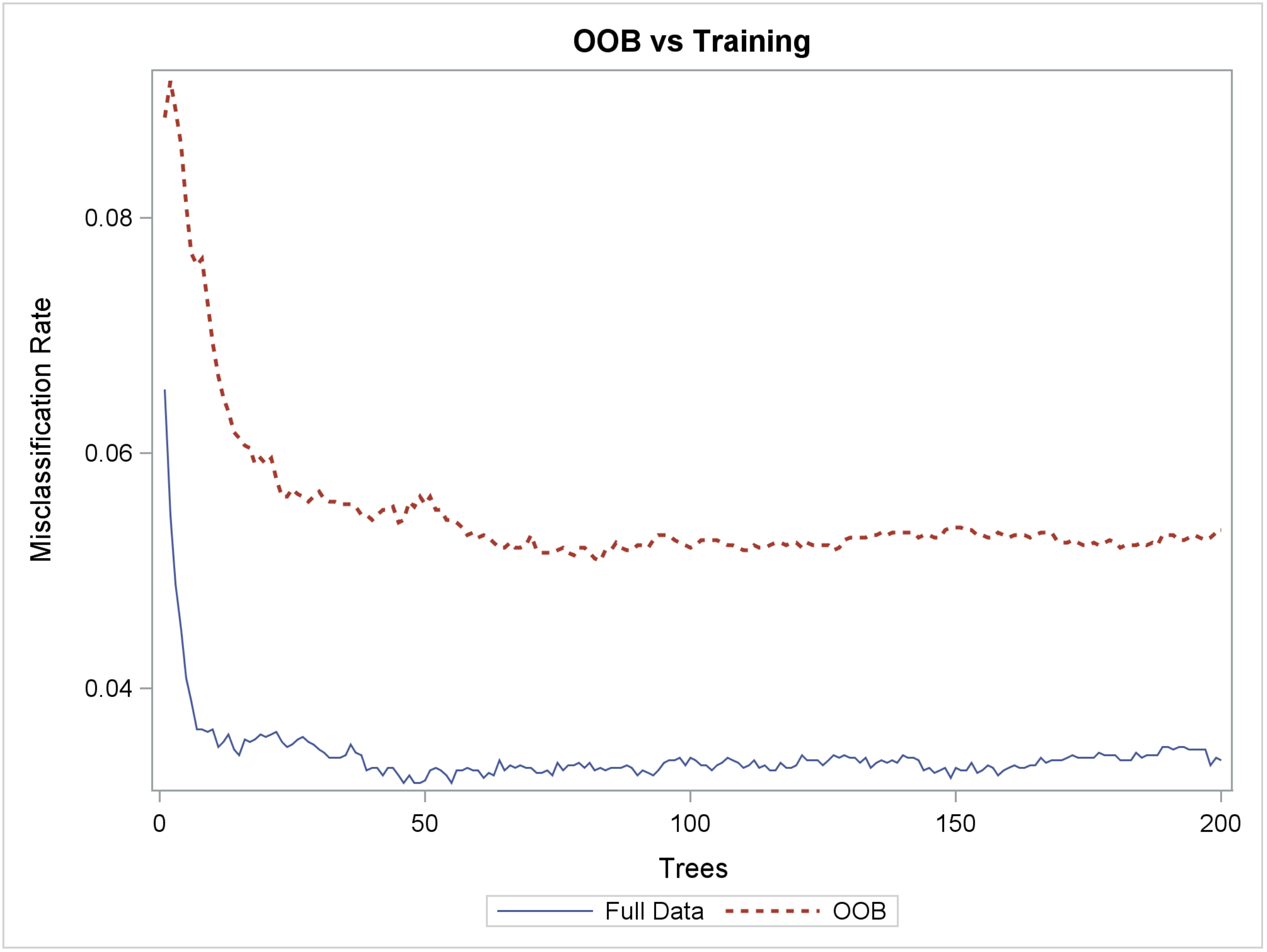
Using the original training data to evaluate a forest model is poor practice because the forest predicts the training data much better than it predicts similar data withheld from training. Using the out-of-bag data is better practice because, with enough trees, the fit of a forest to the out-of-bag data converges to what the fit would be on similar data withheld from training. With only a few trees, the fit to the out-of-bag data is worse than what the fit would be on withheld data. Consequently, the training and out-of-bag data provide lower and upper bounds to what the error rate will be when the forest is applied to new data.

This example illustrates the difference between the misclassification rates estimated from the training and out-of-bag data. The HPFOREST procedure is run on the spambase data. The target, SPAM, has two values: 0 indicates a legitimate e-mail, 1 indicates spam. A significance level of 0.2 lets the trees grow large (ALPHA=0.2), and the number of trees is set large enough for the out-of-bag misclassification error rates to converge (MAXTREES=200).

The following SAS statements create a SAS data set from the data downloaded into a file called *c:\spambase_data.txt*:

```
   data spambase;
      infile 'c:\spambase_data.txt' delimiter = ',';
      input wf_make       wf_adress     wf_all        wf_3d      wf_our
            wf_over       wf_remove     wf_internet   wf_order   wf_mail
            wf_receive    wf_will       wf_people     wf_report  wf_addresses
            wf_free       wf_business   wf_email      wf_you     wf_credit
            wf_your       wf_font       wf_000        wf_money   wf_hp
            wf_hpl        wf_george     wf_650        wf_lab     wf_labs
            wf_telnet     wf_857        wf_data       wf_415     wf_85
            wf_technology wf_1999       wf_parts      wf_pm      wf_direct
            wf_cs         wf_meeting    wf_original   wf_project wf_re
            wf_edu        wf_table      wf_conference
            cf_semicolon  cf_parenthese cf_bracket    cf_exclamation
            cf_dollar     cf_pound
            average       longest       total
            spam;
   run;

proc hpforest data=spambase alpha = 0.2 maxtrees=200;
   input w: c: average longest total/level=interval;
   target spam/level=binary;
   ods output FitStatistics=fitstats(rename=(Ntrees=Trees));
run;

data fitstats;
   set fitstats;
   label Trees = 'Number of Trees';
   label MiscAll = 'Full Data';
   label Miscoob = 'OOB';
run;
proc sgplot data=fitstats;
   title "OOB vs Training";
   series x=Trees y=MiscAll;
   series x=Trees y=MiscOob/lineattrs=(pattern=shortdash thickness=2);
   yaxis label='Misclassification Rate';
run;
title;
```

Output 6.1.1 Plot of OOB versus Training Misclassification Rate



The plot shows the misclassification rate is worse (larger) based on the out-of-bag (OOB) data, and more trees are needed for the out-of-bag rates to level off. Both characteristics are typical of a forest.

Example 6.2: Number of Variables to Try When Splitting a Node

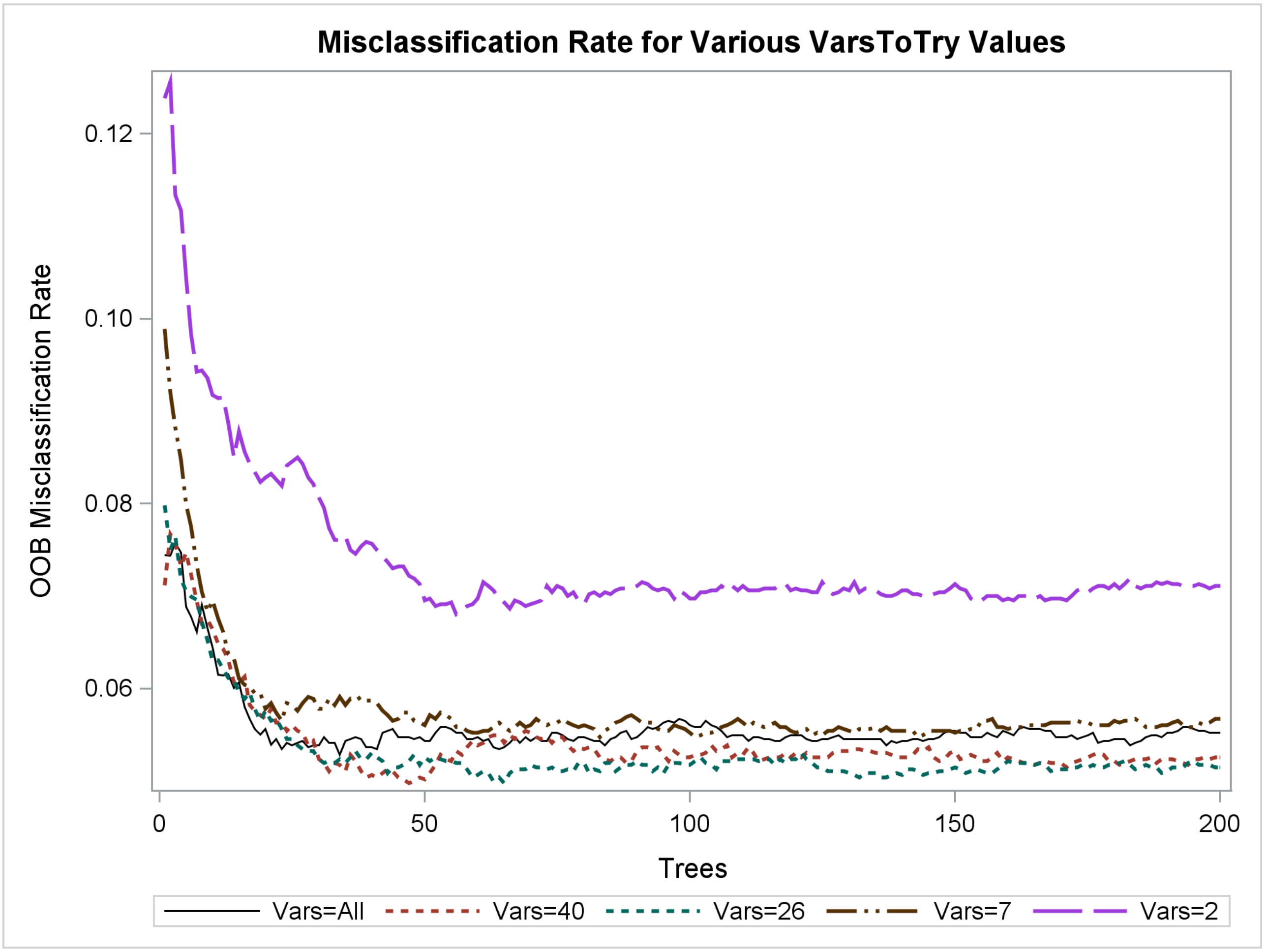
This example illustrates the effect of changing the number of variables to randomly select as candidate splitting variables in a node. In each node in each tree, *m* variables are randomly selected to be candidates to split on. Use the VARS_TO_TRY= option to specify *m*. Specifying *m* fewer than the number of available inputs is one way to reduce the correlation between the trees in the forest. Broadly speaking, the predictions of a forest improve when the trees are less correlated.

The following SAS statements create a SAS data set from the data downloaded into a file called *c:\spambase_data.txt*:

```
   data spambase;
      infile 'c:\spambase_data.txt' delimiter = ',';
      input wf_make       wf_adress     wf_all        wf_3d      wf_our
            wf_over       wf_remove     wf_internet   wf_order   wf_mail
            wf_receive    wf_will       wf_people     wf_report  wf_addresses
            wf_free       wf_business   wf_email      wf_you     wf_credit
            wf_your       wf_font       wf_000        wf_money   wf_hp
            wf_hpl        wf_george     wf_650        wf_lab     wf_labs
            wf_telnet     wf_857        wf_data       wf_415     wf_85
            wf_technology wf_1999       wf_parts      wf_pm      wf_direct
            wf_cs         wf_meeting    wf_original   wf_project wf_re
            wf_edu        wf_table      wf_conference
            cf_semicolon  cf_parenthese cf_bracket    cf_exclamation
            cf_dollar     cf_pound
            average       longest       total
            spam;
   run;


%macro hpforest(Vars=);
proc hpforest data=spambase alpha = 0.2 maxtrees=200
   vars_to_try=&Vars.;
   input w: c: average longest total/level=interval;
   target spam/level=binary;
   ods output
   FitStatistics = fitstats_vars&Vars.(rename=(Miscoob=VarsToTry&Vars.));
run;
%mend;

%hpforest(vars=all);
%hpforest(vars=40);
%hpforest(vars=26);
%hpforest(vars=7);
%hpforest(vars=2);

data fitstats;
   merge
   fitstats_varsall
   fitstats_vars40
   fitstats_vars26
   fitstats_vars7
   fitstats_vars2;
   rename Ntrees=Trees;
   label VarsToTryAll = "Vars=All";
   label VarsToTry40 = "Vars=40";
   label VarsToTry26 = "Vars=26";
   label VarsToTry7 = "Vars=7";
   label VarsToTry2 = "Vars=2";
run;

proc sgplot data=fitstats;
```

```
      title "Misclassification Rate for Various VarsToTry Values";
      series x=Trees y = VarsToTryAll/lineattrs=(Color=black);
      series x=Trees y=VarsToTry40/lineattrs=(Pattern=ShortDash Thickness=2);
      series x=Trees y=VarsToTry26/lineattrs=(Pattern=ShortDash Thickness=2);
      series x=Trees y=VarsToTry7/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
      series x=Trees y=VarsToTry2/lineattrs=(Pattern=LongDash Thickness=2);
      yaxis label='OOB Misclassification Rate';
   run;
   title;
```

Output 6.2.1 Effect of the VARS_TO_TRY= Option on the Misclassification Rate



Specifying a value of 40 or 26 for the VARS_TO_TRY= option results in a slightly more accurate forest than would occur without random selection of variables (VARS_TO_TRY=ALL). Specifying VARS_TO_TRY=7 is slightly worse, and VARS_TO_TRY=2 is much worse, than specifying VARS_TO_TRY=ALL. A good value for VARS_TO_TRY= depends on the data. In this example, the HPFOREST procedure uses a default value of $\sqrt{58} = 7$, which does not improve the error rate.

Example 6.3: Fraction of Training Data to Train a Tree

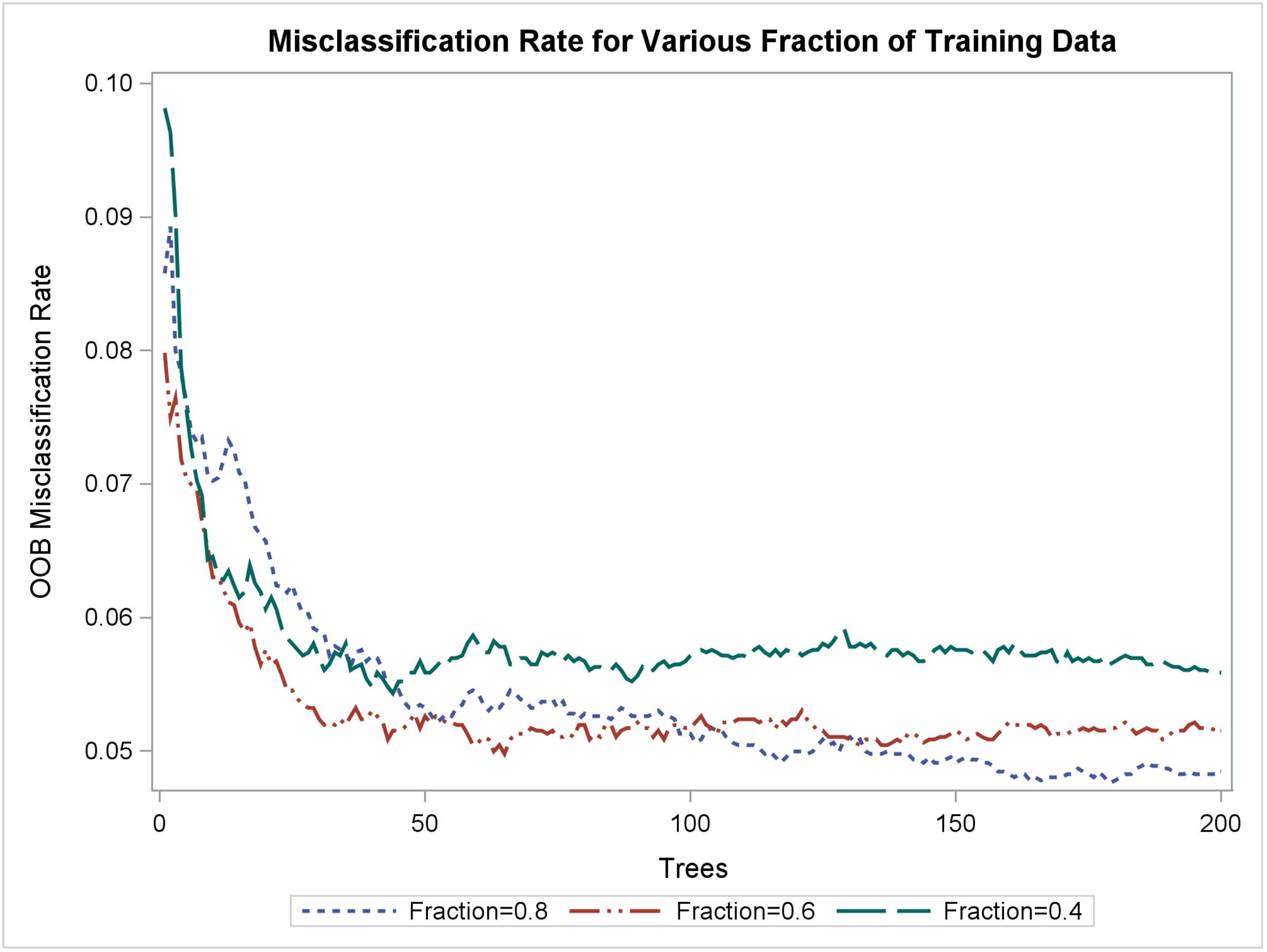
This example illustrates the effect of changing the fraction of original training observations used to train an individual tree. Use the TRAINFRACTION= option to specify f. Specifying f less than 1 is one way to reduce the correlation between the trees in the forest.

The following SAS statements create a SAS data set from the data downloaded into a file called *c:\spambase_data.txt*:

```
   data spambase;
      infile 'c:\spambase_data.txt' delimiter = ',';
      input wf_make       wf_adress     wf_all        wf_3d      wf_our
            wf_over       wf_remove     wf_internet   wf_order   wf_mail
            wf_receive    wf_will       wf_people     wf_report  wf_addresses
            wf_free       wf_business   wf_email      wf_you     wf_credit
            wf_your       wf_font       wf_000        wf_money   wf_hp
            wf_hpl        wf_george     wf_650        wf_lab     wf_labs
            wf_telnet     wf_857        wf_data       wf_415     wf_85
            wf_technology wf_1999       wf_parts      wf_pm      wf_direct
            wf_cs         wf_meeting    wf_original   wf_project wf_re
            wf_edu        wf_table      wf_conference
            cf_semicolon  cf_parenthese cf_bracket    cf_exclamation
            cf_dollar     cf_pound
            average       longest       total
            spam;
   run;


%macro hpforest(f=, output_suffix=);
proc hpforest data=spambase alpha = 0.2 maxtrees=200 vars_to_try=26
   trainfraction=&f;
   input w: c: average longest total/level=interval;
   target spam/level=binary;
   ods output
   FitStatistics = fitstats_f&output_suffix.(rename=(Miscoob=fraction&output_suffix.));
run;
%mend;

%hpforest(f=0.8, output_suffix=08);
%hpforest(f=0.6, output_suffix=06);
%hpforest(f=0.4, output_suffix=04);

data fitstats;
   merge
   fitstats_f08
   fitstats_f06
   fitstats_f04;
   rename Ntrees=Trees;
   label fraction08 = "Fraction=0.8";
   label fraction06 = "Fraction=0.6";
   label fraction04 = "Fraction=0.4";
run;
```

```
proc sgplot data=fitstats;
   title "Misclassification Rate for Various Fractions of Training Data";
   series x=Trees y=fraction08/lineattrs=(Pattern=ShortDash Thickness=2);
   series x=Trees y=fraction06/lineattrs=(Pattern=MediumDashDotDot Thickness=2);
   series x=Trees y=fraction04/lineattrs=(Pattern=LongDash Thickness=2);
   yaxis label='OOB Misclassification Rate';
run;
title;
```

Output 6.3.1 Effect of the TRAINFRACTION Option on the Misclassification Rate



In this example, a larger training fraction f results in a better OOB misclassification rate and requires more trees in the forest to obtain that rate.

References

Asuncion, A. and Newman, D. J. (2007), "UCI Machine Learning Repository," http://archive.ics.uci.edu/ml/.

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 24(2), 123–140.

Breiman, L. (2001), "Random Forests," *Machine Learning*, 45(1), 5–32.

Freedman, J. H. and Popescu, B. E. (2003), *Importance Sampled Learning Ensembles*, Technical report, Department of Statistics, Stanford University.

Hothorn, T., Hornik, K., and Zeileis, A. (2006), "Unbiased Recursive Partitioning: A Conditional Inference Framework," *Journal of Computational and Graphical Statistics*, 15, 651–674.

Loh, W. and Shih, Y. (1997), "Split Selection Methods for Classification Trees," *Statistica Sinica*, 7(4), 815–840.

Chapter 7

The HPLMIXED Procedure

Contents

Overview: HPLMIXED Procedure

The HPLMIXED procedure fits a variety of mixed linear models to data and enables you to use these fitted models to make statistical inferences about the data. A *mixed linear model* is a generalization of the standard linear model used in the GLM procedure in SAS/STAT software; the generalization is that the data are permitted to exhibit correlation and nonconstant variability. Therefore, the mixed linear model provides you with the flexibility of modeling not only the means of your data (as in the standard linear model) but also their variances and covariances.

The primary assumptions underlying the analyses performed by PROC HPLMIXED are as follows:

- The data are normally distributed (Gaussian).
- The means (expected values) of the data are linear in terms of a certain set of parameters.
- The variances and covariances of the data are in terms of a different set of parameters, and they exhibit a structure that matches one of those available in PROC HPLMIXED.

Since Gaussian data can be modeled entirely in terms of their means and variances/covariances, the two sets of parameters in a mixed linear model actually specify the complete probability distribution of the data. The parameters of the mean model are referred to as *fixed-effects parameters*, and the parameters of the variance-covariance model are referred to as *covariance parameters*.

The fixed-effects parameters are associated with known explanatory variables, as in the standard linear model. These variables can be either qualitative (as in the traditional analysis of variance) or quantitative (as in standard linear regression). However, the covariance parameters are what distinguishes the mixed linear model from the standard linear model.

The need for covariance parameters arises quite frequently in applications; the following scenarios are the most typical:

- The experimental units on which the data are measured can be grouped into clusters, and the data from a common cluster are correlated. This scenario can be generalized to include one set of clusters nested within another. For example, if students are the experimental unit, they can be clustered into classes, which in turn can be clustered into schools. Each level of this hierarchy can introduce an additional source of variability and correlation.

- Repeated measurements are taken on the same experimental unit, and these repeated measurements are correlated or exhibit variability that changes. This scenario occurs in longitudinal studies, where repeated measurements are taken over time. Alternatively, the repeated measures could be spatial or multivariate in nature.

PROC HPLMIXED provides a variety of covariance structures to handle these two scenarios. The most common covariance structures arise from the use of *random-effects parameters*, which are additional unknown random variables that are assumed to affect the variability of the data. The variances of the random-effects parameters, commonly known as *variance components*, become the covariance parameters for this particular structure. Traditional mixed linear models contain both fixed- and random-effects parameters; in fact, it is the combination of these two types of effects that led to the name *mixed model*. PROC HPLMIXED fits not only these traditional variance component models but also numerous other covariance structures.

PROC HPLMIXED fits the structure you select to the data by using the method of *restricted maximum likelihood (REML)*, also known as *residual maximum likelihood*. It is here that the Gaussian assumption for the data is exploited.

PROC HPLMIXED Features

PROC HPLMIXED provides easy accessibility to numerous mixed linear models that are useful in many common statistical analyses.

Here are some basic features of PROC HPLMIXED:

- covariance structures, including variance components, compound symmetry, unstructured, AR(1), Toeplitz, and factor analytic
- MODEL, RANDOM, and REPEATED statements for model specification as in the MIXED procedure
- appropriate standard errors, *t* tests, and *F* tests for all specified estimable linear combinations of fixed and random effects
- a subject effect that enables blocking
- REML and ML (maximum likelihood) estimation methods implemented with a variety of optimization algorithms
- capacity to handle unbalanced data
- special dense and sparse algorithms that take advantage of distributed and multicore computing environments

PROC HPLMIXED uses the Output Delivery System (ODS), a SAS subsystem that provides capabilities for displaying and controlling the output from SAS procedures. ODS enables you to convert any output from PROC HPLMIXED into a SAS data set. See the section “ODS Table Names” on page 169.

Notation for the Mixed Model

This section introduces the mathematical notation used throughout this chapter to describe the mixed linear model and assumes familiarity with basic matrix algebra (for an overview, see Searle 1982). A more detailed description of the mixed model is contained in the section "Linear Mixed Models Theory" on page 157.

A statistical model is a mathematical description of how data are generated. The standard linear model, as used by the GLM procedure, is one of the most common statistical models:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

In this expression, $\mathbf{y}$ represents a vector of observed data, $\boldsymbol{\beta}$ is an unknown vector of fixed-effects parameters with a known design matrix $\mathbf{X}$, and $\boldsymbol{\epsilon}$ is an unknown random error vector that models the statistical noise around $\mathbf{X}\boldsymbol{\beta}$. The focus of the standard linear model is to model the mean of $\mathbf{y}$ by using the fixed-effects parameters $\boldsymbol{\beta}$. The residual errors $\boldsymbol{\epsilon}$ are assumed to be independent and identically distributed Gaussian random variables with mean 0 and variance σ^2.

The mixed model generalizes the standard linear model as follows:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\boldsymbol{\gamma} + \boldsymbol{\epsilon}$$

Here, $\boldsymbol{\gamma}$ is an unknown vector of random-effects parameters with a known design matrix $\mathbf{Z}$, and $\boldsymbol{\epsilon}$ is an unknown random error vector whose elements are no longer required to be independent and homogeneous.

To further develop this notion of variance modeling, assume that $\boldsymbol{\gamma}$ and $\boldsymbol{\epsilon}$ are Gaussian random variables that are uncorrelated, have expectations $\mathbf{0}$, and have variances $\mathbf{G}$ and $\mathbf{R}$, respectively. The variance of $\mathbf{y}$ is thus

$$\mathbf{V} = \mathbf{ZGZ}' + \mathbf{R}$$

Note that when $\mathbf{R} = \sigma^2\mathbf{I}$ and $\mathbf{Z} = \mathbf{0}$, the mixed model reduces to the standard linear model.

You can model the variance of the data $\mathbf{y}$ by specifying the structure of $\mathbf{Z}$, $\mathbf{G}$, and $\mathbf{R}$. The model matrix $\mathbf{Z}$ is set up in the same fashion as $\mathbf{X}$, the model matrix for the fixed-effects parameters. For $\mathbf{G}$ and $\mathbf{R}$, you must select some covariance structure. Possible covariance structures include the following:

- variance components
- compound symmetry (common covariance plus diagonal)
- unstructured (general covariance)
- autoregressive
- spatial
- general linear
- factor analytic

By appropriately defining the model matrices $\mathbf{X}$ and $\mathbf{Z}$ in addition to the covariance structure matrices $\mathbf{G}$ and $\mathbf{R}$, you can perform numerous mixed model analyses.

PROC HPLMIXED Contrasted with Other SAS Procedures

The RANDOM and REPEATED statements of the HPLMIXED procedure follow the convention of the same statements in the MIXED procedure in SAS/STAT software. For information about how these statements differ from RANDOM and REPEATED statements in the MIXED procedure, see the documentation for the MIXED procedure in the *SAS/STAT User's Guide*.

The GLIMMIX procedure in SAS/STAT software fits generalized linear mixed models. Linear mixed models—where the data are normally distributed, given the random effects—are in the class of generalized linear mixed models. Therefore, PROC GLIMMIX accommodates nonnormal data with random effects.

Generalized linear mixed models have intrinsically nonlinear features because a nonlinear mapping (the link function) connects the conditional mean of the data (given the random effects) to the explanatory variables. The NLMIXED procedure also accommodates nonlinear structures in the conditional mean, but places no restrictions on the nature of the nonlinearity.

The HPMIXED procedure in SAS/STAT software is also termed a "high-performance" procedure, but it does not follow the general pattern of High-Performance Analytics procedures. The HPMIXED procedure does not take advantage of distributed or multicore computing environments; it derives high-performance from applying sparse techniques to solving the mixed model equations. The HPMIXED procedure fits a small subset of the statistical models you can fit with the MIXED or HPLMIXED procedures and is particularly suited for problems in which the $[\mathbf{XZ}]'[\mathbf{XZ}]$ crossproducts matrix is sparse.

The HPLMIXED procedure employs algorithms that are specialized for distributed and multicore computing environments. The HPLMIXED procedure does not support BY processing.

Getting Started: HPLMIXED Procedure

Mixed Model Analysis of Covariance with Many Groups

Suppose you are an educational researcher who studies how student scores on math tests change over time. Students are tested four times, and you want to estimate the overall rise or fall, accounting for correlation between test response behaviors of students in the same neighborhood and school. One way to model this correlation is by using a random-effects analysis of covariance, where the scores for students from the same neighborhood and school are all assumed to share the same quadratic mean test response function, the parameters of this response function being random. The following statements simulate a data set with this structure:

```
data SchoolSample;
   do SchoolID = 1 to 300;
      do nID = 1 to 25;
         Neighborhood = (SchoolID-1)*5 + nId;
         bInt   = 5*ranuni(1);
```

```
            bTime  = 5*ranuni(1);
            bTime2 =   ranuni(1);
            do sID = 1 to 2;
               do Time = 1 to 4;
                  Math = bInt + bTime*Time + bTime2*Time*Time + rannor(2);
                  output;
                  end;
               end;
            end;
         end;
   run;
```

In this data, there are 300 schools and about 1,500 neighborhoods; neighborhoods are associated with more than one school and vice versa. The following statements use PROC HPLMIXED to fit a mixed analysis of covariance model to this data:

```
proc hplmixed data=SchoolSample;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC" nodes=20;
   class Neighborhood SchoolID;
   model Math = Time Time*Time / solution;
   random   int Time Time*Time / sub=Neighborhood(SchoolID) type=un;
run;
```

This model fits a quadratic mean response model with an unstructured covariance matrix to model the covariance between the random parameters of the response model. With 7,500 neighborhood/school combinations, this model can be computationally daunting to fit, but PROC HPLMIXED finishes quickly and displays the results shown in Figure 7.1.

Figure 7.1 Mixed Model Analysis of Covariance

```
                          The HPLMIXED Procedure

                         Performance Information

              Host Node                     rdgrd0001.unx.sas.com
              Execution Mode                Distributed
              Number of Compute Nodes       20
              Number of Threads per Node    8

                            Model Information

           Data Set                      WORK.SCHOOLSAMPLE
           Dependent Variable            Math
           Covariance Structure          Unstructured
           Subject Effect                Neighborho(SchoolID)
           Estimation Method             Restricted Maximum Likelihood
           Residual Variance Method      Profile
           Fixed Effects SE Method       Model-Based
           Degrees of Freedom Method     Residual
```

Figure 7.1 *continued*

```
                          Class Level Information

    Class            Levels    Values

    Neighborhood       1520    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                               19 20 ...
    SchoolID            300    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                               19 20 ...

                                 Dimensions

                Covariance Parameters                  7
                Columns in X                           3
                Columns in Z Per Subject               3
                Subjects                            7500
                Max Obs Per Subject                    8

            Number of Observations Read               60000
            Number of Observations Used               60000
            Number of Observations Not Used               0
            Number of Observations Swapped            52500
            Number of Subjects Needing Swap            7500

                         Optimization Information

       Optimization Technique        Newton-Raphson with Ridging
       Parameters in Optimization    6
       Lower Boundaries              3
       Upper Boundaries              0
       Starting Values From          Data

                              Iteration History

                                  Objective                           Max
      Iteration    Evaluations     Function          Change     Gradient

              0              2    225641.67142          .         2.135E-8

          Convergence criterion (ABSGCONV=0.00001) satisfied.

                       Covariance Parameter Estimates

              Cov Parm     Subject                 Estimate

              UN(1,1)      Neighborho(SchoolID)      2.0902
              UN(2,1)      Neighborho(SchoolID)    0.000349
              UN(2,2)      Neighborho(SchoolID)      2.0517
              UN(3,1)      Neighborho(SchoolID)     0.01448
              UN(3,2)      Neighborho(SchoolID)     0.01599
              UN(3,3)      Neighborho(SchoolID)     0.08047
              Residual                               1.0083
```

Figure 7.1 *continued*

```
                              Fit Statistics

                 -2 Res Log Likelihood              225642
                 AIC (smaller is better)            225656
                 AICC (smaller is better)           225656
                 BIC (smaller is better)            225704

                        Solution for Fixed Effects

                                    Standard
         Effect       Estimate         Error       DF    t Value    Pr > |t|

         Intercept      2.5070       0.02828      6E4      88.66      <.0001
         Time           2.5124       0.02659      6E4      94.48      <.0001
         Time*Time      0.5010      0.005247      6E4      95.48      <.0001
```

Syntax: HPLMIXED Procedure

The following statements are available in PROC HPLMIXED.

PROC HPLMIXED < *options* > ;
  CLASS *variables* ;
  MODEL *dependent* **=** < *fixed-effects* > < / *options* > ;
  RANDOM *random-effects* < / *options* > ;
  REPEATED *repeated-effect* < / *options* > ;
  PARMS < **(***value-list***)** . . . > < / *options* > ;
  PERFORMANCE < *options* > ;

Items within angle brackets (< >) are optional. The RANDOM statement can appear multiple times. Other statements can appear only once.

The PROC HPLMIXED and MODEL statements are required, and the MODEL statement must appear after the CLASS statement if a CLASS statement is included. The RANDOM statement must follow the MODEL statement.

Table 7.1 summarizes the basic functions and important options of the PROC HPLMIXED statements. The syntax of each statement in Table 7.1 is described in the following sections in alphabetical order after the description of the PROC HPLMIXED statement.

Table 7.1 Summary of PROC HPLMIXED Statements

Statement	Description	Important Options
PROC HPLMIXED	Invokes the procedure	DATA= specifies the input data set; METHOD= specifies the estimation method.

Table 7.1 *continued*

Statement	Description	Important Options
CLASS	Declares qualitative variables that create indicator variables in **X** and **Z** matrices.	None
MODEL	Specifies dependent variable and fixed effects, setting up **X**	S requests a solution for fixed-effects parameters.
RANDOM	Specifies random effects, setting up **Z** and **G**	SUBJECT= creates block-diagonality; TYPE= specifies the covariance structure; S requests a solution for the random effects.
REPEATED	Sets up **R**	SUBJECT= creates block-diagonality; TYPE= specifies the covariance structure.
PARMS	Specifies a grid of initial values for the covariance parameters	HOLD= and NOITER hold the covariance parameters or their ratios constant; PARMSDATA= reads the initial values from a SAS data set.
PERFORMANCE	Invokes the distributed computing connection	NODES= specifies the number of nodes to use.

PROC HPLMIXED Statement

PROC HPLMIXED < *options* > **;**

The PROC HPLMIXED statement invokes the procedure. Table 7.2 summarizes important options in the PROC HPLMIXED statement by function. These and other options in the PROC HPLMIXED statement are then described fully in alphabetical order.

Table 7.2 PROC HPLMIXED Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
METHOD=	Specifies the estimation method
NAMELEN=	Limits the length of effect names
BLUP	Computes the best linear unbiased prediction
Options Related to Output	
NOCLPRINT	Suppresses the “Class Level Information” table completely or in parts
MAXCLPRINT=	Specifies the maximum levels of CLASS variables to print

Table 7.2 *continued*

Option	Description
Optimization Options	
ABSCONV=	Tunes an absolute function convergence criterion
ABSFCONV=	Tunes an absolute function difference convergence criterion
ABSGCONV=	Tunes the absolute gradient convergence criterion
FCONV=	Tunes the relative function convergence criterion
GCONV=	Tunes the relative gradient convergence criterion
MAXITER=	Chooses the maximum number of iterations in any optimization
MAXFUNC=	Specifies the maximum number of function evaluations in any optimization
MAXTIME=	Specifies the upper limit on seconds of CPU time for any optimization
MINITER=	Specifies the minimum number of iterations in any optimization
TECHNIQUE=	Selects the optimization technique

You can specify the following *options* in the PROC HPLMIXED statement.

ABSCONV=*r*

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

ABSFCONV=*r*

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

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

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

ABSGCONV=*r*

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

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

Here, $\boldsymbol{\psi}$ denotes the vector of parameters that participate in the optimization and $g_j(\cdot)$ is the gradient of the objective function with respect to the j parameter. This criterion is not used by the NMSIMP technique. The default value is *r*=1E–5.

BLUP< *(suboptions)* >

requests the best linear unbiased prediction (BLUP) computation. The covariance parameters are assumed to be known and given by PARMS statement. This option is designed for users who need BLUP solutions for random effects with many levels, up to tens of millions.

You can specify the following *suboptions*:

ITPRINT=*number* specifies that the iteration history be displayed after every *number* of iterations. The default value is 10, which means the procedure displays the iteration history for every 10 iterations.

MAXITER=*number* specifies the maximum number of iterations allowed. The default value is the number of parameters in the BLUP option plus 2.

TOL=*number* specifies the tolerance value. The default value is the square root of machine precision.

DATA=*SAS-data-set*

names the SAS data set to be used as the input data set. The default is the most recently created data set.

FCONV=*r*

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

$$\frac{|f(\boldsymbol{\psi}^{(k)}) - f(\boldsymbol{\psi}^{(k-1)})|}{|f(\boldsymbol{\psi}^{(k-1)})|} \le r$$

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

The default is $r = 10^{-\text{FDIGITS}}$, where FDIGITS is $-\log_{10}(\epsilon)$ and ϵ is the machine precision.

GCONV=*r*

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

$$\frac{\mathbf{g}(\boldsymbol{\psi}^{(k)})'[\mathbf{H}^{(k)}]^{-1}\mathbf{g}(\boldsymbol{\psi}^{(k)})}{|f(\boldsymbol{\psi}^{(k)})|} \le r$$

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

$$\frac{\parallel \mathbf{g}(\boldsymbol{\psi}^{(k)}) \parallel_2^2 \quad \parallel \mathbf{s}(\boldsymbol{\psi}^{(k)}) \parallel_2}{\parallel \mathbf{g}(\boldsymbol{\psi}^{(k)}) - \mathbf{g}(\boldsymbol{\psi}^{(k-1)}) \parallel_2 |f(\boldsymbol{\psi}^{(k)})|} \le r$$

This criterion is not used by the NMSIMP technique. The default value is r=1E–8.

MAXCLPRINT=*number*

specifies the maximum levels of CLASS variables to print in the ODS table "ClassLevels." The default value is 20. MAXCLPRINT=0 enables you to print all levels of each CLASS variable. However, the option NOCLPRINT takes precedence over MAXCLPRINT.

MAXFUNC=*n*

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

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

The optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed *n*. You can choose the optimization technique with the TECHNIQUE= option.

MAXITER=*n*

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

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

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

MAXTIME=*r*

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

METHOD=REML

METHOD=ML

specifies the estimation method for the covariance parameters. METHOD=REML performs residual (restricted) maximum likelihood; it is the default method. METHOD=ML performs maximum likelihood.

MINITER=*n*

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

NAMELEN=*number*

specifies the length to which long effect names are shortened. The minimum value is 20, which is also the default.

NOCLPRINT< =*number* **>**

suppresses the display of the “Class Level Information” table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps to reduce the size of the “Class Level Information” table if some classification variables have a large number of levels.

NOPRINT

suppresses the generation of ODS output.

SINGCHOL=*number*

tunes the singularity criterion in Cholesky decompositions. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

SINGSWEEP=*number*

tunes the singularity criterion for sweep operations. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

SINGULAR=*number*

tunes the general singularity criterion applied by the HPLMIXED procedure in sweeps and inversions. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

TECHNIQUE=*keyword*

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

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization combining a line-search algorithm with ridging.
NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	performs no optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust-region optimization.

The default value is TECHNIQUE=NRRIDG.

CLASS Statement

CLASS *variables* **;**

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. See the section “Levelization of Classification Variables” on page 33 of Chapter 2, “Shared Concepts and Topics” for details about these mappings.

If a CLASS statement is specified, it must precede the MODEL statement in High-Performance Analytics procedures that support a MODEL statement.

Levels of classification variables are ordered by their external formatted values, except for numeric variables with no explicit format, which are ordered by their unformatted (internal) values.

MODEL Statement

MODEL *dependent* **=** < *fixed-effects* >< / *options* > **;**

The MODEL statement names a single dependent variable and the fixed effects, which determine the $\mathbf{X}$ matrix of the mixed model. (For details, see the section “Specification and Parameterization of Model Effects” on page 35 of Chapter 2, “Shared Concepts and Topics”.) The MODEL statement is required.

An intercept is included in the fixed-effects model by default. If no fixed effects are specified, only this intercept term is fit. The intercept can be removed by using the NOINT option.

Table 7.3 summarizes options in the MODEL statement. These are subsequently discussed in detail in alphabetical order.

Table 7.3 Summary of Important MODEL Statement Options

Option	Description
Model Building	
NOINT	Excludes the fixed-effect intercept from model
Statistical Computations	
ALPHA=α	Determines the confidence level $(1-\alpha)$ for fixed effects
DDFM=	Specifies the method for computing denominator degrees of freedom
Statistical Output	
CL	Displays confidence limits for fixed-effects parameter estimates
SOLUTION	Displays fixed-effects parameter estimates

You can specify the following *options* in the MODEL statement after a slash (/).

ALPHA=*number*

sets the confidence level to be 1−*number* for each confidence interval of the fixed-effects parameters. The value of *number* must be between 0 and 1; the default is 0.05.

CL

requests that t-type confidence limits be constructed for each of the fixed-effects parameter estimates. The confidence level is 0.95 by default; this can be changed with the ALPHA= option.

DDFM=NONE | RESIDUAL

specifies the method for computing the denominator degrees of freedom for the tests of fixed effects.

The DDFM=RESIDUAL option performs all tests by using the residual degrees of freedom, $n - \text{rank}(\mathbf{X})$, where n is the number of observations used. It is the default degrees-of-freedom method.

DDFM=NONE specifies that no denominator degrees of freedom be applied. PROC HPLMIXED then essentially assumes that infinite degrees of freedom are available in the calculation of p-values. The p-values for t tests are then identical to p-values that are derived from the standard normal distribution. In the case of F tests, the p-values equal those of chi-square tests determined as follows: if F_{obs} is the observed value of the F test with l numerator degrees of freedom, then

$$p = \Pr\{F_{l,\infty} > F_{obs}\} = \Pr\{\chi^2_l > lF_{obs}\}$$

NOINT

requests that no intercept be included in the model. (An intercept is included by default.)

SOLUTION

S

requests that a solution for the fixed-effects parameters be produced. Using notation from the section “Linear Mixed Models Theory” on page 157, the fixed-effects parameter estimates are $\widehat{\boldsymbol{\beta}}$ and their approximate standard errors are the square roots of the diagonal elements of $(\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-}$.

Along with the estimates and their approximate standard errors, a t statistic is computed as the estimate divided by its standard error. The Pr > |t| column contains the two-tailed p-value that corresponds to the t statistic and associated degrees of freedom. You can use the CL option to request confidence intervals for all of the parameters; they are constructed around the estimate by using a radius that is the product of the standard error times a percentage point from the t distribution.

PARMS Statement

PARMS < *(value-list)*. . . > < / *options* > **;**

The PARMS statement specifies initial values for the covariance parameters, or it requests a grid search over several values of these parameters. You must specify the values in the order in which they appear in the “Covariance Parameter Estimates” table.

The *value-list* specification can take any of several forms:

m	a single value
$m_1, m_2, \ldots, m_n$	several values
m to *n*	a sequence in which *m* equals the starting value, *n* equals the ending value, and the increment equals 1
m to *n* by *i*	a sequence in which *m* equals the starting value, *n* equals the ending value, and the increment equals *i*
m_1, m_2 to m_3	mixed values and sequences

You can use the PARMS statement to input known parameters.

If you specify more than one set of initial values, PROC HPLMIXED performs a grid search of the likelihood surface and uses the best point on the grid for subsequent analysis. Specifying a large number of grid points can result in long computing times.

The results from the PARMS statement are the values of the parameters on the specified grid (denoted by CovP1 through CovP*n*), the residual variance (possibly estimated) for models with a residual variance parameter, and various functions of the likelihood.

You can specify the following *options* in the PARMS statement after a slash (/).

HOLD=*all*

EQCONS=*all*

specifies that all parameter values be held to equal the specified values.

For example, the following statement constrains all covariance parameters to equal 5, 3, 2, and 3:

```
parms (5) (3) (2) (3) / hold=all;
```

LOWERB=*value-list*

enables you to specify lower boundary constraints on the covariance parameters. The *value-list* specification is a list of numbers or missing values (.) separated by commas. You must list the numbers in the order that PROC HPLMIXED uses for the covariance parameters, and each number corresponds to the lower boundary constraint. A missing value instructs PROC HPLMIXED to use its default constraint. If you do not specify numbers for all of the covariance parameters, PROC HPLMIXED assumes the remaining ones are missing.

This option is useful when you want to constrain the **G** matrix to be positive definite in order to avoid the more computationally intensive algorithms that would be required when **G** becomes singular. The corresponding statements for a random coefficients model are as follows:

```
proc hplmixed;
   class person;
   model y = time;
   random int time / type=fa0(2) sub=person;
   parms / lowerb=1e-4,.,1e-4;
run;
```

The TYPE=FA0(2) structure specifies a Cholesky root parameterization for the 2×2 unstructured blocks in **G**. This parameterization ensures that the **G** matrix is nonnegative definite, and the PARMS statement then ensures that it is positive definite by constraining the two diagonal terms to be greater than or equal to 1E–4.

NOITER

requests that no optimization iterations be performed and that PROC HPLMIXED use the best value from the grid search to perform inferences. By default, iterations begin at the best value from the PARMS grid search.

PARMSDATA=*SAS-data-set*

PDATA=*SAS-data-set*

reads in covariance parameter values from a SAS data set. The data set should contain the Est or Covp1 through Covp*n* variables.

UPPERB=*value-list*

enables you to specify upper boundary constraints on the covariance parameters. The *value-list* specification is a list of numbers or missing values (.) separated by commas. You must list the numbers in the order that PROC HPLMIXED uses for the covariance parameters, and each number corresponds to the upper boundary constraint. A missing value instructs PROC HPLMIXED to use its default constraint. If you do not specify numbers for all of the covariance parameters, PROC HPLMIXED assumes that the remaining ones are missing.

PERFORMANCE Statement

PERFORMANCE < *performance-options*> **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

With the PERFORMANCE statement you can also control whether a SAS High-Performance Analytics procedure executes in SMP or MPP mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

RANDOM Statement

RANDOM *random-effects* < / *options*> **;**

The RANDOM statement defines the random effects that constitute the γ vector in the mixed model. You can use this statement to specify traditional variance component models and to specify random coefficients. The random effects can be classification or continuous, and multiple RANDOM statements are possible.

Using notation from the section "Linear Mixed Models Theory" on page 157, the purpose of the RANDOM statement is to define the $\mathbf{Z}$ matrix of the mixed model, the random effects in the $\boldsymbol{\gamma}$ vector, and the structure of $\mathbf{G}$. The $\mathbf{Z}$ matrix is constructed exactly as the $\mathbf{X}$ matrix for the fixed effects is constructed, and the $\mathbf{G}$ matrix is constructed to correspond with the effects that constitute $\mathbf{Z}$. The structure of $\mathbf{G}$ is defined by using the TYPE= option.

You can specify INTERCEPT (or INT) as a random effect to indicate the intercept. PROC HPLMIXED does not include the intercept in the RANDOM statement by default as it does in the MODEL statement.

Table 7.4 summarizes important options in the RANDOM statement. All options are subsequently discussed in alphabetical order.

Table 7.4 Summary of Important RANDOM Statement Options

Option	Description
Construction of Covariance Structure	
SUBJECT=	Identifies the subjects in the model
TYPE=	Specifies the covariance structure
Statistical Output	
ALPHA=α	Determines the confidence level $(1-\alpha)$
CL	Requests confidence limits for predictors of random effects
SOLUTION	Displays solutions $\widehat{\boldsymbol{\gamma}}$ of the random effects

You can specify the following *options* in the RANDOM statement after a slash (/).

ALPHA=*number*

sets the confidence level to be $1-$*number* for each confidence interval of the random-effects estimates. The value of *number* must be between 0 and 1; the default is 0.05.

CL

requests that t-type confidence limits be constructed for each of the random-effect estimates. The confidence level is 0.95 by default; this can be changed with the ALPHA= option.

SOLUTION

S

requests that the solution for the random-effects parameters be produced. Using notation from the section "Linear Mixed Models Theory" on page 157, these estimates are the empirical best linear unbiased predictors (EBLUPs), $\widehat{\boldsymbol{\gamma}} = \widehat{\mathbf{G}}\mathbf{Z}'\widehat{\mathbf{V}}^{-1}(\mathbf{y}-\mathbf{X}\widehat{\boldsymbol{\beta}})$. They can be useful for comparing the random effects from different experimental units and can also be treated as residuals in performing diagnostics for your mixed model.

The numbers displayed in the SE Pred column of the "Solution for Random Effects" table are not the standard errors of the $\widehat{\boldsymbol{\gamma}}$ displayed in the Estimate column; rather, they are the standard errors of predictions $\widehat{\gamma}_i - \gamma_i$, where $\widehat{\gamma}_i$ is the ith EBLUP and γ_i is the ith random-effect parameter.

SUBJECT=*effect*

SUB=*effect*

identifies the subjects in your mixed model. Complete independence is assumed across subjects; thus, for the RANDOM statement, the SUBJECT= option produces a block-diagonal structure in **G** with identical blocks. In fact, specifying a subject effect is equivalent to nesting all other effects in the RANDOM statement within the subject effect.

When you specify the SUBJECT= option and a classification random effect, computations are usually much quicker if the levels of the random effect are duplicated within each level of the SUBJECT= effect.

TYPE=*covariance-structure*

specifies the covariance structure of **G**. Valid values for *covariance-structure* and their descriptions are listed in Table 7.5. Although a variety of structures are available, most applications call for either TYPE=VC or TYPE=UN. The TYPE=VC (variance components) option is the default structure, and it models a different variance component for each random effect.

The TYPE=UN (unstructured) option is useful for correlated random coefficient models. For example, the following statement specifies a random intercept-slope model that has different variances for the intercept and slope and a covariance between them:

```
   random intercept age / type=un subject=person;
```

You can also use TYPE=FA0(2) here to request a **G** estimate that is constrained to be nonnegative definite.

If you are constructing your own columns of **Z** with continuous variables, you can use the TYPE=TOEP(1) structure to group them together to have a common variance component. If you want to have different covariance structures in different parts of **G**, you must use multiple RANDOM statements with different TYPE= options.

Table 7.5 Covariance Structures

Structure	Description	Parms	(i,j) element
ANTE(1)	Antedependence	$2t-1$	$\sigma_i\sigma_j\prod_{k=i}^{j-1}\rho_k$
AR(1)	Autoregressive(1)	2	$\sigma^2\rho^{\|i-j\|}$
ARH(1)	Heterogeneous AR(1)	$t+1$	$\sigma_i\sigma_j\rho^{\|i-j\|}$
ARMA(1,1)	Autoregressive moving average(1,1)	3	$\sigma^2[\gamma\rho^{\|i-j\|-1}1(i\neq j)+1(i=j)]$
CS	Compound symmetry	2	$\sigma_1+\sigma^2 1(i=j)$
CSH	Heterogeneous compound symmetry	$t+1$	$\sigma_i\sigma_j[\rho 1(i\neq j)+1(i=j)]$
FA(q)	Factor analytic	$\frac{q}{2}(2t-q+1)+t$	$\Sigma_{k=1}^{\min(i,j,q)}\lambda_{ik}\lambda_{jk}+\sigma_i^2 1(i=j)$
FA0(q)	No diagonal FA	$\frac{q}{2}(2t-q+1)$	$\Sigma_{k=1}^{\min(i,j,q)}\lambda_{ik}\lambda_{jk}$
FA1(q)	Equal diagonal FA	$\frac{q}{2}(2t-q+1)+1$	$\Sigma_{k=1}^{\min(i,j,q)}\lambda_{ik}\lambda_{jk}+\sigma^2 1(i=j)$
HF	Huynh-Feldt	$t+1$	$(\sigma_i^2+\sigma_j^2)/2+\lambda 1(i\neq j)$
SIMPLE	An alias for VC	q	$\sigma_k^2 1(i=j)$ for the kth effect
TOEP	Toeplitz	t	$\sigma_{\|i-j\|+1}$
TOEP(q)	Banded Toeplitz	q	$\sigma_{\|i-j\|+1}1(\|i-j\|<q)$
TOEPH	Heterogeneous TOEP	$2t-1$	$\sigma_i\sigma_j\rho_{\|i-j\|}$

Table 7.5 *continued*

Structure	Description	Parms	(i,j) element
TOEPH(q)	Banded heterogeneous TOEP	$t+q-1$	$\sigma_i \sigma_j \rho_{\lvert i-j \rvert} 1(\lvert i-j \rvert < q)$
UN	Unstructured	$t(t+1)/2$	σ_{ij}
UN(q)	Banded	$\frac{q}{2}(2t-q+1)$	$\sigma_{ij} 1(\lvert i-j \rvert < q)$
UNR	Unstructured correlation	$t(t+1)/2$	$\sigma_i \sigma_j \rho_{\max(i,j)\min(i,j)}$
UNR(q)	Banded correlations	$\frac{q}{2}(2t-q+1)$	$\sigma_i \sigma_j \rho_{\max(i,j)\min(i,j)}$
VC	Variance components	q	$\sigma_k^2 1(i=j)$ for the kth effect

In Table 7.5, the Parms column represents the number of covariance parameters in the structure, t is the overall dimension of the covariance matrix, and $1(A)$ equals 1 when A is true and 0 otherwise. For example, $1(i=j)$ equals 1 when $i=j$ and 0 otherwise, and $1(\lvert i-j \rvert < q)$ equals 1 when $\lvert i-j \rvert < q$ and 0 otherwise. For the TYPE=TOEPH structures, $\rho_0 = 1$; for the TYPE=UNR structures, $\rho_{ii} = 1$ for all i.

Table 7.6 lists some examples of the structures in Table 7.5.

Table 7.6 Covariance Structure Examples

Description	Structure	Example
Variance components	VC (default)	$\begin{bmatrix} \sigma_B^2 & 0 & 0 & 0 \\ 0 & \sigma_B^2 & 0 & 0 \\ 0 & 0 & \sigma_{AB}^2 & 0 \\ 0 & 0 & 0 & \sigma_{AB}^2 \end{bmatrix}$
Compound symmetry	CS	$\begin{bmatrix} \sigma^2+\sigma_1 & \sigma_1 & \sigma_1 & \sigma_1 \\ \sigma_1 & \sigma^2+\sigma_1 & \sigma_1 & \sigma_1 \\ \sigma_1 & \sigma_1 & \sigma^2+\sigma_1 & \sigma_1 \\ \sigma_1 & \sigma_1 & \sigma_1 & \sigma^2+\sigma_1 \end{bmatrix}$
Unstructured	UN	$\begin{bmatrix} \sigma_1^2 & \sigma_{21} & \sigma_{31} & \sigma_{41} \\ \sigma_{21} & \sigma_2^2 & \sigma_{32} & \sigma_{42} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 & \sigma_{43} \\ \sigma_{41} & \sigma_{42} & \sigma_{43} & \sigma_4^2 \end{bmatrix}$
Banded main diagonal	UN(1)	$\begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 \\ 0 & 0 & 0 & \sigma_4^2 \end{bmatrix}$
First-order autoregressive	AR(1)	$\sigma^2 \begin{bmatrix} 1 & \rho & \rho^2 & \rho^3 \\ \rho & 1 & \rho & \rho^2 \\ \rho^2 & \rho & 1 & \rho \\ \rho^3 & \rho^2 & \rho & 1 \end{bmatrix}$
Toeplitz	TOEP	$\begin{bmatrix} \sigma^2 & \sigma_1 & \sigma_2 & \sigma_3 \\ \sigma_1 & \sigma^2 & \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_1 & \sigma^2 & \sigma_1 \\ \sigma_3 & \sigma_2 & \sigma_1 & \sigma^2 \end{bmatrix}$

Table 7.6 *continued*

Description	Structure	Example
Toeplitz with two bands	TOEP(2)	$\begin{bmatrix} \sigma^2 & \sigma_1 & 0 & 0 \\ \sigma_1 & \sigma^2 & \sigma_1 & 0 \\ 0 & \sigma_1 & \sigma^2 & \sigma_1 \\ 0 & 0 & \sigma_1 & \sigma^2 \end{bmatrix}$
Heterogeneous autoregressive(1)	ARH(1)	$\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho & \sigma_1\sigma_3\rho^2 & \sigma_1\sigma_4\rho^3 \\ \sigma_2\sigma_1\rho & \sigma_2^2 & \sigma_2\sigma_3\rho & \sigma_2\sigma_4\rho^2 \\ \sigma_3\sigma_1\rho^2 & \sigma_3\sigma_2\rho & \sigma_3^2 & \sigma_3\sigma_4\rho \\ \sigma_4\sigma_1\rho^3 & \sigma_4\sigma_2\rho & \sigma_4\sigma_3\rho & \sigma_4^2 \end{bmatrix}$
First-order autoregressive moving average	ARMA(1,1)	$\sigma^2 \begin{bmatrix} 1 & \gamma & \gamma\rho & \gamma\rho^2 \\ \gamma & 1 & \gamma & \gamma\rho \\ \gamma\rho & \gamma & 1 & \gamma \\ \gamma\rho^2 & \gamma\rho & \gamma & 1 \end{bmatrix}$
Heterogeneous compound symmetry	CSH	$\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho & \sigma_1\sigma_3\rho & \sigma_1\sigma_4\rho \\ \sigma_2\sigma_1\rho & \sigma_2^2 & \sigma_2\sigma_3\rho & \sigma_2\sigma_4\rho \\ \sigma_3\sigma_1\rho & \sigma_3\sigma_2\rho & \sigma_3^2 & \sigma_3\sigma_4\rho \\ \sigma_4\sigma_1\rho & \sigma_4\sigma_2\rho & \sigma_4\sigma_3\rho & \sigma_4^2 \end{bmatrix}$
First-order factor analytic	FA(1)	$\begin{bmatrix} \lambda_1^2 + d_1 & \lambda_1\lambda_2 & \lambda_1\lambda_3 & \lambda_1\lambda_4 \\ \lambda_2\lambda_1 & \lambda_2^2 + d_2 & \lambda_2\lambda_3 & \lambda_2\lambda_4 \\ \lambda_3\lambda_1 & \lambda_3\lambda_2 & \lambda_3^2 + d_3 & \lambda_3\lambda_4 \\ \lambda_4\lambda_1 & \lambda_4\lambda_2 & \lambda_4\lambda_3 & \lambda_4^2 + d_4 \end{bmatrix}$
Huynh-Feldt	HF	$\begin{bmatrix} \sigma_1^2 & \frac{\sigma_1^2+\sigma_2^2}{2} - \lambda & \frac{\sigma_1^2+\sigma_3^2}{2} - \lambda \\ \frac{\sigma_2^2+\sigma_1^2}{2} - \lambda & \sigma_2^2 & \frac{\sigma_2^2+\sigma_3^2}{2} - \lambda \\ \frac{\sigma_3^2+\sigma_1^2}{2} - \lambda & \frac{\sigma_3^2+\sigma_2^2}{2} - \lambda & \sigma_3^2 \end{bmatrix}$
First-order antedependence	ANTE(1)	$\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho_1 & \sigma_1\sigma_3\rho_1\rho_2 \\ \sigma_2\sigma_1\rho_1 & \sigma_2^2 & \sigma_2\sigma_3\rho_2 \\ \sigma_3\sigma_1\rho_2\rho_1 & \sigma_3\sigma_2\rho_2 & \sigma_3^2 \end{bmatrix}$
Heterogeneous Toeplitz	TOEPH	$\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho_1 & \sigma_1\sigma_3\rho_2 & \sigma_1\sigma_4\rho_3 \\ \sigma_2\sigma_1\rho_1 & \sigma_2^2 & \sigma_2\sigma_3\rho_1 & \sigma_2\sigma_4\rho_2 \\ \sigma_3\sigma_1\rho_2 & \sigma_3\sigma_2\rho_1 & \sigma_3^2 & \sigma_3\sigma_4\rho_1 \\ \sigma_4\sigma_1\rho_3 & \sigma_4\sigma_2\rho_2 & \sigma_4\sigma_3\rho_1 & \sigma_4^2 \end{bmatrix}$
Unstructured correlations	UNR	$\begin{bmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho_{21} & \sigma_1\sigma_3\rho_{31} & \sigma_1\sigma_4\rho_{41} \\ \sigma_2\sigma_1\rho_{21} & \sigma_2^2 & \sigma_2\sigma_3\rho_{32} & \sigma_2\sigma_4\rho_{42} \\ \sigma_3\sigma_1\rho_{31} & \sigma_3\sigma_2\rho_{32} & \sigma_3^2 & \sigma_3\sigma_4\rho_{43} \\ \sigma_4\sigma_1\rho_{41} & \sigma_4\sigma_2\rho_{42} & \sigma_4\sigma_3\rho_{43} & \sigma_4^2 \end{bmatrix}$

The following list provides some further information about these covariance structures:

TYPE=ANTE(1) specifies the first-order antedependence structure (Kenward 1987; Patel 1991; Macchiavelli and Arnold 1994). In Table 7.5, σ_i^2 is the i variance parameter, and ρ_k is the k autocorrelation parameter that satisfies $|\rho_k| < 1$.

TYPE=AR(1) specifies a first-order autoregressive structure. PROC HPLMIXED imposes the constraint $|\rho| < 1$ for stationarity.

TYPE=ARH(1) specifies a heterogeneous first-order autoregressive structure. As with TYPE=AR(1), PROC HPLMIXED imposes the constraint $|\rho| < 1$ for stationarity.

TYPE=ARMA(1,1) specifies the first-order autoregressive moving average structure. In Table 7.5, ρ is the autoregressive parameter, γ models a moving average component, and σ^2 is the residual variance. In the notation of Fuller (1976, p. 68), $\rho = \theta_1$ and

$$\gamma = \frac{(1 + b_1\theta_1)(\theta_1 + b_1)}{1 + b_1^2 + 2b_1\theta_1}$$

The example in Table 7.6 and $|b_1| < 1$ imply that

$$b_1 = \frac{\beta - \sqrt{\beta^2 - 4\alpha^2}}{2\alpha}$$

where $\alpha = \gamma - \rho$ and $\beta = 1 + \rho^2 - 2\gamma\rho$. PROC HPLMIXED imposes the constraints $|\rho| < 1$ and $|\gamma| < 1$ for stationarity, although the resulting covariance matrix is not positive definite for some values of ρ and γ in this region. When the estimated value of ρ becomes negative, the computed covariance is multiplied by $\cos(\pi d_{ij})$ to account for the negativity.

TYPE=CS specifies the compound-symmetry structure, which has constant variance and constant covariance.

TYPE=CSH specifies the heterogeneous compound-symmetry structure. This structure has a different variance parameter for each diagonal element, and it uses the square roots of these parameters in the off-diagonal entries. In Table 7.5, σ_i^2 is the i variance parameter, and ρ is the correlation parameter that satisfies $|\rho| < 1$.

TYPE=FA(q) specifies the factor-analytic structure with q factors (Jennrich and Schluchter 1986). This structure is of the form $\mathbf{\Lambda\Lambda}' + \mathbf{D}$, where $\mathbf{\Lambda}$ is a $t \times q$ rectangular matrix and $\mathbf{D}$ is a $t \times t$ diagonal matrix with t different parameters. When $q > 1$, the elements of $\mathbf{\Lambda}$ in its upper right corner (that is, the elements in the i row and j column for $j > i$) are set to zero to fix the rotation of the structure.

TYPE=FA0(q) is similar to the FA(q) structure except that no diagonal matrix $\mathbf{D}$ is included. When $q < t$ (that is, when the number of factors is less than the dimension of the matrix), this structure is nonnegative definite but not of full rank. In this situation, you can use this structure for approximating an unstructured $\mathbf{G}$ matrix in the RANDOM statement. When $q = t$, you can use this structure to constrain $\mathbf{G}$ to be nonnegative definite in the RANDOM statement.

TYPE=FA1(q) is similar to the TYPE=FA(q) structure except that all of the elements in $\mathbf{D}$ are constrained to be equal. This offers a useful and more parsimonious alternative to the full factor-analytic structure.

TYPE=HF specifies the Huynh-Feldt covariance structure (Huynh and Feldt 1970). This structure is similar to the TYPE=CSH structure in that it has the same number of parameters and heterogeneity along the main diagonal. However, it constructs the off-diagonal elements by taking arithmetic means rather than geometric means.

You can perform a likelihood ratio test of the Huynh-Feldt conditions by running PROC HPLMIXED twice, once with TYPE=HF and once with TYPE=UN, and then subtracting their respective values of -2 times the maximized likelihood.

If PROC HPLMIXED does not converge under your Huynh-Feldt model, you can specify your own starting values with the PARMS statement. The default MIVQUE(0) starting values can sometimes be poor for this structure. A good choice for starting values is often the parameter estimates that correspond to an initial fit that uses TYPE=CS.

TYPE=SIMPLE is an alias for TYPE=VC.

TYPE=TOEP<(q)> specifies a banded Toeplitz structure. This can be viewed as a moving average structure with order equal to $q - 1$. The TYPE=TOEP option is a full Toeplitz matrix, which can be viewed as an autoregressive structure with order equal to the dimension of the matrix. The specification TYPE=TOEP(1) is the same as $\sigma^2 I$, where I is an identity matrix, and it can be useful for specifying the same variance component for several effects.

TYPE=TOEPH<(q)> specifies a heterogeneous banded Toeplitz structure. In Table 7.5, σ_i^2 is the i variance parameter and ρ_j is the j correlation parameter that satisfies $|\rho_j| < 1$. If you specify the order parameter q, then PROC HPLMIXED estimates only the first q bands of the matrix, setting all higher bands equal to 0. The option TOEPH(1) is equivalent to both the TYPE=UN(1) and TYPE=UNR(1) options.

TYPE=UN<(q)> specifies a completely general (unstructured) covariance matrix that is parameterized directly in terms of variances and covariances. The variances are constrained to be nonnegative, and the covariances are unconstrained. This structure is not constrained to be nonnegative definite in order to avoid nonlinear constraints. However, you can use the TYPE=FA0 structure if you want this constraint to be imposed by a Cholesky factorization. If you specify the order parameter q, then PROC HPLMIXED estimates only the first q bands of the matrix, setting all higher bands equal to 0.

TYPE=UNR<(q)> specifies a completely general (unstructured) covariance matrix that is parameterized in terms of variances and correlations. This structure fits the same model as the TYPE=UN(q) option but with a different parameterization. The i variance parameter is σ_i^2. The parameter ρ_{jk} is the correlation between the j and k measurements; it satisfies $|\rho_{jk}| < 1$. If you specify the order parameter r, then PROC HPLMIXED estimates only the first q bands of the matrix, setting all higher bands equal to zero.

TYPE=VC specifies standard variance components. This is the default structure for both the RANDOM and REPEATED statements. In the RANDOM statement, a distinct variance component is assigned to each effect.

Jennrich and Schluchter (1986) provide general information about the use of covariance structures, and Wolfinger (1996) presents details about many of the heterogeneous structures.

REPEATED Statement

REPEATED *repeated-effect* < / *options* > **;**

The REPEATED statement specifies the **R** matrix in the mixed model. If no REPEATED statement is specified, **R** is assumed to be equal to $\sigma^2\mathbf{I}$. For this release, you can use the REPEATED statement only with the BLUP option. The statement is ignored when no BLUP option is specified.

The *repeated-effect* is required, since the order of the input data is not necessarily reproducible in a distributed environment. The *repeated-effect* must contain only classification variables. Make sure that the levels of the *repeated-effect* are different for each observation within a subject; otherwise, PROC HPLMIXED constructs identical rows in **R** that correspond to the observations with the same level. This results in a singular **R** matrix and an infinite likelihood.

Table 7.7 summarizes important options in the REPEATED statement. All options are subsequently discussed in alphabetical order.

Table 7.7 Summary of Important REPEATED Statement Options

Option	Description
Construction of Covariance Structure	
SUBJECT=	Identifies the subjects in the R-side model
TYPE=	Specifies the R-side covariance structure

You can specify the following *options* in the REPEATED statement after a slash (/).

SUBJECT=*effect*

SUB=*effect*

identifies the subjects in your mixed model. Complete independence is assumed across subjects; therefore, the SUBJECT= option produces a block-diagonal structure in **R** with identical blocks. When the SUBJECT= effect consists entirely of classification variables, the blocks of **R** correspond to observations that share the same level of that effect. These blocks are sorted according to this effect as well.

If you want to model nonzero covariance among all of the observations in your SAS data set, specify SUBJECT=Dummy_Intercept to treat the data as if they are all from one subject. You need to create this Dummy_Intercept variable in the data set. However, be aware that in this case PROC HPLMIXED manipulates an **R** matrix with dimensions equal to the number of observations.

TYPE=*covariance-structure*

specifies the covariance structure of the **R** matrix. The SUBJECT= option defines the blocks of **R**, and the TYPE= option specifies the structure of these blocks. The default structure is VC. You can specify any of the covariance structures that are described in the TYPE= option in the RANDOM statement.

Details: HPLMIXED Procedure

Linear Mixed Models Theory

This section provides an overview of a likelihood-based approach to linear mixed models. This approach simplifies and unifies many common statistical analyses, including those that involve repeated measures, random effects, and random coefficients. The basic assumption is that the data are linearly related to unobserved multivariate normal random variables. For extensions to nonlinear and nonnormal situations, see the documentation of the GLIMMIX and NLMIXED procedures in the *SAS/STAT User's Guide*. Additional theory and examples are provided in Littell et al. (2006), Verbeke and Molenberghs (1997, 2000), and Brown and Prescott (1999).

Matrix Notation

Suppose that you observe n data points $y_1, \ldots, y_n$ and that you want to explain them by using n values for each of p explanatory variables $x_{11}, \ldots, x_{1p}, x_{21}, \ldots, x_{2p}, \ldots, x_{n1}, \ldots, x_{np}$. The x_{ij} values can be either regression-type continuous variables or dummy variables that indicate class membership. The standard linear model for this setup is

$$y_i = \sum_{j=1}^{p} x_{ij}\beta_j + \epsilon_i \quad i = 1, \ldots, n$$

where $\beta_1, \ldots, \beta_p$ are unknown fixed-effects parameters to be estimated and $\epsilon_1, \ldots, \epsilon_n$ are unknown independent and identically distributed normal (Gaussian) random variables with mean 0 and variance σ^2.

The preceding equations can be written simultaneously by using vectors and a matrix, as follows:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

For convenience, simplicity, and extendability, this entire system is written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{y}$ denotes the vector of observed y_i's, $\mathbf{X}$ is the known matrix of x_{ij}'s, $\boldsymbol{\beta}$ is the unknown fixed-effects parameter vector, and $\boldsymbol{\epsilon}$ is the unobserved vector of independent and identically distributed Gaussian random errors.

In addition to denoting data, random variables, and explanatory variables in the preceding fashion, the subsequent development makes use of basic matrix operators such as transpose ($'$), inverse ($^{-1}$), generalized inverse ($^{-}$), determinant ($|\cdot|$), and matrix multiplication. See Searle (1982) for details about these and other matrix techniques.

Formulation of the Mixed Model

The previous general linear model is certainly a useful one (Searle 1971), and it is the one fitted by the GLM procedure. However, many times the distributional assumption about ϵ is too restrictive. The mixed model extends the general linear model by allowing a more flexible specification of the covariance matrix of ϵ. In other words, it allows for both correlation and heterogeneous variances, although you still assume normality.

The mixed model is written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\boldsymbol{\gamma} + \boldsymbol{\epsilon}$$

where everything is the same as in the general linear model except for the addition of the known design matrix, $\mathbf{Z}$, and the vector of unknown random-effects parameters, $\boldsymbol{\gamma}$. The matrix $\mathbf{Z}$ can contain either continuous or dummy variables, just like $\mathbf{X}$. The name *mixed model* comes from the fact that the model contains both fixed-effects parameters, $\boldsymbol{\beta}$, and random-effects parameters, $\boldsymbol{\gamma}$. See Henderson (1990) and Searle, Casella, and McCulloch (1992) for historical developments of the mixed model.

A key assumption in the foregoing analysis is that $\boldsymbol{\gamma}$ and $\boldsymbol{\epsilon}$ are normally distributed with

$$\mathrm{E}\begin{bmatrix} \boldsymbol{\gamma} \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

$$\mathrm{Var}\begin{bmatrix} \boldsymbol{\gamma} \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

Therefore, the variance of $\mathbf{y}$ is $\mathbf{V} = \mathbf{Z}\mathbf{G}\mathbf{Z}' + \mathbf{R}$. You can model $\mathbf{V}$ by setting up the random-effects design matrix $\mathbf{Z}$ and by specifying covariance structures for $\mathbf{G}$ and $\mathbf{R}$.

Note that this is a general specification of the mixed model, in contrast to many texts and articles that discuss only simple random effects. Simple random effects are a special case of the general specification with $\mathbf{Z}$ containing dummy variables, $\mathbf{G}$ containing variance components in a diagonal structure, and $\mathbf{R} = \sigma^2\mathbf{I}_n$, where $\mathbf{I}_n$ denotes the $n \times n$ identity matrix. The general linear model is a further special case with $\mathbf{Z} = \mathbf{0}$ and $\mathbf{R} = \sigma^2\mathbf{I}_n$.

The following two examples illustrate the most common formulations of the general linear mixed model.

Example: Growth Curve with Compound Symmetry

Suppose that you have three growth curve measurements for s individuals and that you want to fit an overall linear trend in time. Your $\mathbf{X}$ matrix is as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ \vdots & \vdots \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$$

The first column (coded entirely with 1s) fits an intercept, and the second column (coded with series of $1, 2, 3$) fits a slope. Here, $n = 3s$ and $p = 2$.

Suppose further that you want to introduce a common correlation among the observations from a single individual, with correlation being the same for all individuals. One way of setting this up in the general mixed model is to eliminate the **Z** and **G** matrices and let the **R** matrix be block-diagonal with blocks corresponding to the individuals and with each block having the *compound-symmetry* structure. This structure has two unknown parameters, one modeling a common covariance and the other modeling a residual variance. The form for **R** would then be

$$\mathbf{R} = \begin{bmatrix} \sigma_1^2+\sigma^2 & \sigma_1^2 & \sigma_1^2 & & & & \\ \sigma_1^2 & \sigma_1^2+\sigma^2 & \sigma_1^2 & & & & \\ \sigma_1^2 & \sigma_1^2 & \sigma_1^2+\sigma^2 & & & & \\ & & & \ddots & & & \\ & & & & \sigma_1^2+\sigma^2 & \sigma_1^2 & \sigma_1^2 \\ & & & & \sigma_1^2 & \sigma_1^2+\sigma^2 & \sigma_1^2 \\ & & & & \sigma_1^2 & \sigma_1^2 & \sigma_1^2+\sigma^2 \end{bmatrix}$$

where blanks denote zeros. There are $3s$ rows and columns altogether, and the common correlation is $\sigma_1^2/(\sigma_1^2+\sigma^2)$.

The following PROC HPLMIXED statements fit this model:

```
proc hplmixed;
   class indiv;
   model y = time;
   repeated morder/ type=cs subject=indiv;
run;
```

Here, INDIV is a classification variable that indexes individuals. The MODEL statement fits a straight line for TIME ; the intercept is fit by default just as in PROC GLM. The REPEATED statement models the **R** matrix: TYPE=CS specifies the compound symmetry structure, and SUBJECT=INDIV specifies the blocks of **R**, and MORDER is the repeated effect that records the order of the measurements for each individual.

An alternative way of specifying the common intra-individual correlation is to let

$$\mathbf{Z} = \begin{bmatrix} 1 & & & \\ 1 & & & \\ 1 & & & \\ & 1 & & \\ & 1 & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & 1 \\ & & & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_1^2 & & \\ & & \ddots & \\ & & & \sigma_1^2 \end{bmatrix}$$

and $\mathbf{R} = \sigma^2\mathbf{I}_n$. The **Z** matrix has $3s$ rows and s columns, and **G** is $s \times s$.

You can set up this model in PROC HPLMIXED in two different but equivalent ways:

```
proc hplmixed;
   class indiv;
   model y = time;
   random indiv;
run;

proc hplmixed;
   class indiv;
   model y = time;
   random intercept / subject=indiv;
run;
```

Both of these specifications fit the same model as the previous one that used the REPEATED statement. However, the RANDOM specifications constrain the correlation to be positive, whereas the REPEATED specification leaves the correlation unconstrained.

Example: Split-Plot Design

The split-plot design involves two experimental treatment factors, A and B, and two different sizes of experimental units to which they are applied (Winer 1971; Snedecor and Cochran 1980; Milliken and Johnson 1992; Steel, Torrie, and Dickey 1997). The levels of A are randomly assigned to the larger-sized experimental units, called *whole plots*, whereas the levels of B are assigned to the smaller-sized experimental units, the *subplots*. The subplots are assumed to be nested within the whole plots, so that a whole plot consists of a cluster of subplots and a level of A is applied to the entire cluster.

Such an arrangement is often necessary by nature of the experiment; the classical example is the application of fertilizer to large plots of land and different crop varieties planted in subdivisions of the large plots. For this example, fertilizer is the whole-plot factor A and variety is the subplot factor B.

The first example is a split-plot design for which the whole plots are arranged in a randomized block design. The appropriate PROC HPLMIXED statements are as follows:

```
proc hplmixed;
   class a b block;
   model y = a b a*b;
   random block a*block;
run;
```

Here

$$\mathbf{R} = \sigma^2 \mathbf{I}_{24}$$

and **X**, **Z**, and **G** have the following form:

$$\mathbf{X} = \begin{bmatrix}
1 & 1 & & & 1 & & 1 & & & & & \\
1 & 1 & & & & 1 & & 1 & & & & \\
1 & & 1 & & 1 & & & & 1 & & & \\
1 & & 1 & & & 1 & & & & 1 & & \\
1 & & & 1 & 1 & & & & & & 1 & \\
1 & & & 1 & & 1 & & & & & & 1 \\
\vdots & & \vdots & & \vdots & & & & & \vdots & & \\
1 & 1 & & & 1 & & 1 & & & & & \\
1 & 1 & & & & 1 & & 1 & & & & \\
1 & & 1 & & 1 & & & & 1 & & & \\
1 & & 1 & & & 1 & & & & 1 & & \\
1 & & & 1 & 1 & & & & & & 1 & \\
1 & & & 1 & & 1 & & & & & & 1
\end{bmatrix}$$

$$\mathbf{Z} = \begin{bmatrix}
1 & & & & 1 & & & & & & & & & & & \\
1 & & & & 1 & & & & & & & & & & & \\
1 & & & & & 1 & & & & & & & & & & \\
1 & & & & & 1 & & & & & & & & & & \\
1 & & & & & & 1 & & & & & & & & & \\
1 & & & & & & 1 & & & & & & & & & \\
 & 1 & & & & & & 1 & & & & & & & & \\
 & 1 & & & & & & 1 & & & & & & & & \\
 & 1 & & & & & & & 1 & & & & & & & \\
 & 1 & & & & & & & 1 & & & & & & & \\
 & 1 & & & & & & & & 1 & & & & & & \\
 & 1 & & & & & & & & 1 & & & & & & \\
 & & 1 & & & & & & & & 1 & & & & & \\
 & & 1 & & & & & & & & 1 & & & & & \\
 & & 1 & & & & & & & & & 1 & & & & \\
 & & 1 & & & & & & & & & 1 & & & & \\
 & & 1 & & & & & & & & & & 1 & & & \\
 & & 1 & & & & & & & & & & 1 & & & \\
 & & & 1 & & & & & & & & & & 1 & & \\
 & & & 1 & & & & & & & & & & 1 & & \\
 & & & 1 & & & & & & & & & & & 1 & \\
 & & & 1 & & & & & & & & & & & 1 & \\
 & & & 1 & & & & & & & & & & & & 1 \\
 & & & 1 & & & & & & & & & & & & 1
\end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix}
\sigma_B^2 & & & & & & & \\
 & \sigma_B^2 & & & & & & \\
 & & \sigma_B^2 & & & & & \\
 & & & \sigma_B^2 & & & & \\
 & & & & \sigma_{AB}^2 & & & \\
 & & & & & \sigma_{AB}^2 & & \\
 & & & & & & \ddots & \\
 & & & & & & & \sigma_{AB}^2
\end{bmatrix}$$

where σ_B^2 is the variance component for Block and σ_{AB}^2 is the variance component for A*Block. Changing the RANDOM statement as follows fits the same model, but with **Z** and **G** sorted differently:

```
random int a / subject=block;
```

$$
\mathbf{Z} = \begin{bmatrix}
1 & 1 & & & & & & & & & & & & & & \\
1 & 1 & & & & & & & & & & & & & & \\
1 & & 1 & & & & & & & & & & & & & \\
1 & & 1 & & & & & & & & & & & & & \\
1 & & & 1 & & & & & & & & & & & & \\
1 & & & 1 & & & & & & & & & & & & \\
 & & & & 1 & 1 & & & & & & & & & & \\
 & & & & 1 & 1 & & & & & & & & & & \\
 & & & & 1 & & 1 & & & & & & & & & \\
 & & & & 1 & & 1 & & & & & & & & & \\
 & & & & 1 & & & 1 & & & & & & & & \\
 & & & & 1 & & & 1 & & & & & & & & \\
 & & & & & & & & 1 & 1 & & & & & & \\
 & & & & & & & & 1 & 1 & & & & & & \\
 & & & & & & & & 1 & & 1 & & & & & \\
 & & & & & & & & 1 & & 1 & & & & & \\
 & & & & & & & & 1 & & & 1 & & & & \\
 & & & & & & & & 1 & & & 1 & & & & \\
 & & & & & & & & & & & & 1 & 1 & & \\
 & & & & & & & & & & & & 1 & 1 & & \\
 & & & & & & & & & & & & 1 & & 1 & \\
 & & & & & & & & & & & & 1 & & 1 & \\
 & & & & & & & & & & & & 1 & & & 1 \\
 & & & & & & & & & & & & 1 & & & 1
\end{bmatrix}
$$

$$
\mathbf{G} = \begin{bmatrix}
\sigma_B^2 & & & & & & & & \\
 & \sigma_{AB}^2 & & & & & & & \\
 & & \sigma_{AB}^2 & & & & & & \\
 & & & \sigma_{AB}^2 & & & & & \\
 & & & & \ddots & & & & \\
 & & & & & \sigma_B^2 & & & \\
 & & & & & & \sigma_{AB}^2 & & \\
 & & & & & & & \sigma_{AB}^2 & \\
 & & & & & & & & \sigma_{AB}^2
\end{bmatrix}
$$

Estimating Covariance Parameters in the Mixed Model

Estimation is more difficult in the mixed model than in the general linear model. Not only do you have $\boldsymbol{\beta}$ as in the general linear model, but you also have unknown parameters in $\boldsymbol{\gamma}$, **G**, and **R** . Least squares is no longer the best method. Generalized least squares (GLS) is more appropriate, minimizing

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'\mathbf{V}^{-1}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

However, GLS requires knowledge of $\mathbf{V}$ and therefore knowledge of $\mathbf{G}$ and $\mathbf{R}$. Lacking such information, one approach is to use an *estimated* GLS, in which you insert some reasonable estimate for $\mathbf{V}$ into the minimization problem. The goal thus becomes finding a reasonable estimate of $\mathbf{G}$ and $\mathbf{R}$.

In many situations, the best approach is to use likelihood-based methods, exploiting the assumption that $\boldsymbol{\gamma}$ and $\boldsymbol{\epsilon}$ are normally distributed (Hartley and Rao 1967; Patterson and Thompson 1971; Harville 1977; Laird and Ware 1982; Jennrich and Schluchter 1986). PROC HPLMIXED implements two likelihood-based methods: maximum likelihood (ML) and restricted (residual) maximum likelihood (REML). A favorable theoretical property of ML and REML is that they accommodate data that are missing at random (Rubin 1976; Little 1995).

PROC HPLMIXED constructs an objective function associated with ML or REML and maximizes it over all unknown parameters. Using calculus, it is possible to reduce this maximization problem to one over only the parameters in $\mathbf{G}$ and $\mathbf{R}$. The corresponding log-likelihood functions are as follows:

$$\begin{aligned}\text{ML}: \quad l(\mathbf{G},\mathbf{R}) &= -\frac{1}{2}\log|\mathbf{V}| - \frac{1}{2}\mathbf{r}'\mathbf{V}^{-1}\mathbf{r} - \frac{n}{2}\log(2\pi) \\ \text{REML}: \quad l_R(\mathbf{G},\mathbf{R}) &= -\frac{1}{2}\log|\mathbf{V}| - \frac{1}{2}\log|\mathbf{X}'\mathbf{V}^{-1}\mathbf{X}| - \frac{1}{2}\mathbf{r}'\mathbf{V}^{-1}\mathbf{r} - \frac{n-p}{2}\log(2\pi)\end{aligned}$$

where $\mathbf{r} = \mathbf{y} - \mathbf{X}(\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$ and p is the rank of $\mathbf{X}$. By default, PROC HPLMIXED actually minimizes a normalized form of -2 times these functions by using a ridge-stabilized Newton-Raphson algorithm by default. Lindstrom and Bates (1988) provide reasons for preferring Newton-Raphson to the expectation-maximum (EM) algorithm described in Dempster, Laird, and Rubin (1977) and Laird, Lange, and Stram (1987), in addition to analytical details for implementing a QR-decomposition approach to the problem. Wolfinger, Tobias, and Sall (1994) present the sweep-based algorithms that are implemented in PROC HPLMIXED. You can change the optimization technique with the TECHNIQUE= option in the PROC HPLMIXED statement.

One advantage of using the Newton-Raphson algorithm is that the second derivative matrix of the objective function evaluated at the optima is available upon completion. Denoting this matrix $\mathbf{H}$, the asymptotic theory of maximum likelihood (Serfling 1980) shows that $2\mathbf{H}^{-1}$ is an asymptotic variance-covariance matrix of the estimated parameters of $\mathbf{G}$ and $\mathbf{R}$. Thus, tests and confidence intervals based on asymptotic normality can be obtained. However, these can be unreliable in small samples, especially for parameters such as variance components that have sampling distributions that tend to be skewed to the right.

If a residual variance σ^2 is a part of your mixed model, it can usually be *profiled* out of the likelihood. This means solving analytically for the optimal σ^2 and plugging this expression back into the likelihood formula (Wolfinger, Tobias, and Sall 1994). This reduces the number of optimization parameters by 1 and can improve convergence properties. PROC HPLMIXED profiles the residual variance out of the log likelihood.

Estimating Fixed and Random Effects in the Mixed Model

ML and REML methods provide estimates of $\mathbf{G}$ and $\mathbf{R}$, which are denoted $\widehat{\mathbf{G}}$ and $\widehat{\mathbf{R}}$, respectively. To obtain estimates of $\boldsymbol{\beta}$ and predicted values of $\boldsymbol{\gamma}$, the standard method is to solve the *mixed model equations* (Henderson 1984):

$$\left[\begin{array}{cc} \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{Z} \\ \mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{Z} + \widehat{\mathbf{G}}^{-1} \end{array}\right] \left[\begin{array}{c} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{\gamma}} \end{array}\right] = \left[\begin{array}{c} \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{y} \\ \mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{y} \end{array}\right]$$

The solutions can also be written as

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{y}$$
$$\widehat{\boldsymbol{\gamma}} = \widehat{\mathbf{G}}\mathbf{Z}'\widehat{\mathbf{V}}^{-1}(\mathbf{y} - \mathbf{X}\widehat{\boldsymbol{\beta}})$$

and have connections with empirical Bayes estimators (Laird and Ware 1982; Carlin and Louis 1996). Note that the $\boldsymbol{\gamma}$ are random variables and not parameters (unknown constants) in the model. Technically, determining values for $\boldsymbol{\gamma}$ from the data is thus a prediction task, whereas determining values for $\boldsymbol{\beta}$ is an estimation task.

The mixed model equations are extended normal equations. The preceding expression assumes that $\widehat{\mathbf{G}}$ is nonsingular. For the extreme case where the eigenvalues of $\widehat{\mathbf{G}}$ are very large, $\widehat{\mathbf{G}}^{-1}$ contributes very little to the equations and $\widehat{\boldsymbol{\gamma}}$ is close to what it would be if $\boldsymbol{\gamma}$ actually contained fixed-effects parameters. On the other hand, when the eigenvalues of $\widehat{\mathbf{G}}$ are very small, $\widehat{\mathbf{G}}^{-1}$ dominates the equations and $\widehat{\boldsymbol{\gamma}}$ is close to 0. For intermediate cases, $\widehat{\mathbf{G}}^{-1}$ can be viewed as shrinking the fixed-effects estimates of $\boldsymbol{\gamma}$ toward 0 (Robinson 1991).

If $\widehat{\mathbf{G}}$ is singular, then the mixed model equations are modified (Henderson 1984) as follows:

$$\begin{bmatrix} \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{Z}\widehat{\mathbf{G}} \\ \widehat{\mathbf{G}}'\mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \widehat{\mathbf{G}}'\mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{Z}\widehat{\mathbf{G}} + \mathbf{G} \end{bmatrix} \begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{\tau}} \end{bmatrix} = \begin{bmatrix} \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{y} \\ \widehat{\mathbf{G}}'\mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{y} \end{bmatrix}$$

Denote the generalized inverses of the nonsingular $\widehat{\mathbf{G}}$ and singular $\widehat{\mathbf{G}}$ forms of the mixed model equations by $\mathbf{C}$ and $\mathbf{M}$, respectively. In the nonsingular case, the solution $\widehat{\boldsymbol{\gamma}}$ estimates the random effects directly. But in the singular case, the estimates of random effects are achieved through a back-transformation $\widehat{\boldsymbol{\gamma}} = \widehat{\mathbf{G}}\widehat{\boldsymbol{\tau}}$ where $\widehat{\boldsymbol{\tau}}$ is the solution to the modified mixed model equations. Similarly, while in the nonsingular case $\mathbf{C}$ itself is the estimated covariance matrix for $(\widehat{\boldsymbol{\beta}}, \widehat{\boldsymbol{\gamma}})$, in the singular case the covariance estimate for $(\widehat{\boldsymbol{\beta}}, \widehat{\mathbf{G}}\widehat{\boldsymbol{\tau}})$ is given by $\mathbf{PMP}$ where

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} & \\ & \widehat{\mathbf{G}} \end{bmatrix}$$

An example of when the singular form of the equations is necessary is when a variance component estimate falls on the boundary constraint of 0.

Statistical Properties

If $\mathbf{G}$ and $\mathbf{R}$ are known, $\widehat{\boldsymbol{\beta}}$ is the best linear unbiased estimator (BLUE) of $\boldsymbol{\beta}$, and $\widehat{\boldsymbol{\gamma}}$ is the best linear unbiased predictor (BLUP) of $\boldsymbol{\gamma}$ (Searle 1971; Harville 1988, 1990; Robinson 1991; McLean, Sanders, and Stroup 1991). Here, "best" means minimum mean squared error. The covariance matrix of $(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}, \widehat{\boldsymbol{\gamma}} - \boldsymbol{\gamma})$ is

$$\mathbf{C} = \begin{bmatrix} \mathbf{X}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{X}'\mathbf{R}^{-1}\mathbf{Z} \\ \mathbf{Z}'\mathbf{R}^{-1}\mathbf{X} & \mathbf{Z}'\mathbf{R}^{-1}\mathbf{Z} + \mathbf{G}^{-1} \end{bmatrix}^{-}$$

where $^{-}$ denotes a generalized inverse (Searle 1971).

However, $\mathbf{G}$ and $\mathbf{R}$ are usually unknown and are estimated by using one of the aforementioned methods. These estimates, $\widehat{\mathbf{G}}$ and $\widehat{\mathbf{R}}$, are therefore simply substituted into the preceding expression to obtain

$$\widehat{\mathbf{C}} = \begin{bmatrix} \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \mathbf{X}'\widehat{\mathbf{R}}^{-1}\mathbf{Z} \\ \mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{X} & \mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{Z} + \widehat{\mathbf{G}}^{-1} \end{bmatrix}^{-}$$

as the approximate variance-covariance matrix of $(\widehat{\boldsymbol{\beta}}-\boldsymbol{\beta}, \widehat{\boldsymbol{\gamma}}-\boldsymbol{\gamma})$. In this case, the BLUE and BLUP acronyms no longer apply, but the word *empirical* is often added to indicate such an approximation. The appropriate acronyms thus become EBLUE and EBLUP.

McLean and Sanders (1988) show that $\widehat{\mathbf{C}}$ can also be written as

$$\widehat{\mathbf{C}} = \begin{bmatrix} \widehat{\mathbf{C}}_{11} & \widehat{\mathbf{C}}'_{21} \\ \widehat{\mathbf{C}}_{21} & \widehat{\mathbf{C}}_{22} \end{bmatrix}$$

where

$$\begin{aligned} \widehat{\mathbf{C}}_{11} &= (\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{X})^{-} \\ \widehat{\mathbf{C}}_{21} &= -\widehat{\mathbf{G}}\mathbf{Z}'\widehat{\mathbf{V}}^{-1}\mathbf{X}\widehat{\mathbf{C}}_{11} \\ \widehat{\mathbf{C}}_{22} &= (\mathbf{Z}'\widehat{\mathbf{R}}^{-1}\mathbf{Z} + \widehat{\mathbf{G}}^{-1})^{-1} - \widehat{\mathbf{C}}_{21}\mathbf{X}'\widehat{\mathbf{V}}^{-1}\mathbf{Z}\widehat{\mathbf{G}} \end{aligned}$$

Note that $\widehat{\mathbf{C}}_{11}$ is the familiar estimated generalized least squares formula for the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}$.

Computational Method

Distributed Computing

Distributed computing refers to the use of multiple autonomous computers that communicate through a secure network. Distributed computing solves computational problems by dividing them into many tasks, each of which is solved by one or more computers. Each computer in this distributed environment is referred to as a node.

You can specify the number of nodes to use with the NODES= option in the PERFORMANCE statement. Specify NODES=0 to force the execution to be done locally (often referred to as soloist-mode computing).

Multithreading

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

The number of threads spawned by the HPLMIXED procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statements, the HPLMIXED procedure schedules threads as if it executes on a system with four CPUs, regardless of the actual CPU count:

```
options cpucount=4;
```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This specification overrides the system option. Specify NTHREADS=1 to force single-threaded execution.
- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement, which overrides the CPUCOUNT= system option and instructs the HPLMIXED procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output. The HPLMIXED procedure allocates two threads per CPU.

The tasks multithreaded by the HPLMIXED procedure are primarily defined by dividing the data processed on a single machine among the threads—that is, the HPLMIXED procedure implements multithreading through a data-parallel model. For example, if the input data set has 1,000 observations and you are running with four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following:

- variable levelization
- effect levelization
- formation of the crossproducts matrix
- the log-likelihood computation

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

Displayed Output

The following sections describe the output produced by PROC HPLMIXED. The output is organized into various tables, which are discussed in the order of their appearance.

Performance Information

The "Performance Information" table is produced by default. It displays information about the grid host for distributed execution and information about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also displayed, depending on the environment.

Model Information

The "Model Information" table describes the model, some of the variables it involves, and the method used in fitting it. The "Model Information" table also has a row labeled Fixed Effects SE Method. This row describes the method used to compute the approximate standard errors for the fixed-effects parameter estimates and related functions of them.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the CLASS statement.

Dimensions

The “Dimensions” table lists the sizes of relevant matrices. This table can be useful in determining the requirements for CPU time and memory.

Number of Observations

The “Number of Observations” table shows the number of observations read from the data set and the number of observations used in fitting the model.

Optimization Information

The “Optimization Information” table displays important details about the optimization process.

The number of parameters that are updated in the optimization equals the number of parameters in this table minus the number of equality constraints. The number of constraints is displayed if you fix covariance parameters with the HOLD= option in the PARMS statement. The HPLMIXED procedure also lists the number of upper and lower boundary constraints. PROC HPLMIXED might impose boundary constraints for certain parameters, such as variance components and correlation parameters. If you specify the HOLD= option in the PARMS statement, covariance parameters have an upper and lower boundary equal to the parameter value.

Iteration History

The “Iteration History” table describes the optimization of the restricted log likelihood or log likelihood. The function to be minimized (the *objective function*) is $-2l$ for ML and $-2l_R$ for REML; the column name of the objective function in the “Iteration History” table is “-2 Log Like” for ML and “-2 Res Log Like” for REML. The minimization is performed by using a ridge-stabilized Newton-Raphson algorithm, and the rows of this table describe the iterations that this algorithm takes in order to minimize the objective function.

The Evaluations column of the “Iteration History” table tells how many times the objective function is evaluated during each iteration.

The Criterion column of the “Iteration History” table is, by default, a relative Hessian convergence quantity given by

$$\frac{\mathbf{g}_k' \mathbf{H}_k^{-1} \mathbf{g}_k}{|f_k|}$$

where f_k is the value of the objective function at iteration k, $\mathbf{g}_k$ is the gradient (first derivative) of f_k, and $\mathbf{H}_k$ is the Hessian (second derivative) of f_k. If $\mathbf{H}_k$ is singular, then PROC HPLMIXED uses the following relative quantity:

$$\frac{\mathbf{g}_k'\mathbf{g}_k}{|f_k|}$$

To prevent division by $|f_k|$, specify the ABSGCONV option in the PROC HPLMIXED statement. To use a relative function or gradient criterion, specify the FCONV or GCONV option, respectively.

The Hessian criterion is considered superior to function and gradient criteria because it measures orthogonality rather than lack of progress (Bates and Watts 1988). Provided that the initial estimate is feasible and the maximum number of iterations is not exceeded, the Newton-Raphson algorithm is considered to have converged when the criterion is less than the tolerance specified with the FCONV or GCONV option in the PROC HPLMIXED statement. The default tolerance is 1E–8. If convergence is not achieved, PROC HPLMIXED displays the estimates of the parameters at the last iteration.

A convergence criterion that is missing indicates that a boundary constraint has been dropped; it is usually not a cause for concern.

Convergence Status

The "Convergence Status" table displays the status of the iterative estimation process at the end of the optimization. The status appears as a message in the listing, and this message is repeated in the log. The ODS object "ConvergenceStatus" also contains several nonprinting columns that can be helpful in checking the success of the iterative process, in particular during batch processing. The Status variable takes on the value 0 for a successful convergence (even if the Hessian matrix might not be positive definite). The values 1 and 2 of the Status variable indicate lack of convergence and infeasible initial parameter values, respectively. The variable pdG can be used to check whether the $\mathbf{G}$ matrix is positive definite.

For models that are not fit iteratively, such as models without random effects or when the NOITER option is in effect, the "Convergence Status" is not produced.

Covariance Parameter Estimates

The "Covariance Parameter Estimates" table contains the estimates of the parameters in $\mathbf{G}$ and $\mathbf{R}$. (See the section "Estimating Covariance Parameters in the Mixed Model" on page 162.) Their values are labeled in the table along with Subject information if applicable. The estimates are displayed in the Estimate column and are the results of either the REML or the ML estimation method.

Fit Statistics

The "Fit Statistics" table provides some statistics about the estimated mixed model. Expressions for -2 times the log likelihood are provided in the section "Estimating Covariance Parameters in the Mixed Model" on page 162. If the log likelihood is an extremely large number, then PROC HPLMIXED has deemed the estimated $\mathbf{V}$ matrix to be singular. In this case, all subsequent results should be viewed with caution.

In addition, the “Fit Statistics” table lists three information criteria: AIC, AICC, and BIC. All these criteria are in smaller-is-better form and are described in Table 7.8.

Table 7.8 Information Criteria

Criterion	Formula	Reference
AIC	$-2\ell + 2d$	Akaike (1974)
AICC	$-2\ell + 2dn^*/(n^* - d - 1)$	Hurvich and Tsai (1989)
		Burnham and Anderson (1998)
BIC	$-2\ell + d \log n$ for $n > 0$	Schwarz (1978)

Here ℓ denotes the maximum value of the (possibly restricted) log likelihood, d is the dimension of the model. n equals the number of effective subjects as displayed in the “Dimensions” table, unless this value equals 1, in which case n equals the number of levels of the first random effect that you specify in a RANDOM statement. If the number of effective subjects equals 1 and you have no RANDOM statements, then n equals the number of valid observations for maximum likelihood estimation and $n - p$ for restricted maximum likelihood estimation, where p equals the rank of $\mathbf{X}$. For AICC (a finite-sample corrected version of AIC), n^* equals the number of valid observations for maximum likelihood estimation and $n - p$ equals the number of valid observations for restricted maximum likelihood estimation, unless this number is less than $d + 2$, in which case it equals $d + 2$. When $n = 0$, the value of the BIC is -2ℓ.

For restricted likelihood estimation, d equals q, the effective number of estimated covariance parameters. For maximum likelihood estimation, d equals $q + p$.

Timing Information

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a “Timing” table in which the elapsed time for each main task of the procedure is displayed.

ODS Table Names

Each table created by PROC HPLMIXED has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 7.9.

Table 7.9 ODS Tables Produced by PROC HPLMIXED

Table Name	Description	Required Statement / Option
ClassLevels	Level information from the CLASS statement	Default output
ConvergenceStatus	Convergence status	Default output
CovParms	Estimated covariance parameters	Default output
Dimensions	Dimensions of the model	Default output
FitStatistics	Fit statistics	Default output
IterHistory	Iteration history	Default output

Table 7.9 *continued*

Table Name	Description	Required Statement / Option
ModelInfo	Model information	Default output
NObs	Number of observations read and used	Default output
OptInfo	Optimization information	Default output
ParmSearch	Parameter search values	PARMS
PerformanceInfo	Information about high-performance computing environment	Default output
SolutionF	Fixed-effects solution vector	MODEL / S
SolutionR	Random-effects solution vector	RANDOM / S
Timing	Timing breakdown by task	DETAILS option in the PERFORMANCE statement

Examples: HPLMIXED Procedure

Example 7.1: Computing BLUPs for a Large Number of Subjects

Suppose you are using health measurements on patients treated by each medical center to monitor the performance of those centers. Different measurements within each patient are correlated, and there is enough data to fit the parameters of an unstructured covariance model for this correlation. In fact, long experience with historical data provides you with values for the covariance model that are essentially known, and the task is to apply these known values in order to compute best linear unbiased predictors (BLUPs) of the random effect of medical center. You can use these BLUPs to determine the best and worst performing medical centers, adjusting for other factors, on a weekly basis. Another reason why you want to do this with fixed values for the covariance parameters is to make the week-to-week BLUPs more comparable.

Although you cannot use the REPEATED statement in PROC HPLMIXED to fit models in this release, you can use it to compute BLUPs for such models with known values of the variance parameters. For illustration, the following statements create a simulated data set of a given week's worth of patient health measurements across 100 different medical centers. Measurements at three different times are simulated for each patient, and each center has about 50 patients. The simulated model includes a fixed gender effect, a random effect due to center, and covariance between different measurements on the same patient.

```
%let NCenter  = 100;
%let NPatient = %eval(&NCenter*50);
%let NTime    = 3;
%let SigmaC   = 2.0;
%let SigmaP   = 4.0;
%let SigmaE   = 8.0;
%let Seed     = 12345;
```

```
data WeekSim;
   keep Gender Center Patient Time Measurement;
   array PGender{&NPatient};
   array PCenter{&NPatient};
   array PEffect{&NPatient};
   array CEffect{&NCenter};
   array GEffect{2};

   do Center = 1 to &NCenter;
      CEffect{Center} = sqrt(&SigmaC)*rannor(&Seed);
      end;

   GEffect{1} = 10*ranuni(&Seed);
   GEffect{2} = 10*ranuni(&Seed);

   do Patient = 1 to &NPatient;
      PGender{Patient} = 1 + int(2       *ranuni(&Seed));
      PCenter{Patient} = 1 + int(&NCenter*ranuni(&Seed));
      PEffect{Patient} = sqrt(&SigmaP)*rannor(&Seed);
      end;

   do Patient = 1 to &NPatient;
      Gender = PGender{Patient};
      Center = PCenter{Patient};
      Mean = 1 + GEffect{Gender} + CEffect{Center} + PEffect{Patient};
      do Time = 1 to &nTime;
         Measurement = Mean + sqrt(&SigmaE)*rannor(&Seed);
      output;
      end;
   end;
run;
```

Suppose that the known values for the covariance parameters are

$$\begin{aligned} \text{Var(Center)} &= 1.7564 \\ \text{Cov(Patient)} &= \begin{bmatrix} 11.4555 & 3.6883 & 4.5951 \\ 3.6883 & 11.2071 & 3.6311 \\ 4.5951 & 3.6311 & 12.1050 \end{bmatrix} \end{aligned}$$

Incidentally, these are not precisely the same estimates you would get if you estimated these parameters based on the preceding data (for example, with the HPMIXED procedure).

The following statements use PROC HPLMIXED to compute the BLUPs for the random medical center effect. Instead of simply displaying them (as PROC HPMIXED does), PROC HPLMIXED sorts them and displays the five highest and lowest values.

```
ods listing close;
proc hplmixed data=WeekSim blup;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC" nodes=20;
   class Gender Center Patient Time;
   model Measurement = Gender;
   random   Center / s;
   repeated Time   / sub=Patient type=un;
   parms   1.7564
          11.4555
           3.6883 11.2071
           4.5951  3.6311 12.1050;
   ods output SolutionR=BLUPs;
run;
ods listing;

proc sort data=BLUPs;
   by Estimate;
run;

data BLUPs; set BLUPs;
   Rank = _N_;
run;

proc print data=BLUPs;
   where ((Rank <= 5) | (Rank >= 96));
   var Center Estimate;
run;
```

Three parts of the PROC HPLMIXED syntax are required in order to compute BLUPs for this model: the BLUP option in the HPLMIXED statement, the REPEATED statement, and the PARMS statement with fixed values for all parameters. The resulting values of the best and worst performing medical centers for this week are shown in Output 7.1.1. Apparently, for this week's data, medical center 54 had the most decreasing effect, and medical center 48 the most increasing effect, on patient measurements overall.

Output 7.1.1 Highest and Lowest Medical Center BLUPs

```
 Obs    Center    Estimate

   1      54       -2.9369
   2       7       -2.4614
   3      50       -2.2467
   4      51       -2.2281
   5      93       -2.1644
  96      26        2.1603
  97      99        2.2718
  98      44        2.4222
  99      60        2.6089
 100      48        2.6443
```

References

Akaike, H. (1974), "A New Look at the Statistical Model Identification," *IEEE Transaction on Automatic Control*, AC–19, 716–723.

Burdick, R. K. and Graybill, F. A. (1992), *Confidence Intervals on Variance Components,* New York: Marcel Dekker.

Burnham, K. P. and Anderson, D. R. (1998), *Model Selection and Inference: A Practical Information-Theoretic Approach,* New York: Springer-Verlag.

Brown, H. and Prescott, R. (1999), *Applied Mixed Models in Medicine*, New York: John Wiley & Sons.

Carlin, B. P. and Louis, T. A. (1996), *Bayes and Empirical Bayes Methods for Data Analysis,* London: Chapman and Hall.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977), "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society*, Ser. B., 39, 1–38.

Fai, A. H. T. and Cornelius, P. L. (1996), "Approximate F-tests of Multiple Degree of Freedom Hypotheses in Generalized Least Squares Analyses of Unbalanced Split-Plot Experiments," *Journal of Statistical Computation and Simulation,* 54, 363–378.

Fuller, W. A. (1976), *Introduction to Statistical Time Series,* New York: John Wiley & Sons.

Giesbrecht, F. G. and Burns, J. C. (1985), "Two-Stage Analysis Based on a Mixed Model: Large-sample Asymptotic Theory and Small-Sample Simulation Results," *Biometrics*, 41, 477–486.

Hartley, H. O. and Rao, J. N. K. (1967), "Maximum-Likelihood Estimation for the Mixed Analysis of Variance Model," *Biometrika*, 54, 93–108.

Hurvich, C. M. and Tsai, C.-L. (1989), "Regression and Time Series Model Selection in Small Samples," *Biometrika*, 76, 297–307.

Harville, D. A. (1977), "Maximum Likelihood Approaches to Variance Component Estimation and to Related Problems," *Journal of the American Statistical Association*, 72, 320–338.

Harville, D. A. (1988), "Mixed-Model Methodology: Theoretical Justifications and Future Directions," *Proceedings of the Statistical Computing Section*, American Statistical Association, New Orleans, 41–49.

Harville, D. A. (1990), "BLUP (Best Linear Unbiased Prediction), and Beyond," in *Advances in Statistical Methods for Genetic Improvement of Livestock*, Springer-Verlag, 239–276.

Henderson, C. R. (1984), *Applications of Linear Models in Animal Breeding*, University of Guelph.

Henderson, C. R. (1990), "Statistical Method in Animal Improvement: Historical Overview," in *Advances in Statistical Methods for Genetic Improvement of Livestock*, New York: Springer-Verlag, 1–14.

Huynh, H. and Feldt, L. S. (1970), "Conditions Under Which Mean Square Ratios in Repeated Measurements Designs Have Exact F-Distributions," *Journal of the American Statistical Association*, 65, 1582–1589.

Jennrich, R. I. and Schluchter, M. D. (1986), "Unbalanced Repeated-Measures Models with Structured Covariance Matrices," *Biometrics*, 42, 805–820.

Kenward, M. G. (1987), "A Method for Comparing Profiles of Repeated Measurements," *Applied Statistics*, 36, 296–308.

Laird, N. M., Lange, N., and Stram, D. (1987), "Maximum Likelihood Computations with Repeated Measures: Application of the EM Algorithm," *Journal of the American Statistical Association*, 82, 97–105.

Laird, N. M. and Ware, J. H. (1982), "Random-Effects Models for Longitudinal Data," *Biometrics*, 38, 963–974.

Lindstrom, M. J. and Bates, D. M. (1988), "Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data," *Journal of the American Statistical Association*, 83, 1014–1022.

Littell, R. C., Milliken, G. A., Stroup, W. W., Wolfinger, R. D., and Schabenberger, O. (2006), *SAS for Mixed Models,* Second Edition, Cary, NC: SAS Institute Inc.

Little, R. J. A. (1995), "Modeling the Drop-Out Mechanism in Repeated-Measures Studies," *Journal of the American Statistical Association*, 90, 1112–1121.

Macchiavelli, R. E. and Arnold, S. F. (1994), "Variable Order Ante-dependence Models," *Communications in Statistics–Theory and Methods*, 23(9), 2683–2699.

McLean, R. A. and Sanders, W. L. (1988), "Approximating Degrees of Freedom for Standard Errors in Mixed Linear Models," *Proceedings of the Statistical Computing Section*, American Statistical Association, New Orleans, 50–59.

McLean, R. A., Sanders, W. L., and Stroup, W. W. (1991), "A Unified Approach to Mixed Linear Models," *The American Statistician*, 45, 54–64.

Milliken, G. A. and Johnson, D. E. (1992), *Analysis of Messy Data, Volume 1: Designed Experiments*, New York: Chapman and Hall.

Patel, H. I. (1991), "Analysis of Incomplete Data from a Clinical Trial with Repeated Measurements," *Biometrika*, 78, 609–619.

Patterson, H. D. and Thompson, R. (1971), "Recovery of Inter-block Information When Block Sizes Are Unequal," *Biometrika*, 58, 545–554.

Robinson, G. K. (1991), "That BLUP Is a Good Thing: The Estimation of Random Effects," *Statistical Science*, 6, 15–51.

Rubin, D. B. (1976), "Inference and Missing Data," *Biometrika*, 63, 581–592.

Schluchter, M. D. and Elashoff, J. D. (1990), "Small-Sample Adjustments to Tests with Unbalanced Repeated Measures Assuming Several Covariance Structures," *Journal of Statistical Computation and Simulation*, 37, 69–87.

Schwarz, G. (1978), "Estimating the Dimension of a Model," *Annals of Statistics*, 6, 461–464.

Searle, S. R. (1971), *Linear Models*, New York: John Wiley & Sons.

Searle, S. R. (1982), *Matrix Algebra Useful for Statisticians*, New York: John Wiley & Sons.

Searle, S. R., Casella, G., and McCulloch, C. E. (1992), *Variance Components*, New York: John Wiley & Sons.

Serfling, R. J. (1980), *Approximation Theorems of Mathematical Statistics*, New York: John Wiley & Sons.

Snedecor, G. W. and Cochran, W. G. (1980), *Statistical Methods*, Ames: Iowa State University Press.

Steel, R. G. D., Torrie, J. H., and Dickey D. (1997), *Principles and Procedures of Statistics: A Biometrical Approach*, Third Edition, New York: McGraw-Hill, Inc.

Verbeke, G. and Molenberghs, G., eds. (1997), *Linear Mixed Models in Practice: A SAS-Oriented Approach,* New York: Springer.

Verbeke, G. and Molenberghs, G. (2000), *Linear Mixed Models for Longitudinal Data,* New York: Springer.

Winer, B. J. (1971), *Statistical Principles in Experimental Design*, Second Edition, New York: McGraw-Hill, Inc.

Wolfinger, R. D. (1996), "Heterogeneous Variance-Covariance Structures for Repeated Measures," *Journal of Agricultural, Biological, and Environmental Statistics,* 1, 205–230.

Wolfinger, R. D., Tobias, R. D., and Sall, J. (1994), "Computing Gaussian Likelihoods and Their Derivatives for General Linear Mixed Models," *SIAM Journal on Scientific Computing*, 15(6), 1294–1310.

Chapter 8

The HPLOGISTIC Procedure

Contents

Overview: HPLOGISTIC Procedure

The HPLOGISTIC procedure is a high-performance procedure that fits logistic regression models for binary, binomial, and multinomial data on the SAS appliance.

The HPLOGISTIC procedure fits logistic regression models in the broader sense; the procedure permits several link functions and can handle ordinal and nominal data with more than two response categories (multinomial data).

With the HPLOGISTIC procedure you can read and write data in distributed form and perform analyses in parallel in symmetric multiprocessing (SMP) or massively parallel processing (MPP) mode. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details.

PROC HPLOGISTIC Features

The HPLOGISTIC procedure estimates the parameters of a logistic regression model by using maximum likelihood techniques. It also does the following:

- provides the ability to perform analyses on a massively parallel SAS high-performance appliance
- performs parallel reads of input data and parallel writes of output data when the data source is the appliance database
- is highly multithreaded during all phases of analytic execution
- provides model-building syntax with the CLASS and effect-based MODEL statements, which are familiar from SAS/STAT analytic procedures (in particular, the GLM, LOGISTIC, GLIMMIX, and MIXED procedures)
- provides response-variable options as in the LOGISTIC procedure
- performs maximum likelihood estimation
- provides multiple link functions
- provides cumulative link models for ordinal data and generalized logit modeling for unordered multinomial data
- enables model building (variable selection) through the SELECTION statement
- provides a WEIGHT statement for weighted analysis
- provides a FREQ statement for grouped analysis
- provides an OUTPUT statement to produce a data set with predicted probabilities and other observationwise statistics

PROC HPLOGISTIC Contrasted with Other SAS Procedures

For general contrasts, see the section "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10. The following remarks contrast the HPLOGISTIC procedure with the LOGISTIC procedure in SAS/STAT software.

The CLASS statement in the HPLOGISTIC procedure permits two parameterizations: the GLM parameterization and a reference parameterization. In contrast to the LOGISTIC, GENMOD, and other procedures that permit multiple parameterizations, the HPLOGISTIC procedure does not mix parameterizations across the variables in the CLASS statement. In other words, all classification variables have the same parameterization, and this parameterization is either the GLM or reference parameterization.

The default optimization technique used by the LOGISTIC procedure is Fisher scoring; the HPLOGISTIC procedure uses by default a modification of the Newton-Raphson algorithm with a ridged Hessian. You can choose different optimization techniques, including first-order methods that do not require a crossproducts matrix or Hessian, with the TECHNIQUE= option in the PROC HPLOGISTIC statement.

The default parameterization of CLASS variables in the HPLOGISTIC procedure is the GLM parameterization. The LOGISTIC procedure uses the EFFECT parameterization for the CLASS variables by default. In either procedure, you can change the parameterization with the PARAM= option in the CLASS statement.

The LOGISTIC procedure offers a wide variety of postfitting analyses, such as contrasts, estimates, tests of model effects, least squares means, and odds ratios. This release of the HPLOGISTIC procedure is limited in postfitting functionality, since with large data sets the focus is primarily on model fitting and scoring.

The HPLOGISTIC procedure is specifically designed to operate in the high-performance distributed environment. By default, PROC HPLOGISTIC performs computations in multiple threads. The LOGISTIC procedure executes in a single thread.

Getting Started: HPLOGISTIC Procedure

Binary Logistic Regression

The following DATA step contains 100 observations on a dichotomous response variable (y), a character variable (C), and 10 continuous variables (x1–x10):

```
data getStarted;
   input C$ y x1-x10;
   datalines;
   D  0  10.2  6  1.6  38  15  2.4  20  0.8  8.5  3.9
   F  1  12.2  6  2.6  42  61  1.5  10  0.6  8.5  0.7
   D  1   7.7  1  2.1  38  61    1  90  0.6  7.5  5.2
   J  1  10.9  7  3.5  46  42  0.3   0  0.2    6  3.6
   E  0  17.3  6  3.8  26  47  0.9  10  0.4  1.5  4.7
```

```
A  0  18.7  4  1.8   2  34  1.7  80    1  9.5  2.2
B  0   7.2  1  0.3  48  61  1.1  10  0.8  3.5    4
D  0   0.1  3  2.4   0  65  1.6  70  0.8  3.5  0.7
H  1   2.4  4  0.7  38  22  0.2  20    0    3  4.2
J  0  15.6  7  1.4   0  98  0.3   0    1    5  5.2
J  0  11.1  3  2.4  42  55  2.2  60  0.6  4.5  0.7
F  0     4  6  0.9   4  36  2.1  30  0.8    9  4.6
A  0   6.2  2  1.8  14  79  1.1  70  0.2    0  5.1
H  0   3.7  3  0.8  12  66  1.3  40  0.4  0.5  3.3
A  1   9.2  3  2.3  48  51  2.3  50    0    6  5.4
G  0    14  3    2  18  12  2.2   0    0    3  3.4
E  1  19.5  6  3.7  26  81  0.1  30  0.6    5  4.8
C  0    11  3  2.8  38   9  1.7  50  0.8  6.5  0.9
I  0  15.3  7  2.2  20  98  2.7 100  0.4    7  0.8
H  1   7.4  4  0.5  28  65  1.3  60  0.2  9.5  5.4
F  0  11.4  2  1.4  42  12  2.4  10  0.4    1  4.5
C  1  19.4  1  0.4  42   4  2.4  10    0  6.5  0.1
G  0   5.9  4  2.6  12  57  0.8  50  0.4    2  5.8
G  1  15.8  6  3.7  34   8  1.3  90  0.6  2.5  5.7
I  0    10  3  1.9  16  80    3  90  0.4  9.5  1.9
E  0  15.7  1  2.7  32  25  1.7  20  0.2  8.5    6
G  0    11  5  2.9  48  53  0.1  50    1  3.5  1.2
J  1  16.8  0  0.9  14  86  1.4  40  0.8    9    5
D  1    11  4  3.2  48  63  2.8  90  0.6    0  2.2
J  1   4.8  7  3.6  24   1  2.2  20    1  8.5  0.5
J  1  10.4  5    2  42  56    1  20    0  3.5  4.2
G  0  12.7  7  3.6   8  56  2.1  70    1  4.5  1.5
G  0   6.8  1  3.2  30  27  0.6   0  0.8    2  5.6
E  0   8.8  0  3.2   2  67  0.7  10  0.4    1    5
I  1   0.2  0  2.9  10  41  2.3  60  0.2    9  0.3
J  1   4.6  7  3.9  50  61  2.1  50  0.4    3  4.9
J  1   2.3  2  3.2  36  98  0.1  40  0.6  4.5  4.3
I  0  10.8  3  2.7  28  58  0.8  80  0.8    3    6
B  0   9.3  2  3.3  44  44  0.3  50  0.8  5.5  0.4
F  0   9.2  6  0.6   4  64  0.1   0  0.6  4.5  3.9
D  0   7.4  0  2.9  14   0  0.2  30  0.8  7.5  4.5
G  0  18.3  3  3.1   8  60  0.3  60  0.2    7  1.9
F  0   5.3  4  0.2  48  63  2.3  80  0.2    8  5.2
C  0   2.6  5  2.2  24   4  1.3  20    0    2  1.4
F  0  13.8  4  3.6   4   7  1.1  10  0.4  3.5  1.9
B  1  12.4  6  1.7  30  44  1.1  60  0.2    6  1.5
I  0   1.3  1  1.3   8  53  1.1  70  0.6    7  0.8
F  0  18.2  7  1.7  26  92  2.2  30    1  8.5  4.8
J  0   5.2  2  2.2  18  12  1.4  90  0.8    4  4.9
G  1   9.4  2  0.8  22  86  0.4  30  0.4    1  5.9
J  1  10.4  2  1.7  26  31  2.4  10  0.2    7  1.6
J  0    13  1  1.8  14  11  2.3  50  0.6  5.5  2.6
A  0  17.9  4  3.1  46  58  2.6  90  0.6  1.5  3.2
D  1  19.4  6    3  20  50  2.8 100  0.2    9  1.2
I  0  19.6  3  3.6  22  19  1.2   0  0.6    5  4.1
I  1     6  2  1.5  30  30  2.2  20  0.4  8.5  5.3
G  0  13.8  1  2.7   0  52  2.4  20  0.8    6    2
B  0  14.3  4  2.9  30  11  0.6  90  0.6  0.5  4.9
E  0  15.6  0  0.4  38  79  0.4  80  0.4    1  3.3
```

```
   D  0    14  2    1  22  61    3  90  0.6    2  0.1
   C  1   9.4  5  0.4  12  53  1.7  40    0    3  1.1
   H  0  13.2  1  1.6  40  15  0.7  40  0.2    9  5.5
   A  0  13.5  5  2.4  18  89  1.6  20  0.4  9.5  4.7
   E  0   2.6  4  2.3  38   6  0.8  20  0.4    5  5.3
   E  0  12.4  3  1.3  26   8  2.8  10  0.8    6  5.8
   D  0   7.6  2  0.9  44  89  1.3  50  0.8    6  0.4
   I  0  12.7  1  2.3  42   6  2.4  10  0.4    1    3
   C  1  10.7  4  3.2  28  23  2.2  90  0.8  5.5  2.8
   H  0  10.1  2  2.3  10  62  0.9  50  0.4  2.5  3.7
   C  1  16.6  1  0.5  12  88  0.1  20  0.6  5.5  1.8
   I  1   0.2  3  2.2   8  71  1.7  80  0.4  0.5  5.5
   C  0  10.8  4  3.5  30  70  2.3  60  0.4  4.5  5.9
   F  0   7.1  4    3  14  63  2.4  70    0    7  3.1
   D  0  16.5  1  3.3  30  80  1.6  40    0  3.5  2.7
   H  0  17.1  7  2.1  30  45  1.5  60  0.6  0.5  2.8
   D  0   4.3  1  1.5  24  44    0  70    0    5  0.5
   H  0    15  2  0.2  14  87  1.8  50    0  4.5  4.7
   G  0  19.7  3  1.9  36  99  1.5  10  0.6    3  1.7
   H  1   2.8  6  0.6  34  21    2  60    1    9  4.7
   G  0  16.6  3  3.3  46   1  1.4  70  0.6  1.5  5.3
   E  0  11.7  5  2.7  48   4  0.9  60  0.8  4.5  1.6
   F  0  15.6  3  0.2   4  79  0.5   0  0.8  1.5  2.9
   C  1   5.3  6  1.4   8  64    2  80  0.4    9  4.2
   B  1   8.1  7  1.7  40  36  1.4  60  0.6    6  3.9
   I  0  14.8  2  3.2   8  37  0.4  10    0  4.5    3
   D  0   7.4  4    3  12   3  0.6  60  0.6    7  0.7
   D  0   4.8  3  2.3  44  41  1.9  60  0.2    3  3.1
   A  0   4.5  0  0.2   4  48  1.7  80  0.8    9  4.2
   D  0   6.9  6  3.3  14  92  0.5  40  0.4  7.5    5
   B  0   4.7  4  0.9  14  99  2.4  80    1  0.5  0.7
   I  1   7.5  4  2.1  20  79  0.4  40  0.4  2.5  0.7
   C  0   6.1  0  1.4  38  18  2.3  60  0.8  4.5  0.7
   C  0  18.3  1    1  26  98  2.7  20    1  8.5  0.5
   F  0  16.4  7  1.2  32  94  2.9  40  0.4  5.5  2.1
   I  0   9.4  2  2.3  32  42  0.2  70  0.4  8.5  0.3
   F  1  17.9  4  1.3  32  42    2  40  0.2    1  5.4
   H  0  14.9  3  1.6  36  74  2.6  60  0.2    1  2.3
   C  0  12.7  0  2.6   0  88  1.1  80  0.8  0.5  2.1
   F  0   5.4  4  1.5   2   1  1.8  70  0.4  5.5  3.6
   J  1  12.1  4  1.8  20  59  1.3  60  0.4    3  3.8
;
```

The following statements fit a logistic model to these data by using a classification effect for variable C and 10 regressor effects for x1–x10:

```
proc hplogistic data=getStarted;
   class C;
   model y = C x1-x10;
run;
```

The default output from this analysis is presented in Figure 8.1 through Figure 8.11.

The “Performance Information” table in Figure 8.1 shows that the procedure executes in client mode—that is, the model is fit on the machine where the SAS session executes. This run of the HPLOGISTIC procedure was performed on a multicore machine with the same number of CPUs as there are threads; that is, one computational thread was spawned per CPU.

Figure 8.1 Performance Information

```
                    The HPLOGISTIC Procedure

                    Performance Information

                 Execution Mode       On client
                 Number of Threads    8
```

Figure 8.2 displays the “Model Information” table. The HPLOGISTIC procedure uses a Newton-Raphson algorithm to model a binary distribution for the variable y with a logit link function. The CLASS variable C is parameterized using the GLM parameterization, which is the default.

Figure 8.2 Model Information

```
                         Model Information

       Data Source                 WORK.GETSTARTED
       Response Variable           y
       Class Parameterization      GLM
       Distribution                Binary
       Link Function               Logit
       Optimization Technique      Newton-Raphson with Ridging
```

The CLASS variable C has 10 unique formatted levels, and these are displayed in the “Class Level Information” table in Figure 8.3.

Figure 8.3 Class Level Information

```
                    Class Level Information

              Class    Levels    Values

              C            10    A B C D E F G H I J
```

Figure 8.4 displays the “Number of Observations” table. All 100 observations in the data set are used in the analysis.

Figure 8.4 Number of Observations

```
             Number of Observations Read             100
             Number of Observations Used             100
```

The “Response Profile” table in Figure 8.5 is produced by default for binary and multinomial response variables. It shows the breakdown of the response variable levels by frequency. By default for binary data, the HPLOGISTIC procedure models the probability of the event with the lower-ordered value in the “Response Profile” table—this is indicated by the note that follows the table. In this example, the values represented by y = ‘0’ are modeled as the “successes” in the Bernoulli experiments.

Figure 8.5 Response Profile

```
                    Response Profile

              Ordered             Total
                Value     y    Frequency

                    1     0           69
                    2     1           31

       You are modeling the probability that y='0'.
```

You can use the response-variable options in the MODEL statement to affect which value of the response variable is modeled.

Figure 8.6 displays the “Dimensions” table for this model. This table summarizes some important sizes of various model components. For example, it shows that there are 21 columns in the design matrix **X**, which correspond to one column for the intercept, 10 columns for the effect associated with the classification variable C, and one column each for the continuous variables x1–x10. However, the rank of the crossproducts matrix is only 20. Because the classification variable C uses GLM parameterization and because the model contains an intercept, there is one singularity in the crossproducts matrix of the model. Consequently, only 20 parameters enter the optimization.

Figure 8.6 Dimensions in Binomial Logistic Regression

```
                          Dimensions

          Columns in X                            21
          Number of Effects                       12
          Max Effect Columns                      10
          Rank of Cross-product Matrix            20
          Parameters in Optimization              20
```

The “Iteration History” table is shown in Figure 8.7. The Newton-Raphson algorithm with ridging converged after four iterations, not counting the initial setup iteration.

Figure 8.7 Iteration History

```
                              Iteration History

                                    Objective                          Max
      Iteration     Evaluations      Function         Change      Gradient

              0               4    0.4493546916          .        0.410972
              1               2    0.4436453992     0.00570929    0.081339
              2               2    0.4435038109     0.00014159    0.003302
              3               2    0.4435035933     0.00000022    5.623E-6
              4               2    0.4435035933     0.00000000    1.59E-11
```

Figure 8.8 displays the final convergence status of the Newton-Raphson algorithm. The GCONV= relative convergence criterion is satisfied.

Figure 8.8 Convergence Status

```
                  Convergence criterion (GCONV=1E-8) satisfied.
```

The "Fit Statistics" table is shown in Figure 8.9. The –2 log likelihood at the converged estimates is 88.7007. You can use this value to compare the model to nested model alternatives by means of a likelihood-ratio test. To compare models that are not nested, information criteria such as AIC (Akaike's information criterion), AICC (Akaike's bias-corrected information criterion), and BIC (Schwarz' Bayesian information criterion) are used. These criteria penalize the –2 log likelihood for the number of parameters. Because of the large number of parameters relative to the number of observations, the discrepancy between the –2 log likelihood and, say, AIC, is substantial in this case.

Figure 8.9 Fit Statistics

```
                              Fit Statistics

                    -2 Log Likelihood                 88.7007
                    AIC  (smaller is better)           128.70
                    AICC (smaller is better)           139.33
                    BIC  (smaller is better)           180.80
```

Figure 8.10 shows the global test for the null hypothesis that all model effects jointly do not affect the probability of success of the binary response. The test is significant (*p*-value = 0.0135). One or more of the model effects thus significantly affects the probability of observing an event.

Figure 8.10 Null Test

```
                 Testing Global Null Hypothesis: BETA=0

         Test                    Chi-Square       DF     Pr > ChiSq

         Likelihood Ratio           35.1194       19         0.0135
```

However, a look at the “Parameter Estimates” table in Figure 8.11 shows that many parameters have fairly large *p*-values, indicating that one or more of the model effects might not be necessary.

Figure 8.11 Parameter Estimates

```
                           Parameter Estimates

                                     Standard
      Parameter     Estimate            Error        DF     t Value     Pr > |t|

      Intercept       1.2101           1.7507     Infty        0.69       0.4894
      C A             3.4341           1.6131     Infty        2.13       0.0333
      C B             2.1638           1.4271     Infty        1.52       0.1295
      C C             0.6552           1.0810     Infty        0.61       0.5445
      C D             2.4945           1.1094     Infty        2.25       0.0245
      C E             3.2449           1.4321     Infty        2.27       0.0235
      C F             3.6054           1.3070     Infty        2.76       0.0058
      C G             2.0841           1.1898     Infty        1.75       0.0798
      C H             2.9368           1.2939     Infty        2.27       0.0232
      C I             1.3785           1.0319     Infty        1.34       0.1816
      C J                  0                .         .           .            .
      x1             0.03218          0.05710     Infty        0.56       0.5730
      x2             -0.3677           0.1538     Infty       -2.39       0.0168
      x3              0.3146           0.3574     Infty        0.88       0.3787
      x4            -0.05196          0.02443     Infty       -2.13       0.0334
      x5            -0.00683          0.01056     Infty       -0.65       0.5177
      x6              0.2539           0.3785     Infty        0.67       0.5024
      x7            -0.00723          0.01073     Infty       -0.67       0.5004
      x8              2.5370           0.9942     Infty        2.55       0.0107
      x9             -0.1675           0.1068     Infty       -1.57       0.1168
      x10            -0.2222           0.1577     Infty       -1.41       0.1590
```

Syntax: HPLOGISTIC Procedure

The following statements are available in the HPLOGISTIC procedure:

PROC HPLOGISTIC < *options* > **;**
  CLASS *variable* < *(options)* >. . . < *variable* < *(options)* > > < / *global-options* > **;**
  MODEL *response*< **(***response-options***)** > **=** < *effects* > < / *model-options* > **;**
  MODEL *events/trials*< **(***response-options***)** > **=** < *effects* > < / *model-options* > **;**
  OUTPUT < **OUT=***SAS-data-set* >
    < *keyword* < **=***name* > >. . .
    < *keyword* < **=***name* > > < / *options* > **;**
  PERFORMANCE *performance-options* **;**
  SELECTION *selection-options* **;**
  FREQ *variable* **;**
  ID *variables* **;**
  WEIGHT *variable* **;**

The PROC HPLOGISTIC statement and at least one MODEL statement is required. The CLASS statement can appear multiple times. If a CLASS statement is specified, it must precede the MODEL statements.

PROC HPLOGISTIC Statement

PROC HPLOGISTIC < *options* > **;**

The PROC HPLOGISTIC statement invokes the procedure. Table 8.1 summarizes the available options in the PROC HPLOGISTIC statement by function. The options are then described fully in alphabetical order.

Table 8.1 PROC HPLOGISTIC Statement Options

Option	Description
Basic Options	
ALPHA=	Specifies a global significance level
DATA=	Specifies the input data set
NAMELEN=	Limits the length of effect names
Options Related to Output	
ITDETAILS	Adds detail information to “Iteration History” table
ITSELECT	Displays the “Iteration History” table with model selection
NOPRINT	Suppresses ODS output
NOCLPRINT	Limits or suppresses the display of class levels
NOITPRINT	Suppresses generation of the iteration history table
NOSTDERR	Suppresses computation of covariance matrix and standard errors

Table 8.1 *continued*

Option	Description
Options Related to Optimization	
ABSCONV=	Tunes the absolute function convergence criterion
ABSFCONV=	Tunes the absolute function difference convergence criterion
ABSGCONV=	Tunes the absolute gradient convergence criterion
FCONV=	Tunes the relative function difference convergence criterion
GCONV=	Tunes the relative gradient convergence criterion
MAXITER=	Chooses the maximum number of iterations in any optimization
MAXFUNC=	Specifies the maximum number of function evaluations in any optimization
MAXTIME=	Specifies the upper limit of CPU time (in seconds) for any optimization
MINITER=	Specifies the minimum number of iterations in any optimization
NORMALIZE=	Specifies whether the objective function is normalized during optimization
TECHNIQUE=	Selects the optimization technique
Tolerances	
SINGCHOL=	Tunes the singularity criterion for Cholesky decompositions
SINGSWEEP=	Tunes the singularity criterion for the sweep operator
SINGULAR=	Tunes the general singularity criterion
User-Defined Formats	
FMTLIBXML=	Specifies the file reference for a format stream

You can specify the following options in the PROC HPLOGISTIC statement.

ABSCONV=*r*

ABSTOL=*r*

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

ABSFCONV=*r* <*n*>

ABSFTOL=*r*<*n*>

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

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

Here, ψ denotes the vector of parameters that participate in the optimization, and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\psi(k)$ is defined as the vertex with the lowest function value and $\psi^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is $r = 0$. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

ABSGCONV=*r* < *n* >

ABSGTOL=*r*< *n* >

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

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

Here, $\boldsymbol{\psi}$ denotes the vector of parameters that participate in the optimization, and $g_j(\cdot)$ is the gradient of the objective function with respect to the jth parameter. This criterion is not used by the NMSIMP technique. The default value is *r*=1E–5. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

ALPHA=*number*

specifies a global significance level for the construction of confidence intervals. The confidence level is 1–*number*. The value of *number* must be between 0 and 1; the default is 0.05. You can override the global specification with the ALPHA= option in the MODEL statement.

DATA=*SAS-data-set*

names the input SAS data set for PROC HPLOGISTIC to use. The default is the most recently created data set.

If the procedure executes in MPP mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. In that case the procedure reads the data alongside the distributed database. For information about the various execution modes, see the section "SMP and MPP Modes" on page 10; for information about the alongside-the-database model, see the section "Alongside-the-Database Execution" on page 15.

FCONV=*r*< *n* >

FTOL=*r*< *n* >

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

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

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

The default value is *r*=2 $\times\ \epsilon$ where ϵ is the machine precision. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

FMTLIBXML=*file-ref*

specifies the file reference for the XML stream that contains the user-defined format definitions. User-defined formats are handled differently in a distributed computing environment than they are in other SAS products. See the section "Working with Formats" on page 18 in Chapter 2, "Shared Concepts and Topics," for details about how to generate a XML stream for your formats.

GCONV=*r< n>*

GTOL=*r< n>*

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

$$\frac{\mathbf{g}(\boldsymbol{\psi}^{(k)})'[\mathbf{H}^{(k)}]^{-1}\mathbf{g}(\boldsymbol{\psi}^{(k)})}{|f(\boldsymbol{\psi}^{(k)})|} \leq r$$

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

$$\frac{\parallel \mathbf{g}(\boldsymbol{\psi}^{(k)}) \parallel_2^2 \quad \parallel \mathbf{s}(\boldsymbol{\psi}^{(k)}) \parallel_2}{\parallel \mathbf{g}(\boldsymbol{\psi}^{(k)}) - \mathbf{g}(\boldsymbol{\psi}^{(k-1)}) \parallel_2 |f(\boldsymbol{\psi}^{(k)})|} \leq r$$

This criterion is not used by the NMSIMP technique. The default value is *r*=1E–8. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

ITDETAILS

adds to the "Iteration History" table the current values of the parameter estimates and their gradients. These quantities are reported only for parameters that participate in the optimization. The ITDETAILS option is not available with model selection.

ITSELECT

generates the "Iteration History" table when you perform a model selection.

MAXFUNC=*n*

MAXFU=*n*

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

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

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

MAXITER=*n*

MAXIT=*n*

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

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

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

MAXTIME=*r*

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

MINITER=*n*

MINIT=*n*

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

NAMELEN=*number*

specifies the length to which long effect names are shortened. The default and minimum value is 20.

NOCLPRINT< =*number* >

suppresses the display of the "Class Level Information" table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps to reduce the size of the "Class Level Information" table if some classification variables have a large number of levels.

NOITPRINT

suppresses the generation of the "Iteration History" table.

NOPRINT

suppresses the generation of ODS output.

NORMALIZE=YES | NO

specifies whether the objective function should be normalized during the optimization by the reciprocal of the used frequency count. The default is to normalize the objective function. This option affects the values reported in the "Iteration History" table. The results reported in the "Fit Statistics" are always displayed for the nonnormalized log-likelihood function.

NOSTDERR

suppresses the computation of the covariance matrix and the standard errors of the logistic regression coefficients. When the model contains many variables (thousands), the inversion of the Hessian matrix to derive the covariance matrix and the standard errors of the regression coefficients can be time-consuming.

SINGCHOL=*number*

tunes the singularity criterion in Cholesky decompositions. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

SINGSWEEP=*number*

tunes the singularity criterion for sweep operations. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

SINGULAR=*number*

tunes the general singularity criterion applied by the HPLOGISTIC procedure in sweeps and inversions. The default is 1E4 times the machine epsilon; this product is approximately 1E–12 on most computers.

TECHNIQUE=*keyword*

TECH=*keyword*

specifies the optimization technique for obtaining maximum likelihood estimates. You can choose from the following techniques by specifying the appropriate *keyword*:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NEWRAP	performs a Newton-Raphson optimization with line search.
NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	performs no optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a dual quasi-Newton optimization.
TRUREG	performs a trust-region optimization

The default value is TECHNIQUE=NRRIDG.

For more information, see the section “Choosing an Optimization Algorithm” on page 206.

CLASS Statement

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

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. The CLASS statement must precede the MODEL statement. You can list the response variable for binary and multinomial models in the CLASS statement, but this is not necessary.

The CLASS statement for High-Performance Analytics procedures is documented in the section "CLASS Statement" on page 20 of Chapter 2, "Shared Concepts and Topics."

The HPLOGISTIC procedure does not support the SPLIT option in the CLASS statement. The HPLOGISTIC procedure additionally supports the following global-option in the CLASS statement:

UPCASE

uppercases the values of character-valued CLASS variables before levelizing them. For example, if the UPCASE option is in effect and a CLASS variable can take the values 'a', 'A', and 'b', then 'a' and 'A' represent the same level and the CLASS variable is treated as having only two values: 'A' and 'B'.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. High-Performance Analytics procedures that support the FREQ statement treat each observation as if it appeared f times, where the frequency value f is the value of the FREQ variable for the observation. If f is not an integer, then f is truncated to an integer. If f is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID *variables* **;**

The ID statement lists one or more variables from the input data set that are to be transferred to output data sets created by High-Performance Analytics procedures, provided that the output data set produces one (or more) records per input observation.

For documentation about the common ID statement in High-Performance Analytics procedures, see the section "ID Statement" on page 23 in Chapter 2, "Shared Concepts and Topics."

MODEL Statement

MODEL *response* < **(***response-options***)** > **=** < *effects* > < / *model-options* > **;**

MODEL *events* / *trials* < **(***response-options***)** > **=** < *effects* > < / *model-options* > **;**

The MODEL statement defines the statistical model in terms of a response variable (the target) or an *events/trials* specification, model effects constructed from variables in the input data set, and options. An intercept is included in the model by default. You can remove the intercept with the NOINT option.

You can specify a single *response* variable that contains your binary, ordinal, or nominal response values. When you have binomial data, you can specify the *events/trials* form of the response, where one variable contains the number of positive responses (or events) and another variable contains the number of trials. Note that the values of both *events* and (*trials* – *events*) must be nonnegative and the value of *trials* must be positive.

For information about constructing the model effects, see the section "Specification and Parameterization of Model Effects" on page 35 of Chapter 2, "Shared Concepts and Topics."

There are two sets of options in the MODEL statement. The *response-options* determine how the HPLOGISTIC procedure models probabilities for binary data. The *model-options* control other aspects of model formation and inference. Table 8.2 summarizes these options.

Table 8.2 MODEL Statement Options

Option	**Description**
Response Variable Options	
DESCENDING	Reverses the response categories
EVENT=	Specifies the event category
ORDER=	Specifies the sort order
REF=	Specifies the reference category
Model Options	
ALPHA=	Specifies the confidence level for confidence limits
CL	Requests confidence limits
DDFM=	Specifies the degrees-of-freedom method
LACKFIT	Requests the Hosmer and Lemeshow goodness-of-fit test
LINK=	Specifies the link function
NOINT	Suppresses the intercept
OFFSET=	Specifies the offset variable
RSQUARE	Requests a generalized coefficient of determination

Response Variable Options

Response variable options determine how the HPLOGISTIC procedure models probabilities for binary and multinomial data.

You can specify the following *response-options* by enclosing them in parentheses after the *response* or *trials* variable.

DESCENDING

DESC

reverses the order of the response categories. If both the DESCENDING and ORDER= options are specified, PROC HPLOGISTIC orders the response categories according to the ORDER= option and then reverses that order.

EVENT=’*category*’ | FIRST | LAST

specifies the event category for the binary response model. PROC HPLOGISTIC models the probability of the event category. The EVENT= option has no effect when there are more than two response categories.

You can specify the value (formatted, if a format is applied) of the event category in quotes, or you can specify one of the following:

FIRST

designates the first ordered category as the event. This is the default.

LAST

designates the last ordered category as the event.

For example, the following statements specify that observations with formatted value ‘1’ represent events in the data. The probability modeled by the HPLOGISTIC procedure is thus the probability that the variable def takes on the (formatted) value ‘1’.

```
proc hplogistic data=MyData;
   class A B C;
   model def(event ='1') = A B C x1 x2 x3;
run;
```

ORDER=DATA | FORMATTED | INTERNAL

ORDER=FREQ | FREQDATA | FREQFORMATTED | FREQINTERNAL

specifies the sort order for the levels of the *response* variable. When ORDER=FORMATTED (the default) for numeric variables for which you have supplied no explicit format (that is, for which there is no corresponding FORMAT statement in the current PROC HPLOGISTIC run or in the DATA step that created the data set), the levels are ordered by their internal (numeric) value. The following table shows the interpretation of the ORDER= option:

ORDER=	Levels Sorted By
DATA	Order of appearance in the input data set
FORMATTED	External formatted value, except for numeric variables with no explicit format, which are sorted by their unformatted (internal) value
FREQ	Descending frequency count (levels with the most observations come first in the order)

Table 8.2 *continued*

ORDER=	Levels Sorted By
FREQDATA	Order of descending frequency count; within counts by order of appearance in the input data set when counts are tied
FREQFORMATTED	Order of descending frequency count; within counts by formatted value (as above) when counts are tied
FREQINTERNAL	Order of descending frequency count; within counts by unformatted value when counts are tied
INTERNAL	Unformatted value

By default, ORDER=FORMATTED. For the FORMATTED and INTERNAL orders, the sort order is machine-dependent.

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

REF='*category*' | FIRST | LAST

specifies the reference category for the generalized logit model and the binary response model. For the generalized logit model, each logit contrasts a nonreference category with the reference category. For the binary response model, specifying one response category as the reference is the same as specifying the other response category as the event category. You can specify the value (formatted if a format is applied) of the reference category in quotes, or you can specify one of the following:

FIRST

designates the first ordered category as the reference

LAST

designates the last ordered category as the reference. This is the default.

Model Options

ALPHA=*number*

requests that confidence intervals for each of the parameters be constructed with confidence level 1–*number*. The value of *number* must be between 0 and 1; the default is 0.05.

CL

requests that confidence limits be constructed for each of the parameter estimates. The confidence level is 0.95 by default; this can be changed with the ALPHA= option.

DDFM=RESIDUAL | NONE

specifies how degrees of freedom for statistical inference be determined in the "Parameter Estimates Table."

The HPLOGISTIC procedure always displays the statistical tests and confidence intervals in the "Parameter Estimates" tables in terms of a t test and a two-sided probability from a t distribution. With the DDFM= option, you can control the degrees of freedom of this t distribution and thereby switch between small-sample inference and large-sample inference based on the normal or chi-square distribution.

The default is DDFM=NONE, which leads to z-based statistical tests and confidence intervals. The HPLOGISTIC procedure then displays the degrees of freedom in the DF column as Infty, the p-values are identical to those from a Wald chi-square test, and the square of the t value equals the Wald chi-square statistic.

If you specify DDFM=RESIDUAL, the degrees of freedom are finite and determined by the number of usable frequencies (observations) minus the number of nonredundant model parameters. This leads to t-based statistical tests and confidence intervals. If the number of frequencies is large relative to the number of parameters, the inferences from the two degrees-of-freedom methods are almost identical.

LACKFIT< (DFREDUCE=*r* NGROUPS=*G*) >

performs the Hosmer and Lemeshow goodness-of-fit test (Hosmer and Lemeshow 2000) for binary response models.

The subjects are divided into at most G groups of roughly the same size, based on the percentiles of the estimated probabilities. You can specify G as any integer greater than or equal to 5; by default, G=10. Let the actual number of groups created be g. The discrepancies between the observed and expected number of observations in these g groups are summarized by the Pearson chi-square statistic, which is then compared to a chi-square distribution with g–r degrees of freedom. You can specify a nonnegative integer r that satisfies $g–r \geq 1$; by default, r=2.

A small p-value suggests that the fitted model is not an adequate model. See the section "The Hosmer-Lemeshow Goodness-of-Fit Test" on page 204 for more information.

LINK=*keyword*

specifies the link function for the model. The ***keywords*** and the associated link functions are shown in Table 8.3.

Table 8.3 Built-in Link Functions of the HPLOGISTIC Procedure

LINK=	Link Function	$g(\mu) = \eta =$
CLOGLOG | CLL	Complementary log-log	$\log(-\log(1-\mu))$
GLOGIT | GENLOGIT	Generalized logit	
LOGIT	Logit	$\log(\mu/(1-\mu))$
LOGLOG	Log-log	$-\log(-\log(\mu))$
PROBIT	Probit	$\Phi^{-1}(\mu)$

For the probit and cumulative probit links, $\Phi^{-1}(\cdot)$ denotes the quantile function of the standard normal distribution.

If the response variable has more than two categories, the HPLOGISTIC procedure fits a model with a cumulative link function based on the specified link. However, if you specify LINK=GLOGIT, the procedure assumes a generalized logit model for nominal (unordered) data, regardless of the number of response categories.

NOINT

requests that no intercept be included in the model. An intercept is included by default. The NOINT option is not available in multinomial models.

OFFSET=*variable*

specifies a *variable* to be used as an offset to the linear predictor. An offset plays the role of an effect whose coefficient is known to be 1. The offset variable cannot appear in the CLASS statement or elsewhere in the MODEL statement. Observations with missing values for the offset variable are excluded from the analysis.

RSQUARE

R2

requests a generalized coefficient of determination (R square, R^2) and a scaled version thereof for the fitted model. The results are added to the "Fit Statistics" table. For more information about the computation of these measures, see the section "Generalized Coefficient of Determination" on page 203.

OUTPUT Statement

OUTPUT < **OUT=***SAS-data-set*>
< keyword < =name > >. . . < keyword < =name > > < / options > **;**

The OUTPUT statement creates a data set that contains observationwise statistics that are computed after fitting the model. The variables in the input data set are *not* included in the output data set to avoid data duplication for large data sets; however, variables specified in the ID statement are included.

If the input data are in distributed form, where access of data in a particular order cannot be guaranteed, the HPLOGISTIC procedure copies the distribution or partition key to the output data set so that its contents can be joined with the input data.

The output statistics are computed based on the final parameter estimates. If the model fit does not converge, missing values are produced for the quantities that depend on the estimates.

When there are more than two response levels, only variables named by the XBETA and PREDICTED keywords have their values computed; the other variables have missing values. These statistics are computed for every response category, and the automatic variable _LEVEL_ identifies the response category upon which the computed values are based. If you also specify the OBSCAT option, then the observationwise statistics are computed only for the observed response category, as indicated by the value of the _LEVEL_ variable.

For observations in which only the response variable is missing, values of the XBETA and PREDICTED statistics are computed even though these observations do not affect the model fit. This enables, for instance, predicted probabilities to be computed for new observations.

You can specify the following syntax elements in the OUTPUT statement before the slash (/).

OUT=*SAS-data-set*

DATA=*SAS-data-set*

specifies the name of the output data set. If the OUT= (or DATA=) option is omitted, the procedure uses the DATAn convention to name the output data set.

keyword **< =***name***>**

specifies a statistic to include in the output data set and optionally names the variable *name*. If you do not provide a *name*, the HPLOGISTIC procedure assigns a default name based on the type of statistic requested.

The following are valid *keywords* for adding statistics to the OUTPUT data set:

LINP | XBETA

requests the linear predictor $\eta = \mathbf{x}'\boldsymbol{\beta}$.

PREDICTED | PRED | P

requests predicted values (predicted probabilities of events) for the response variable.

RESIDUAL | RESID | R

requests the raw residual, $y - \mu$, where μ is the estimate of the predicted event probability. This statistic is not computed for multinomial models.

PEARSON | PEARS | RESCHI

requests the Pearson residual, $\frac{\sqrt{wn}(y/n-\mu)}{\sqrt{\mu(1-\mu)}}$, where μ is the estimate of the predicted event probability, w is the weight of the observation, and n is the number of binomial trials (n=1 for binary observations). This statistic is not computed for multinomial models.

You can specify the following *option* in the OUTPUT statement after the slash (/):

OBSCAT

requests (for multinomial models) that observationwise statistics be produced for the response level only. If the OBSCAT option is not specified and the response variable has J levels, then the following outputs are created: for cumulative link models, $J - 1$ records are output for every observation in the input data that corresponds to the $J - 1$ lower-ordered response categories; for generalized logit models, J records are output that correspond to all J response categories.

PERFORMANCE Statement

PERFORMANCE < *performance-options*> **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of the HPLOGISTIC procedure.

With the PERFORMANCE statement you can also control whether the HPLOGISTIC procedure executes in SMP or MPP mode.

The PERFORMANCE statement for High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

SELECTION Statement

SELECTION < *options* > **;**

The SELECTION statement performs model selection by examining whether effects should be added to or removed from the model according to rules defined by model selection methods. The statement is fully documented in the section "SELECTION Statement" on page 24 in Chapter 2, "Shared Concepts and Topics."

The HPLOGISTIC procedure supports the following effect-selection methods in the SELECTION statement:

METHOD=NONE	results in no model selection. This method fits the full model.
METHOD=FORWARD	performs forward selection. This method starts with no effects in the model and adds effects.
METHOD=BACKWARD	performs backward elimination. This method starts with all effects in the model and deletes effects.
METHOD=BACKWARD(FAST)	performs fast backward elimination. This method starts with all effects in the model and deletes effects without refitting the model.
METHOD=STEPWISE	performs stepwise regression. This method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.

The only effect-selection criterion supported by the HPLOGISTIC procedure is SELECT=SL, where effects enter and leave the model based on an evaluation of the significance level. To determine this level of significance for each candidate effect, the HPLOGISTIC procedure calculates an approximate chi-square score test statistic.

The criteria available for the CHOOSE= and STOP= options in the SELECT statement are

SL	the significance level of the score test
AIC	Akaike's information criterion (Akaike 1974)
AICC	a small-sample bias corrected version of Akaike's information criterion as promoted in, for example, Hurvich and Tsai (1989) and Burnham and Anderson (1998)
BIC | SBC	Schwarz' Bayesian criterion (Schwarz 1978)

The calculation of the information criteria uses the following formulas, where p denotes the number of effective parameters in the candidate model, f denotes the number of frequencies used, and l is the log likelihood evaluated at the converged estimates:

$$\text{AIC} = -2l + 2p$$

$$\text{AICC} = \begin{cases} -2l + 2pf/(f-p-1) & \text{when } f > p+2 \\ -2l + 2p(p+2) & \text{otherwise} \end{cases}$$

$$\text{BIC} = -2l + p\log(f)$$

NOTE: If you use the fast backward elimination method, the –2 log likelihood, AIC, AICC, and BIC statistics are approximated at each step where the model is not refit, and hence do not match the values computed when that model is fit outside of the selection routine.

When you specify the DETAILS= option in the SELECTION statement, the HPLOGISTIC procedure produces the following:

DETAILS=SUMMARY	produces a summary table that shows the effect added or removed at each step along with the *p*-value. The summary table is produced by default if the DETAILS= option is not specified.
DETAILS=STEPS	produces a detailed listing of all candidates at each step and their ranking in terms of the significance level for entry into or removal from the model.
DETAILS=ALL	produces the preceding two tables and a table of selection details which displays fit statistics for the model at each step of the selection process and the approximate chi-square score statistic.

WEIGHT Statement

WEIGHT *variable* **;**

The *variable* in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations with nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, then all observations used in the analysis are assigned a weight of 1.

Details: HPLOGISTIC Procedure

Missing Values

Any observation with missing values for the response, frequency, weight, offset, or explanatory variables is excluded from the analysis; however, missing values are valid for response and explanatory variables that are specified with the MISSING option in the CLASS statement. Observations with a nonpositive weight or with a frequency less than 1 are also excluded.

The estimated linear predictor and the fitted probabilities are not computed for any observation that has missing offset or explanatory variable values. However, if only the response value is missing, the linear predictor and the fitted probabilities can be computed and output to a data set by using the OUTPUT statement.

Response Distributions

The response distribution is the probability distribution of the response (target) variable. The HPLOGISTIC procedure can fit data for the following distributions:

- binary distribution
- binomial distribution
- multinomial distribution

The expressions for the log-likelihood functions of these distributions are given in the next section.

The binary (or Bernoulli) distribution is the elementary distribution of a discrete random variable that can take on two values with probabilities p and $1 - p$. Suppose the random variable is denoted Y and

$$\Pr(Y = 1) = p$$
$$\Pr(Y = 0) = 1 - p$$

The value associated with probability p is often termed the *event* or “success”; the complementary event is termed the *non-event* or “failure.” A Bernoulli experiment is a random draw from a binary distribution and generates events with probability p.

If $Y_1, \cdots, Y_n$ are n independent Bernoulli random variables, then their sum follows a binomial distribution. In other words, if $Y_i = 1$ denotes an event (success) in the ith Bernoulli trial, a binomial random variable is the number of events (successes) in n independent Bernoulli trials. If you use the events/trials syntax in the MODEL statement, the HPLOGISTIC procedure fits the model as if the data had arisen from a binomial distribution. For example, the following statements fit a binomial regression model with regressors x1 and x2. The variables e and t represent the events and trials for the binomial distribution:

```
proc hplogistic;
   model e/t = x1 x2;
run;
```

If the events/trials syntax is used, then both variables must be numeric and the value of the events variable cannot be less than 0 or exceed the value of the trials variable. A “Response Profile” table is not produced for binomial data, since the response variable is not subject to levelization.

The multinomial distribution is a generalization of the binary distribution and allows for more than two outcome categories. Because there are more than two possible outcomes for the multinomial distribution, the terminology of “successes,” “failures,” “events,” and “non-events” no longer applies. With multinomial data, these outcomes are generically referred to as “categories” or levels.

Whenever the HPLOGISTIC procedure determines that the response variable has more than two levels (unless the events/trials syntax is used), the procedure fits the model as if the data had arisen from a multinomial distribution. By default, it is then assumed that the response categories are ordered and a cumulative link model is fit by applying the default or specified link function. If the response categories are unordered, then you should fit a generalized logit model by choosing LINK=GLOGIT in the MODEL statement.

Log-Likelihood Functions

The HPLOGISTIC procedure forms the log-likelihood functions of the various models as

$$L(\boldsymbol{\mu};\mathbf{y}) = \sum_{i=1}^{n} f_i \, l(\mu_i; y_i, w_i)$$

where $l(\mu_i; y_i, w_i)$ is the log-likelihood contribution of the ith observation with weight w_i and f_i is the value of the frequency variable. For the determination of w_i and f_i, see the WEIGHT and FREQ statements. The individual log-likelihood contributions for the various distributions are as follows.

Binary Distribution

The HPLOGISTIC procedure computes the log-likelihood function $l(\mu_i(\boldsymbol{\beta}); y_i)$ for the ith binary observation as

$$\begin{aligned}
\eta_i &= \mathbf{x}_i'\boldsymbol{\beta} \\
\mu_i(\boldsymbol{\beta}) &= g^{-1}(\eta_i) \\
l(\mu_i(\boldsymbol{\beta}); y_i) &= y_i \log\{\mu_i\} + (1-y_i)\log\{1-\mu_i\}
\end{aligned}$$

Here, μ_i is the probability of an event, and the variable y_i takes on the value 1 for an event and the value 0 for a non-event. The inverse link function $g^{-1}(\cdot)$ maps from the scale of the linear predictor η_i to the scale of the mean. For example, for the logit link (the default),

$$\mu_i(\boldsymbol{\beta}) = \frac{\exp\{\eta_i\}}{1+\exp\{\eta_i\}}$$

You can control which binary outcome in your data is modeled as the event with the *response-options* in the MODEL statement, and you can choose the link function with the LINK= option in the MODEL statement.

If a WEIGHT statement is given and w_i denotes the weight for the current observation, the log-likelihood function is computed as

$$l(\mu_i(\boldsymbol{\beta}); y_i, w_i) = w_i l(\mu_i(\boldsymbol{\beta}); y_i)$$

Binomial Distribution

The HPLOGISTIC procedure computes the log-likelihood function $l(\mu_i(\boldsymbol{\beta}); y_i)$ for the ith binomial observation as

$$\begin{aligned}
\eta_i &= \mathbf{x}_i'\boldsymbol{\beta} \\
\mu_i(\boldsymbol{\beta}) &= g^{-1}(\eta_i) \\
l(\mu_i(\boldsymbol{\beta}); y_i, w_i) &= w_i\,(y_i \log\{\mu_i\} + (n_i-y_i)\log\{1-\mu_i\}) \\
&\quad + w_i\,(\log\{\Gamma(n_i+1)\} - \log\{\Gamma(y_i+1)\} - \log\{\Gamma(n_i-y_i+1)\})
\end{aligned}$$

where y_i and n_i are the values of the events and trials of the ith observation, respectively. μ_i measures the probability of events (successes) in the underlying Bernoulli distribution whose aggregate follows the binomial distribution.

Multinomial Distribution

The multinomial distribution modeled by the HPLOGISTIC procedure is a generalization of the binary distribution; it is the distribution of a single draw from a discrete distribution with J possible values. The log-likelihood function for the ith observation is thus deceptively simple:

$$l(\boldsymbol{\mu}_i; \mathbf{y}_i, w_i) = w_i \sum_{j=1}^{J} y_{ij} \log\{\mu_{ij}\}$$

In this expression, J denotes the number of response categories (the number of possible outcomes) and μ_{ij} is the probability that the ith observation takes on the response value associated with category j. The category probabilities must satisfy

$$\sum_{j=1}^{J} \mu_j = 1$$

and the constraint is satisfied by modeling $J - 1$ categories. In models with ordered response categories, the probabilities are expressed in cumulative form, so that the last category is redundant. In generalized logit models (multinomial models with unordered categories), one category is chosen as the reference category and the linear predictor in the reference category is set to zero.

Generalized Coefficient of Determination

The goal of a coefficient of determination, also known as an R-square measure, is to express the agreement between a stipulated model and the data in terms of variation in the data explained by the model. In linear models, the R-square measure is based on residual sums of squares; because these are additive, a measure bounded between 0 and 1 is easily derived.

In more general models where parameters are estimated by the maximum likelihood principle, Cox and Snell (1989, pp. 208–209) and Magee (1990) proposed the following generalization of the coefficient of determination:

$$R^2 = 1 - \left\{ \frac{L(\mathbf{0})}{L(\widehat{\boldsymbol{\beta}})} \right\}^{\frac{2}{n}}$$

Here, $L(\mathbf{0})$ is the likelihood of the intercept-only model, $L(\widehat{\boldsymbol{\beta}})$ is the likelihood of the specified model, and n denotes the number of observations used in the analysis. This number is adjusted for frequencies if a FREQ statement is present and is based on the trials variable for binomial models.

As discussed in Nagelkerke (1991), this generalized R-square measure has properties similar to the coefficient of determination in linear models. If the model effects do not contribute to the analysis, $L(\widehat{\boldsymbol{\beta}})$ approaches $L(\mathbf{0})$ and R^2 approaches zero.

However, R^2 does not have an upper limit of 1. Nagelkerke suggested a rescaled generalized coefficient of determination that achieves an upper limit of 1, by dividing R^2 by its maximum value,

$$R^2_{\text{max}} = 1 - \{L(\mathbf{0})\}^{\frac{2}{n}}$$

If you specify the RSQUARE option in the MODEL statement, the HPLOGISTIC procedure computes R^2 and the rescaled coefficient of determination according to Nagelkerke:

$$\tilde{R}^2 = \frac{R^2}{R^2_{\max}}$$

The R^2 and $\tilde{R}^2$ measures are most useful for comparing competing models that are not necessarily nested—that is, models that cannot be reduced to one another by simple constraints on the parameter space. Larger values of the measures indicate better models.

The Hosmer-Lemeshow Goodness-of-Fit Test

To evaluate the fit of the model, Hosmer and Lemeshow (2000) proposed a statistic that they show, through simulation, is distributed as chi-square when there is no replication in any of the subpopulations. This goodness-of-fit test is available only for binary response models.

The unit interval is partitioned into 2,000 equal-sized bins, and each observation i is placed into the bin that contains its estimated event probability. This effectively sorts the observations in increasing order of their estimated event probability.

The observations (and frequencies) are further combined into G groups. By default G=10, but you can specify $G \geq 5$ with the NGROUPS= suboption of the LACKFIT option in the MODEL statement. Let F be the total frequency. The target frequency for each group is $T = \lfloor F/G + 0.5 \rfloor$, which is the integer part of $F/G + 0.5$. Load the first group ($g_j, j = 1$) with the first of the 2,000 bins that has nonzero frequency f_1, and let the next nonzero bin have a frequency of f. PROC HPLOGISTIC performs the following steps for each nonzero bin to create the groups:

1. If $j = G$, then add this bin to group g_j.
2. Otherwise, if $f_j < T$ and $f_j + \lfloor f/2 \rfloor \leq T$, then add this bin to group g_j.
3. Otherwise, start loading the next group (g_{j+1}) with $f_{j+1} = f$, and set $j = j + 1$.

If the final group g_j has frequency $f_j < T/2$, then add these observations to the preceding group. The total number of groups actually created, g, can be less than G.

The Hosmer-Lemeshow goodness-of-fit statistic is obtained by calculating the Pearson chi-square statistic from the $2 \times g$ table of observed and expected frequencies. The statistic is written

$$\chi^2_{HL} = \sum_{j=1}^{g} \frac{(O_j - F_j \bar{\pi}_j)^2}{F_j \bar{\pi}_j (1 - \bar{\pi}_j)}$$

where, for the jth group g_j, $F_j = \sum_{i \in g_j} f_i$ is the total frequency of subjects, O_j is the total frequency of event outcomes, and $\bar{\pi}_j = \sum_{i \in g_j} f_i \widehat{p}_i / F_j$ is the average estimated predicted probability of an event outcome. Let ϵ be the square root of the machine epsilon divided by 4,000, which is about 2.5E–12. Any $\bar{\pi}_j < \epsilon$ is set to ϵ; similarly, any $\bar{\pi}_j > 1 - \epsilon$ is set to $1 - \epsilon$.

The Hosmer-Lemeshow statistic is compared to a chi-square distribution with $g - r$ degrees of freedom. You can specify r with the DFREDUCE= suboption of the LACKFIT option in the MODEL statement. By default, $r = 2$, and to compute the Hosmer-Lemeshow statistic you must have $g - r \geq 1$. Large values of χ^2_{HL} (and small p-values) indicate a lack of fit of the model.

Computational Method: Multithreading

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

The number of threads spawned by the HPLOGISTIC procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statement, the HPLOGISTIC procedure determines threading as if it executed on a system with four CPUs, regardless of the actual CPU count:

  ```
  options cpucount=4;
  ```

- You can specify the NTHREADS= option in the PERFORMANCE statement to control the number of threads. This specification overrides the system option. Specify NTHREADS=1 to force single-threaded execution.

- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement, which overrides the CPUCOUNT system option and instructs the HPLOGISTIC procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Dimensions" table, which is part of the default output. The HPLOGISTIC procedure allocates one thread per CPU by default.

The tasks that are multithreaded by the HPLOGISTIC procedure are primarily defined by dividing the data processed on a single machine among the threads—that is, the HPLOGISTIC procedure implements multithreading through a data-parallel model. For example, if the input data set has 1,000 observations and you are running with four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following:

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

- gradient calculation
- Hessian calculation
- scoring of observations

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

Choosing an Optimization Algorithm

First- or Second-Order Algorithms

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

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

Table 8.4 shows which derivatives are required for each optimization technique.

Table 8.4 Derivatives Required

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

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

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

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

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

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

By default, the HPLOGISTIC procedure applies the NRRIDG algorithm because it can take advantage of multithreading in Hessian computations and inversions. If the number of parameters becomes large, specifying the TECHNIQUE=QUANEW option, which is a first-order method with good overall properties, is recommended.

Algorithm Descriptions

The following subsections provide details about each optimization technique and follow the same order as Table 8.4.

Trust Region Optimization (TRUREG)

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

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

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

Newton-Raphson Optimization with Line Search (NEWRAP)

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

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

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

Newton-Raphson Ridge Optimization (NRRIDG)

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

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

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

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

Quasi-Newton Optimization (QUANEW)

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

The QUANEW technique implemented by the HPLOGISTIC procedure is the dual quasi-Newton algorithm, which updates the Cholesky factor of an approximate Hessian.

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

Double-Dogleg Optimization (DBLDOG)

The double-dogleg optimization method combines the ideas of the quasi-Newton and trust region methods. In each iteration, the double-dogleg algorithm computes the step $\mathbf{s}^{(k)}$ as the linear combination of the steepest descent or ascent search direction $\mathbf{s}_1^{(k)}$ and a quasi-Newton search direction $\mathbf{s}_2^{(k)}$:

$$\mathbf{s}^{(k)} = \alpha_1 \mathbf{s}_1^{(k)} + \alpha_2 \mathbf{s}_2^{(k)}$$

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

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

Conjugate Gradient Optimization (CONGRA)

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

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

Nelder-Mead Simplex Optimization (NMSIMP)

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

The original Nelder-Mead simplex algorithm is implemented and extended to boundary constraints. This algorithm does not compute the objective for infeasible points, but it changes the shape of the simplex adapting to the nonlinearities of the objective function. This change contributes to an increased speed of convergence and uses a special termination criterion.

Displayed Output

The following sections describe the output that PROC HPLOGISTIC produces by default. The output is organized into various tables, which are discussed in the order of appearance.

Performance Information

The “Performance Information” table is produced by default. It displays information about the grid host for distributed execution, whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also displayed, depending on the environment.

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a “Timing” table in which elapsed time (absolute and relative) for the main tasks of the procedure are displayed.

Model Information

The “Model Information” table displays basic information about the model, such as the response variable, frequency variable, link function, and the model category the HPLOGISTIC procedure determined based on your input and options. The “Model Information” table also displays the distribution of the data that is assumed by the HPLOGISTIC procedure. See the section “Response Distributions” on page 201 for how the procedure determines the response distribution.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the CLASS statement. You should check this information to make sure that the data are correct. You can adjust the order of the CLASS variable levels with the ORDER= option in the CLASS statement. You can suppress the “Class Level Information” table completely or partially with the NOCLPRINT= option in the PROC HPLOGISTIC statement.

If the classification variables use reference parameterization, the “Class Level Information” table also displays the reference value for each variable.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data set and the number of observations used in the analysis. If a FREQ statement is present, the sum of the frequencies read and used is displayed. If the events/trials syntax is used, the number of events and trials is also displayed.

Response Profile

The “Response Profile” table displays the ordered value from which the HPLOGISTIC procedure determines the probability being modeled as an event in binary models and the ordering of categories in multinomial models. For each response category level, the frequency used in the analysis is reported. You can affect the ordering of the response values with the *response-options* in the MODEL statement. For binary and generalized logit models, the note that follows the “Response Profile” table indicates which outcome is modeled as the event in binary models and which value serves as the reference category.

The “Response Profile” table is not produced for binomial data. You can find information about the number of events and trials in the “Number of Observations” table.

Selection Information

When you specify the SELECTION statement, the HPLOGISTIC procedure produces by default a series of tables with information about the model selection. The “Selection Information” table informs you about the model selection method, selection and stop criteria, and other parameters that govern the selection. You can suppress this table by specifying DETAILS=NONE in the SELECTION statement.

Selection Summary

When you specify the SELECTION statement, the HPLOGISTIC procedure produces the “Selection Summary” table with information about which effects were entered into or removed from the model at the steps of the model selection process. The p-value for the score chi-square test that led to the removal or entry decision is also displayed. You can request further details about the model selection steps by specifying DETAILS=STEPS or DETAILS=ALL in the SELECTION statement. You can suppress the display of the “Selection Summary” table by specifying DETAILS=NONE in the SELECTION statement.

Stop Reason

When you specify the SELECTION statement, the HPLOGISTIC procedure produces a simple table that tells you why model selection stopped.

Selection Reason

When you specify the SELECTION statement, the HPLOGISTIC procedure produces a simple table that tells you why the final model was selected.

Selected Effects

When you specify the SELECTION statement, the HPLOGISTIC procedure produces a simple table that tells you which effects were selected into the final model.

Iteration History

For each iteration of the optimization, the "Iteration History" table displays the number of function evaluations (including gradient and Hessian evaluations), the value of the objective function, the change in the objective function from the previous iteration and the absolute value of the largest (projected) gradient element. The objective function used in the optimization in the HPLOGISTIC procedure is normalized by default to enable comparisons across data sets with different sampling intensity. You can control normalization with the NORMALIZE= option in the PROC HPLOGISTIC statement.

If you specify the ITDETAILS option in the PROC HPLOGISTIC statement, information about the parameter estimates and gradients in the course of the optimization is added to the "Iteration History" table.

The "Iteration History" table is displayed by default unless you specify the NOITPRINT option or perform a model selection. To generate the history from a model selection process, specify the ITSELECT option.

Convergence Status

The convergence status table is a small ODS table that follows the "Iteration History" table in the default output. In the listing it appears as a message that indicates whether the optimization succeeded and which convergence criterion was met. If the optimization fails, the message indicates the reason for the failure. If you save the convergence status table to an output data set, a numeric Status variable is added that enables you to assess convergence programmatically. The values of the Status variable encode the following:

0	Convergence was achieved, or an optimization was not performed (because TECHNIQUE=NONE is specified).
1	The objective function could not be improved.
2	Convergence was not achieved because of a user interrupt or because a limit was exceeded, such as the maximum number of iterations or the maximum number of function evaluations. To modify these limits, see the MAXITER=, MAXFUNC=, and MAXTIME= options in the PROC HPLOGISTIC statement.
3	Optimization failed to converge because function or derivative evaluations failed at the starting values or during the iterations or because a feasible point that satisfies the parameter constraints could not be found in the parameter space.

Dimensions

The "Dimensions" table displays size measures that are derived from the model and the environment. For example, it displays the number of columns in the design matrix, the rank of the matrix, the largest number of design columns associated with an effect, the number of compute nodes in MPP mode, and the number of threads per node.

Fit Statistics

The "Fit Statistics" table displays a variety of likelihood-based measures of fit. All statistics are presented in "smaller is better" form.

The calculation of the information criteria uses the following formulas, where p denotes the number of effective parameters, f denotes the number of frequencies used, and l is the log likelihood evaluated at the converged estimates:

$$\text{AIC} = -2l + 2p$$

$$\text{AICC} = \begin{cases} -2l + 2pf/(f-p-1) & \text{when } f > p+2 \\ -2l + 2p(p+2) & \text{otherwise} \end{cases}$$

$$\text{BIC} = -2l + p\log(f)$$

If no FREQ statement is given, f equals n, the number of observations used.

The values displayed in the “Fit Statistics” table are not based on a normalized log-likelihood function.

Global Tests

The “Global Tests” table provides a statistical test for the hypothesis of whether the final model provides a better fit than a model without effects (an “intercept-only” model).

If you specify the NOINT option in the MODEL statement, the reference model is one where the linear predictor is 0 for all observations.

Partition for the Hosmer and Lemeshow Test

The “Partition for the Hosmer and Lemeshow Test” table displays the grouping used in the Hosmer-Lemeshow test. This table is displayed if you specify the LACKFIT option in the MODEL statement. See the section “The Hosmer-Lemeshow Goodness-of-Fit Test” on page 204 for details, and see Hosmer and Lemeshow (2000) for examples of using this partition.

Hosmer and Lemeshow Goodness-of-Fit Test

The “Hosmer and Lemeshow Goodness-of-Fit Test” table provides a test of the fit of the model; small p-values reject the null hypothesis that the fitted model is adequate. This table is displayed if you specify the LACKFIT option in the MODEL statement. See the section “The Hosmer-Lemeshow Goodness-of-Fit Test” on page 204 for further details.

Parameter Estimates

The parameter estimates, their estimated (asymptotic) standard errors, and p-values for the hypothesis that the parameter is 0 are presented in the “Parameter Estimates” table. If you request confidence intervals with the CL or ALPHA= options in the MODEL statement, confidence limits are produced for the estimate on the linear scale.

ODS Table Names

Each table created by the HPLOGISTIC procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 8.5.

Table 8.5 ODS Tables Produced by PROC HPLOGISTIC

Table Name	Description	Required Statement / Option
CandidateDetails	Details about candidates for entry into or removal from the model	SELECTION DETAILS=STEP
ClassLevels	Level information from the CLASS statement	CLASS
ConvergenceStatus	Status of optimization at conclusion of optimization	Default output
Dimensions	Model dimensions	Default output
FitStatistics	Fit statistics	Default output
GlobalTests	Test of the model versus the null model	Default output
IterHistory	Iteration history	Default output or PROC HPLOGISTIC ITSELECT
LackFitChiSq	Hosmer-Lemeshow chi-square test results	MODEL / LACKFIT
LackFitPartition	Partition for the Hosmer-Lemeshow test	MODEL / LACKFIT
ModelInfo	Information about the modeling environment	Default output

Table 8.5 *continued*

Table Name	Description	Required Statement / Option
NObs	Number of observations read and used, and number of events and trials, if applicable	Default output
ParameterEstimates	Solutions for the parameter estimates associated with effects in MODEL statements	Default output
PerformanceInfo	Information about the high-performance computing environment	Default output
ResponseProfile	Response categories and category modeled in models for binary and multinomial data	Default output
SelectedEffects	List of effects selected into model	SELECTION
SelectionDetails	Details about model selection, including fit statistics by step	SELECTION DETAILS=ALL
SelectionInfo	Information about the settings for model selection	SELECTION
SelectionReason	Reason why the particular model was selected	SELECTION
SelectionSummary	Summary information about model selection steps	SELECTION
StopReason	Reason for termination of model selection	SELECTION
Timing	Absolute and relative times for tasks performed by the procedure	PERFORMANCE DETAILS

Examples: HPLOGISTIC Procedure

Example 8.1: Model Selection

The following HPLOGISTIC statements examine the same data as in the section "Getting Started: HPLOGISTIC Procedure" on page 179, but they request model selection via the forward selection technique. Model effects are added in the order of their significance until no more effects make a significant improvement of the current model. The DETAILS=ALL option in the SELECTION statement requests that all tables related to model selection be produced.

```
proc hplogistic data=getStarted;
   class C;
   model y = C x1-x10;
   selection method=forward details=all;
run;
```

The model selection tables are shown in Output 8.1.1 through Output 8.1.4.

The "Selection Information" table in Output 8.1.1 summarizes the settings for the model selection. Effects are added to the model only if they produce a significant improvement as judged by comparing the *p*-value of a score test to the entry significance level (SLE), which is 0.05 by default. The forward selection stops when no effect outside the model meets this criterion.

Output 8.1.1 Selection Information

```
                          The HPLOGISTIC Procedure

                           Selection Information

             Selection Method                  Forward
             Select Criterion                  Significance Level
             Stop Criterion                    Significance Level
             Effect Hierarchy Enforced         None
             Entry Significance Level (SLE)    0.05
             Stop Horizon                      1
```

The "Selection Summary" table in Output 8.1.2 shows the effects that were added to the model and their significance level. Step 0 refers to the null model that contains only an intercept. In the next step, effect x8 made the most significant contribution to the model among the candidate effects ($p = 0.0381$). In step 2 the most significant contribution when adding an effect to a model that contains the intercept and x8 was made by x2. In the subsequent step no effect could be added to the model that would produce a *p*-value less than 0.05, so variable selection stops.

Output 8.1.2 Selection Summary Information

```
                         Selection Summary

                         Effect         Number         p
                 Step    Entered     Effects In     Value

                    0    Intercept            1       .
            ---------------------------------------------
                    1    x8                   2    0.0381
                    2    x2                   3    0.0255

Selection stopped because no candidate for entry is significant at the 0.05
level.

                 Selected Effects: Intercept x2 x8
```

The DETAILS=ALL option requests further detail information about the steps of the model selection. The "Candidate Details" table in Output 8.1.3 list all candidates for each step in the order of significance of their score tests. The effect with smallest *p*-value less than the SLE level of 0.05 is added in each step.

Output 8.1.3 Candidate Details

```
                Candidate Entry and Removal Details

                                     Candidate          P
            Step     Rank     Effect     For             Value

               1        1     x8         Entry          0.0381
                        2     x2         Entry          0.0458
                        3     x4         Entry          0.0557
                        4     x9         Entry          0.1631
                        5     C          Entry          0.1858
                        6     x1         Entry          0.2715
                        7     x10        Entry          0.4434
                        8     x5         Entry          0.7666
                        9     x3         Entry          0.8006
                       10     x7         Entry          0.8663
                       11     x6         Entry          0.9626

               2        1     x2         Entry          0.0255
                        2     x4         Entry          0.0721
                        3     x9         Entry          0.1080
                        4     C          Entry          0.1241
                        5     x1         Entry          0.2778
                        6     x10        Entry          0.5250
                        7     x5         Entry          0.6993
                        8     x7         Entry          0.7103
                        9     x3         Entry          0.8743
                       10     x6         Entry          0.9577
```

The DETAILS=ALL option also produces the "Selection Details" table, which provides fit statistics and the value of the score test chi-square statistic at each step.

Output 8.1.4 Selection Details

```
                                   Selection Details

                         Effects                   Pr >
Step    Description      In Model   Chi-Square     ChiSq        -2 LogL          AIC

   0    Initial Model          1                                123.820      125.820
   1    x8 entered             2        4.2986    0.0381        119.462      123.462
   2    x2 entered             3        4.9882    0.0255        114.396      120.396

                                   Selection Details

                         Step          AICC            BIC

                            0       125.861        128.425
                            1       123.586        128.672
                            2       120.646        128.212
```

Output 8.1.5 displays information about the selected model. Notice that the –2 log likelihood value in the "Fit Statistics" table is larger than the value for the full model in Figure 8.9. This is expected because the selected model contains only a subset of the parameters. Because the selected model is more parsimonious than the full model, the discrepancy between the –2 log likelihood and the information criteria is less severe than previously noted.

Output 8.1.5 Fit Statistics and Null Test

```
                                    Fit Statistics

                    -2 Log Likelihood                    114.40
                    AIC  (smaller is better)             120.40
                    AICC  (smaller is better)            120.65
                    BIC  (smaller is better)             128.21

                     Testing Global Null Hypothesis: BETA=0

             Test                 Chi-Square       DF      Pr > ChiSq

             Likelihood Ratio         9.4237        2          0.0090
```

The parameter estimates of the selected model are given in Output 8.1.6. Notice that the effects are listed in the "Parameter Estimates" table in the order in which they were specified in the MODEL statement and not in the order in which they were added to the model.

Output 8.1.6 Parameter Estimates

```
                             Parameter Estimates

                                      Standard
    Parameter     Estimate       Error        DF     t Value     Pr > |t|

    Intercept       0.8584      0.5503     Infty        1.56       0.1188
    x2             -0.2502      0.1146     Infty       -2.18       0.0290
    x8              1.7840      0.7908     Infty        2.26       0.0241
```

You can construct the prediction equation for this model from the parameter estimates as follows. The estimated linear predictor for an observation is

$$\widehat{\eta} = 0.8584 - 0.2503 \times x_2 + 1.7840 \times x_8$$

and the predicted probability that variable y takes on the value 0 is

$$\widehat{\Pr}(Y = 0) = \frac{1}{1 + \exp\{-\widehat{\eta}\}}$$

Example 8.2: Modeling Binomial Data

If $Y_1, \cdots, Y_n$ are independent binary (Bernoulli) random variables with common success probability π, then their sum is a binomial random variable. In other words, a binomial random variable with parameters n and π can be generated as the sum of n Bernoulli(π) random experiments. The HPLOGISTIC procedure uses a special syntax to express data in binomial form, the *events/trials* syntax.

Consider the following data, taken from Cox and Snell (1989, pp. 10–11), of the number, r, of ingots not ready for rolling, out of n tested, for a number of combinations of heating time and soaking time. If each test is carried out independently and if for a particular combination of heating and soaking time there is a constant probability that the tested ingot is not ready for rolling, then the random variable r follows a Binomial(n, π) distribution, where the success probability π is a function of heating and soaking time.

```
data Ingots;
   input Heat Soak r n @@;
   Obsnum= _n_;
   datalines;
7 1.0 0 10   14 1.0 0 31   27 1.0 1 56   51 1.0 3 13
7 1.7 0 17   14 1.7 0 43   27 1.7 4 44   51 1.7 0  1
7 2.2 0  7   14 2.2 2 33   27 2.2 0 21   51 2.2 0  1
7 2.8 0 12   14 2.8 0 31   27 2.8 1 22   51 4.0 0  1
7 4.0 0  9   14 4.0 0 19   27 4.0 1 16
;
```

The following statements show the use of the events/trials syntax to model the binomial response. The *events* variable in this situation is r, the number of ingots not ready for rolling, and the *trials* variable is n, the number of ingots tested. The dependency of the probability of not being ready for rolling is modeled as a function of heating time, soaking time, and their interaction. The OUTPUT statement stores the linear predictors and the predicted probabilities in the Out data set along with the ID variable.

```
proc hplogistic data=Ingots;
   model r/n = Heat Soak Heat*Soak;
   id Obsnum;
   output out=Out xbeta predicted=Pred;
run;
```

The “Performance Information” table in Output 8.2.1 shows that the procedure executes in client (SMP) mode. The example is executed on a single machine with the same number of cores as the number of threads used; that is, one computational thread was spawned per CPU.

Output 8.2.1 Performance Information

```
                          The HPLOGISTIC Procedure

                          Performance Information

                       Execution Mode       On client
                       Number of Threads    8
```

The “Model Information” table shows that the data are modeled as binomially distributed with a logit link function (Output 8.2.2). This is the default link function in the HPLOGISTIC procedure for binary and binomial data. The procedure estimates the parameters of the model by a Newton-Raphson algorithm.

Output 8.2.2 Model Information and Number of Observations

```
                             Model Information

            Data Source                    WORK.INGOTS
            Response Variable (Events)     r
            Response Variable (Trials)     n
            Distribution                   Binomial
            Link Function                  Logit
            Optimization Technique         Newton-Raphson with Ridging

                   Number of Observations Read              19
                   Number of Observations Used              19
                   Number of Events                         12
                   Number of Trials                        387
```

The second table in Output 8.2.2 shows that all 19 observations in the data set were used in the analysis, and that the total number of events and trials equal 12 and 387, respectively. These are the sums of the variables r and n across all observations.

Output 8.2.3 displays the “Iteration History” and convergence status tables for this run. The HPLOGISTIC procedure converged after four iterations (not counting the initial setup iteration) and meets the GCONV= convergence criterion.

Output 8.2.3 Iteration History and Convergence Status

```
                          Iteration History

                                       Objective                       Max
   Iteration     Evaluations          Function          Change     Gradient

           0               4      0.7676329445          .           6.378002
           1               2      0.7365832479      0.03104970     0.754902
           2               2      0.7357086248      0.00087462     0.023623
           3               2      0.7357075299      0.00000109      0.00003
           4               2      0.7357075299      0.00000000     5.42E-11

              Convergence criterion (GCONV=1E-8) satisfied.
```

Output 8.2.4 displays the "Dimensions" table for the model. There are four columns in the design matrix of the model (the **X** matrix); they correspond to the intercept, the Heat effect, the Soak effect, and the interaction of the Heat and Soak effects. The model is nonsingular, since the rank of the crossproducts matrix equals the number of columns in **X**. All parameters are estimable and participate in the optimization.

Output 8.2.4 Dimensions in Binomial Logistic Regression

```
                             Dimensions

                 Columns in X                           4
                 Number of Effects                      4
                 Max Effect Columns                     1
                 Rank of Cross-product Matrix           4
                 Parameters in Optimization             4
```

Output 8.2.5 displays the "Fit Statistics" table for this run. Evaluated at the converged estimates, –2 times the value of the log-likelihood function equals 27.9569. Further fit statistics are also given, all of them in "smaller is better" form. The AIC, AICC, and BIC criteria are used to compare non-nested models and to penalize the model fit for the number of observations and parameters. The –2 log-likelihood value can be used to compare nested models by way of a likelihood ratio test.

Output 8.2.5 Fit Statistics

```
                           Fit Statistics

                -2 Log Likelihood                 27.9569
                AIC (smaller is better)           35.9569
                AICC (smaller is better)          38.8140
                BIC (smaller is better)           39.7346
```

Output 8.2.6 shows the test of the global hypothesis that the effects jointly do not impact the probability of ingot readiness. The chi-square test statistic can be obtained by comparing the –2 log-likelihood value of the model with covariates to the value in the intercept-only model. The test is significant with a *p*-value of 0.0082. One or more of the effects in the model have a significant impact on the probability of ingot readiness.

Output 8.2.6 Null Test

```
                   Testing Global Null Hypothesis: BETA=0

            Test                 Chi-Square       DF     Pr > ChiSq

            Likelihood Ratio        11.7663        3         0.0082
```

The “Parameter Estimates” table in Output 8.2.7 displays the estimates and standard errors of the model effects.

Output 8.2.7 Parameter Estimates

```
                              Parameter Estimates

                                   Standard
       Parameter    Estimate          Error        DF     t Value      Pr > |t|

       Intercept     -5.9902         1.6666     Infty       -3.59        0.0003
       Heat          0.09634        0.04707     Infty        2.05        0.0407
       Soak           0.2996         0.7551     Infty        0.40        0.6916
       Heat*Soak    -0.00884        0.02532     Infty       -0.35        0.7270
```

You can construct the prediction equation of the model from the “Parameter Estimates” table. For example, an observation with Heat equal to 14 and Soak equal to 1.7 has linear predictor

$$\widehat{\eta} = -5.9902 + 0.09634 \times 14 + 0.2996 \times 1.7 - 0.00884 \times 14 \times 7 = -4.34256$$

The probability that an ingot with these characteristics is not ready for rolling is

$$\widehat{\pi} = \frac{1}{1 + \exp\{-(-4.34256)\}} = 0.01284$$

The OUTPUT statement computes these linear predictors and probabilities and stores them in the Out data set. This data set also contains the ID variable, which is used by the following statements to attach the covariates to these statistics. Output 8.2.8 shows the probability that an ingot with Heat equal to 14 and Soak equal to 1.7 is not ready for rolling.

```
data Out;
   merge Out Ingots;
   by Obsnum;
proc print data=Out;
   where Heat=14 & Soak=1.7;
run;
```

Output 8.2.8 Predicted Probability for Heat=14 and Soak=1.7

```
        Obs    Obsnum       Pred         Xbeta      Heat     Soak    r     n

          6       6      0.012836    -4.34256      14       1.7    0    43
```

Binomial data are a form of grouped binary data where "successes" in the underlying Bernoulli trials are totaled. You can thus unwind data for which you use the events/trials syntax and fit it with techniques for binary data.

The following DATA step expands the Ingots data set with 12 events in 387 trials into a binary data set with 387 observations.

```
data Ingots_binary;
   set Ingots;
   do i=1 to n;
     if i <= r then y=1; else y = 0;
     output;
   end;
run;
```

The following HPLOGISTIC statements fit the model with Heat effect, Soak effect, and their interaction to the binary data set. The `event='1'` response-variable option in the MODEL statement ensures that the HPLOGISTIC procedure models the probability that the variable y takes on the value ('1').

```
proc hplogistic data=Ingots_binary;
   model y(event='1') = Heat Soak Heat*Soak;
run;
```

Output 8.2.9 displays the "Performance Information", "Model Information," "Number of Observations," and the "Response Profile" tables. The data are now modeled as binary (Bernoulli distributed) with a logit link function. The "Response Profile" table shows that the binary response breaks down into 375 observations where y equals zero and 12 observations where y equals 1.

Output 8.2.9 Model Information in Binary Model

```
                          The HPLOGISTIC Procedure

                          Performance Information

                       Execution Mode       On client
                       Number of Threads    8

                             Model Information

              Data Source                 WORK.INGOTS_BINARY
              Response Variable           y
              Distribution                Binary
              Link Function               Logit
              Optimization Technique      Newton-Raphson with Ridging
```

Output 8.2.9 *continued*

```
                   Number of Observations Read             387
                   Number of Observations Used             387

                                Response Profile

                              Ordered             Total
                                Value     y    Frequency

                                    1     0          375
                                    2     1           12

                   You are modeling the probability that y='1'.
```

Output 8.2.10 displays the result for the test of the global null hypothesis and the parameter estimates. These results match those in Output 8.2.6 and Output 8.2.7.

Output 8.2.10 Null Test and Parameter Estimates

```
                    Testing Global Null Hypothesis: BETA=0

                Test                Chi-Square       DF     Pr > ChiSq

                Likelihood Ratio       11.7663        3         0.0082

                              Parameter Estimates

                                         Standard
           Parameter    Estimate            Error        DF    t Value    Pr > |t|

           Intercept     -5.9902           1.6666     Infty      -3.59      0.0003
           Heat          0.09634          0.04707     Infty       2.05      0.0407
           Soak           0.2996           0.7551     Infty       0.40      0.6916
           Heat*Soak    -0.00884          0.02532     Infty      -0.35      0.7270
```

Example 8.3: Ordinal Logistic Regression

Consider a study of the effects of various cheese additives on taste. Researchers tested four cheese additives and obtained 52 response ratings for each additive. Each response was measured on a scale of nine categories ranging from strong dislike (1) to excellent taste (9). The data, given in McCullagh and Nelder (1989, p. 175) in the form of a two-way frequency table of additive by rating, are saved in the data set Cheese by using the following program. The variable y contains the response rating. The variable Additive specifies the cheese additive (1, 2, 3, or 4). The variable freq gives the frequency with which each additive received each rating.

```
data Cheese;
   do Additive = 1 to 4;
      do y = 1 to 9;
         input freq @@;
         output;
      end;
   end;
   label y='Taste Rating';
   datalines;
0  0  1  7  8  8 19  8  1
6  9 12 11  7  6  1  0  0
1  1  6  8 23  7  5  1  0
0  0  0  1  3  7 14 16 11
;
```

The response variable y is ordinally scaled. A cumulative logit model is used to investigate the effects of the cheese additives on taste. The following statements invoke PROC HPLOGISTIC to fit this model with y as the response variable and three indicator variables as explanatory variables, with the fourth additive as the reference level. With this parameterization, each Additive parameter compares an additive to the fourth additive.

```
proc hplogistic data=Cheese;
   freq freq;
   class Additive(ref='4') / param=ref ;
   model y=Additive;
   title 'Multiple Response Cheese Tasting Experiment';
run;
```

Results from the logistic analysis are shown in Output 8.3.1 through Output 8.3.3.

The “Response Profile” table in Output 8.3.1 shows that the strong dislike (y=1) end of the rating scale is associated with lower Ordered Values in the “Response Profile” table; hence the probability of disliking the additives is modeled.

Output 8.3.1 Proportional Odds Model Regression Analysis

```
                   Multiple Response Cheese Tasting Experiment

                             The HPLOGISTIC Procedure

                             Performance Information

                          Execution Mode       On client
                          Number of Threads    8

                                Model Information

              Data Source                 WORK.CHEESE
              Response Variable           y
              Frequency Variable          freq
              Class Parameterization      Reference
              Distribution                Multinomial (ordered)
              Link Function               Cumulative Logit
              Optimization Technique      Newton-Raphson with Ridging

                             Class Level Information

                                          Reference
                   Class       Levels     Value        Values

                   Additive         4     4            1 2 3 4

                   Number of Observations Read              36
                   Number of Observations Used              28
                   Sum of Frequencies Read                 208
                   Sum of Frequencies Used                 208

                                 Response Profile

                            Ordered    Taste         Total
                              Value    Rating    Frequency

                                  1    1                 7
                                  2    2                10
                                  3    3                19
                                  4    4                27
                                  5    5                41
                                  6    6                28
                                  7    7                39
                                  8    8                25
                                  9    9                12

  You are modeling the probabilities of levels of y having lower Ordered Values
                          in the Response Profile Table.
```

Output 8.3.2 Proportional Odds Model Regression Analysis

```
                          Iteration History

                                Objective                         Max
    Iteration    Evaluations     Function          Change    Gradient

            0              4   2.0668312595          .       0.137412
            1              2   1.7319560317     0.33487523   0.062757
            2              2   1.7105150048     0.02144103   0.008919
            3              2   1.7099716191     0.00054339    0.00035
            4              2   1.7099709251     0.00000069   6.981E-7
            5              2   1.7099709251     0.00000000   2.98E-12

              Convergence criterion (GCONV=1E-8) satisfied.

                               Dimensions

                 Columns in X                          11
                 Number of Effects                      2
                 Max Effect Columns                     3
                 Rank of Cross-product Matrix          11
                 Parameters in Optimization            11

                             Fit Statistics

                -2 Log Likelihood                  711.35
                AIC (smaller is better)            733.35
                AICC (smaller is better)           734.69
                BIC (smaller is better)            770.06

                 Testing Global Null Hypothesis: BETA=0

         Test                 Chi-Square       DF     Pr > ChiSq

         Likelihood Ratio       148.4539        3         <.0001
```

The positive value (1.6128) for the parameter estimate for Additive=1 in Output 8.3.3 indicates a tendency toward the lower-numbered categories of the first cheese additive relative to the fourth. In other words, the fourth additive tastes better than the first additive. Similarly, the second and third additives are both less favorable than the fourth additive. The relative magnitudes of these slope estimates imply the preference ordering: fourth, first, third, second.

Output 8.3.3 Proportional Odds Model Regression Analysis

```
                              Parameter Estimates

              Taste                  Standard
Parameter     Rating    Estimate        Error        DF    t Value     Pr > |t|

Intercept     1          -7.0802       0.5640     Infty     -12.55       <.0001
Intercept     2          -6.0250       0.4764     Infty     -12.65       <.0001
Intercept     3          -4.9254       0.4257     Infty     -11.57       <.0001
Intercept     4          -3.8568       0.3880     Infty      -9.94       <.0001
Intercept     5          -2.5206       0.3453     Infty      -7.30       <.0001
Intercept     6          -1.5685       0.3122     Infty      -5.02       <.0001
Intercept     7         -0.06688       0.2738     Infty      -0.24       0.8071
Intercept     8           1.4930       0.3357     Infty       4.45       <.0001
Additive 1                1.6128       0.3805     Infty       4.24       <.0001
Additive 2                4.9646       0.4767     Infty      10.41       <.0001
Additive 3                3.3227       0.4218     Infty       7.88       <.0001
```

Example 8.4: Conditional Logistic Regression for Matched Pairs Data

In matched pairs (*case-control*) studies, conditional logistic regression is used to investigate the relationship between an outcome of being an event (case) or a non-event (control) and a set of prognostic factors.

The following data are a subset of the data from the Los Angeles Study of the Endometrial Cancer Data in Breslow and Day (1980). There are 63 matched pairs, each consisting of a case of endometrial cancer (Outcome=1) and a control (Outcome=0). The case and corresponding control have the same ID. Two prognostic factors are included: Gall (an indicator variable for gall bladder disease) and Hyper (an indicator variable for hypertension). The goal of the case-control analysis is to determine the relative risk for gall bladder disease, controlling for the effect of hypertension.

```
data Data1;
  do ID=1 to 63;
    do Outcome = 1 to 0 by -1;
      input Gall Hyper @@;
      output;
    end;
  end;
  datalines;
0 0  0 0    0 0  0 0    0 1  0 1    0 0  1 0    1 0  0 1
0 1  0 0    1 0  0 0    1 1  0 1    0 0  0 0    0 0  0 0
1 0  0 0    0 0  0 1    1 0  0 1    1 0  1 0    1 0  0 1
0 1  0 0    0 0  1 1    0 0  1 1    0 0  0 1    0 1  0 0
0 0  1 1    0 1  0 1    0 1  0 0    0 0  0 0    0 0  0 0
0 0  0 1    1 0  0 1    0 0  0 1    1 0  0 0    0 1  0 0
0 1  0 0    0 1  0 0    0 1  0 0    0 0  0 0    1 1  1 1
0 0  0 1    0 1  0 0    0 1  0 1    0 1  0 1    0 1  0 0
0 0  0 0    0 1  1 0    0 0  0 1    0 0  0 0    1 0  0 0
0 0  0 0    1 1  0 0    0 1  0 0    0 0  0 0    0 1  0 1
0 0  0 0    0 1  0 1    0 1  0 0    0 1  0 0    1 0  0 0
```

```
0 0  0 0    1 1  1 0    0 0  0 0    0 0  0 0    1 1  0 0
1 0  1 0    0 1  0 0    1 0  0 0
;
```

When each matched set consists of one event and one non-event, the conditional likelihood is given by

$$\prod_i (1 + \exp(-(\boldsymbol{x}_{i1} - \boldsymbol{x}_{i0})'\boldsymbol{\psi})^{-1}$$

where $\boldsymbol{x}_{i1}$ and $\boldsymbol{x}_{i0}$ are vectors that represent the prognostic factors for the event and non-event, respectively, of the ith matched set. This likelihood is identical to the likelihood of fitting a logistic regression model to a set of data with constant response, where the model contains no intercept term and has explanatory variables given by $\boldsymbol{d}_i = \boldsymbol{x}_{i1} - \boldsymbol{x}_{i0}$ (Breslow 1982).

To apply this method, the following DATA step transforms each matched pair into a single observation, where the variables Gall and Hyper contain the differences between the corresponding values for the case and the control (case – control). The variable Outcome, which is used as the response variable in the logistic regression model, is given a constant value of 0 (which is the Outcome value for the control, although any constant, numeric or character, suffices).

```
data Data2;
   set Data1;
   drop id1 gall1 hyper1;
   retain id1 gall1 hyper1 0;
   if (ID = id1) then do;
      Gall=gall1-Gall; Hyper=hyper1-Hyper;
      output;
   end;
   else do;
      id1=ID; gall1=Gall; hyper1=Hyper;
   end;
run;
```

Note that there are 63 observations in the data set, one for each matched pair. Since the number of observations (n) is halved, statistics that depend on n such as R^2 will be incorrect. The variable Outcome has a constant value of 0.

In the following statements, PROC HPLOGISTIC is invoked with the NOINT option to obtain the conditional logistic model estimates. Because the option CL is specified, PROC HPLOGISTIC computes a 95% confidence interval for the parameter.

```
proc hplogistic data=Data2;
   model outcome=Gall / noint cl;
run;
```

Results from the conditional logistic analysis are shown in Output 8.4.1 through Output 8.4.3.

Output 8.4.1 shows that you are fitting a binary logistic regression where the response variable Outcome has only one level.

Output 8.4.1 Conditional Logistic Regression (Gall as Risk Factor)

```
                   Multiple Response Cheese Tasting Experiment

                             The HPLOGISTIC Procedure

                             Performance Information

                          Execution Mode       On client
                          Number of Threads    8

                                Model Information

              Data Source                 WORK.DATA2
              Response Variable           Outcome
              Distribution                Binary
              Link Function               Logit
              Optimization Technique      Newton-Raphson with Ridging

                   Number of Observations Read              63
                   Number of Observations Used              63

                                 Response Profile

                            Ordered                   Total
                              Value     Outcome    Frequency

                                  1     0                 63

                 You are modeling the probability that Outcome='0'.
```

Output 8.4.2 shows that the model is marginally significant (p=0.0550).

Output 8.4.2 Conditional Logistic Regression (Gall as Risk Factor)

```
                                Iteration History

                                       Objective                        Max
       Iteration     Evaluations        Function         Change    Gradient

               0               4    0.6662698453          .        0.015669
               1               2    0.6639330101     0.00233684    0.001351
               2               2    0.6639171997     0.00001581     6.88E-6
               3               2    0.6639171993     0.00000000    1.83E-10

                  Convergence criterion (GCONV=1E-8) satisfied.

                                   Dimensions

                     Columns in X                           1
                     Number of Effects                      1
                     Max Effect Columns                     1
                     Rank of Cross-product Matrix           1
                     Parameters in Optimization             1
```

Output 8.4.2 *continued*

```
                              Fit Statistics

                -2 Log Likelihood                  83.6536
                AIC (smaller is better)            85.6536
                AICC (smaller is better)           85.7191
                BIC (smaller is better)            87.7967

                 Testing Global Null Hypothesis: BETA=0

         Test                 Chi-Square       DF     Pr > ChiSq

         Likelihood Ratio         3.6830        1         0.0550
```

Note that there is no intercept term in the "Parameter Estimates" table in Output 8.4.3. The intercepts have been *conditioned out* of the analysis.

Output 8.4.3 Conditional Logistic Regression (Gall as Risk Factor)

```
                             Parameter Estimates

                          Standard
   Parameter    Estimate     Error       DF    t Value    Pr > |t|     Alpha

   Gall           0.9555    0.5262    Infty       1.82      0.0694      0.05

                             Parameter Estimates

                     Parameter        Lower       Upper

                     Gall          -0.07589      1.9869
```

The odds ratio estimate for Gall is $\exp(0.9555) = 2.60$, which is marginally significant (p=0.0694) and which is an estimate of the relative risk for gall bladder disease. A subject who has gall bladder disease has 2.6 times the odds of having endometrial cancer as a subject who does not have gall bladder disease. A 95% confidence interval for this relative risk, produced by exponentiating the confidence interval for the parameter, is (0.927, 7.293).

References

Akaike, H. (1974), "A New Look at the Statistical Model Identification," *IEEE Transaction on Automatic Control*, AC-19, 716–723.

Breslow, N. E. (1982), "Covariance Adjustment of Relative-Risk Estimates in Matched Studies," *Biometrics*, 38, 661–672.

Breslow, N. E. and Day, N. E. (1980), *Statistical Methods in Cancer Research, Volume I: The Analysis of Case-Control Studies*, IARC Scientific Publications, No. 32, Lyon, France: International Agency for Research on Cancer.

Burnham, K. P. and Anderson, D. R. (1998), *Model Selection and Inference: A Practical Information-Theoretic Approach*, New York: Springer-Verlag.

Cox, D. R. and Snell, E. J. (1989), *The Analysis of Binary Data*, Second Edition, London: Chapman & Hall.

Dennis, J. E., Gay, D. M., and Welsch, R. E. (1981), "An Adaptive Nonlinear Least-Squares Algorithm," *ACM Transactions on Mathematical Software*, 7, 348–368.

Dennis, J. E. and Mei, H. H. W. (1979), "Two New Unconstrained Optimization Algorithms Which Use Function and Gradient Values," *Journal of Optimization Theory Applications*, 28, 453–482.

Eskow, E. and Schnabel, R. B. (1991), "Algorithm 695: Software for a New Modified Cholesky Factorization," *ACM Transactions on Mathematical Software*, 17, 306–312.

Fletcher, R. (1987), *Practical Methods of Optimization*, Second Edition, Chichester, UK: John Wiley & Sons.

Gay, D. M. (1983), "Subroutines for Unconstrained Minimization," *ACM Transactions on Mathematical Software*, 9, 503–524.

Hosmer, D. W., Jr. and Lemeshow, S. (2000), *Applied Logistic Regression*, Second Edition, New York: John Wiley & Sons.

Hurvich, C. M. and Tsai, C.-L. (1989), "Regression and Time Series Model Selection in Small Samples," *Biometrika*, 76, 297–307.

Magee, L. (1990), "R^2 Measures Based on Wald and Likelihood Ratio Joint Significant Tests," *The American Statistician*, 44, 250–253.

McCullagh, P. and Nelder, J. A. (1989), *Generalized Linear Models*, Second Edition, London: Chapman & Hall.

Moré, J. J. and Sorensen, D. C. (1983), "Computing a Trust-Region Step," *SIAM Journal on Scientific and Statistical Computing*, 4, 553–572.

Nagelkerke, N. J. D. (1991), "A Note on a General Definition of the Coefficient of Determination," *Biometrika*, 78, 691–692.

Schwarz, G. (1978), "Estimating the Dimension of a Model," *Annals of Statistics*, 6, 461–464.

Chapter 9

The HPNEURAL Procedure

Contents

Overview: HPNEURAL Procedure

The HPNEURAL procedure is a high-performance procedure that trains a multilayer perceptron neural network. For more information about multilayer perceptron neural networks, see Bishop (1995). PROC HPNEURAL can also use the trained network to score the input data set.

PROC HPNEURAL reads and writes data in distributed form and makes full use of multicore computers and distributed computing environments to perform training and scoring. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details.

Multilayer perceptron neural networks require the unconstrained minimization of a nonlinear objective function. Because there are currently no practical methods to guarantee finding a global minimum of that objective function, one way to be reasonably sure of finding a good solution is to train the network multiple times using different sets of initial values for the weights. Thus, even problems with smaller numbers of variables and smaller numbers of training observations have the potential to benefit from the use of multicore computers and distributed computing environments.

PROC HPNEURAL Features

The HPNEURAL procedure was designed with two goals in mind: to perform efficient, high-speed training of neural networks, and to be as easy to use as possible while still creating models that fit the training data well and generalize well. With these goals in mind, most parameters for the neural network are automatically selected.

The following list summarizes some basic features of PROC HPNEURAL:

- ability to train and score on a massively parallel SAS high-performance appliance
- parallel read of input data and parallel write of output data when the data source is the appliance database
- high degree of multithreading during all phases of training and scoring
- automatic standardization of input and target variables
- intelligent defaults for most neural network parameters such as activation and error functions
- either automatic or manual selection and use of a validation data subset
- automatic termination of training when the validation error stops improving
- ability to weight individual observations or automatically use inverse prior probabilities as weights

PROC HPNEURAL Contrasted with Other SAS Procedures

For general contrasts between SAS High-Performance Analytics procedures and other SAS procedures, see the section "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10. Table 9.1 compares the HPNEURAL procedure with the SAS Enterprise Miner Neural Network Node.

Table 9.1 Comparison of PROC HPNEURAL and SAS Enterprise Miner Neural Network Node

PROC HPNEURAL	**Neural Network Node**
Multithreaded	Single-threaded
Can execute in a distributed computing environment	Can execute only on a single computer
Has few required user-specified parameters so users with minimal experience with neural networks can obtain good solutions to problems that are amenable to supervised training	Gives you fine control over the myriad of possible choices of parameters that control the training of a neural network
Automatically selects the data standardization methods, network architecture, activation functions, error functions, and weight initialization method	Can use many different user-specified data standardization methods, network architectures, activation functions, error functions, and ways of selecting initial weights
Uses the limited memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) optimization method (Nocedal and Liu 1989) with proprietary enhancements. LBFGS was chosen for the HPNEURAL procedure because of both its speed of training and its limited use of memory, which can be especially important for problems with large amounts of training data.	Can use a variety of conjugate gradient or quasi-Newton optimization methods, but not LBFGS

Getting Started: HPNEURAL Procedure

The HPNEURAL procedure does not have very many parameters that you must specify. It simply needs to know where the training data are (the DATA= option in the PROC statement), the names and types of the input variables (the INPUT statement), the names and types of the target variables (the TARGET statement), the number of hidden neurons (the HIDDEN statement), the number of training runs, each using different randomly generated initial weights (the TRAIN statement), and, optionally, where to write the score file that contains targets from the input file and predicted targets from the trained network (the SCORE statement).

The single most important parameter you can specify is the number of hidden neurons in the network. A good strategy is to start with a small number (such as the minimum, which is 2) and slowly increase the number until the validation error stops improving.

The next most important parameter you can specify is the number of times the network is to be retrained using different sets of initial weights (the NUMTRIES option in the TRAIN statement). A good strategy is to start with 5 (the default) and increase by 2 until the validation error stops improving.

The following small example trains a neural network to predict the type of iris plant, given several measurements. The DATA step contains 150 observations derived from the R. A. Fisher (1936) Iris data set:

```
title 'Fisher (1936) Iris Data';

proc format;
   value specname
      1='Setosa    '
      2='Versicolor'
      3='Virginica ';
run;

data iris;
   input SepalLength SepalWidth PetalLength PetalWidth Species @@;
   format Species specname.;
   datalines;
50 33 14 02 1 64 28 56 22 3 65 28 46 15 2 67 31 56 24 3
63 28 51 15 3 46 34 14 03 1 69 31 51 23 3 62 22 45 15 2
59 32 48 18 2 46 36 10 02 1 61 30 46 14 2 60 27 51 16 2
65 30 52 20 3 56 25 39 11 2 65 30 55 18 3 58 27 51 19 3
68 32 59 23 3 51 33 17 05 1 57 28 45 13 2 62 34 54 23 3
77 38 67 22 3 63 33 47 16 2 67 33 57 25 3 76 30 66 21 3
49 25 45 17 3 55 35 13 02 1 67 30 52 23 3 70 32 47 14 2
64 32 45 15 2 61 28 40 13 2 48 31 16 02 1 59 30 51 18 3
55 24 38 11 2 63 25 50 19 3 64 32 53 23 3 52 34 14 02 1
49 36 14 01 1 54 30 45 15 2 79 38 64 20 3 44 32 13 02 1
67 33 57 21 3 50 35 16 06 1 58 26 40 12 2 44 30 13 02 1
77 28 67 20 3 63 27 49 18 3 47 32 16 02 1 55 26 44 12 2
50 23 33 10 2 72 32 60 18 3 48 30 14 03 1 51 38 16 02 1
61 30 49 18 3 48 34 19 02 1 50 30 16 02 1 50 32 12 02 1
61 26 56 14 3 64 28 56 21 3 43 30 11 01 1 58 40 12 02 1
51 38 19 04 1 67 31 44 14 2 62 28 48 18 3 49 30 14 02 1
51 35 14 02 1 56 30 45 15 2 58 27 41 10 2 50 34 16 04 1
```

```
46 32 14 02 1 60 29 45 15 2 57 26 35 10 2 57 44 15 04 1
50 36 14 02 1 77 30 61 23 3 63 34 56 24 3 58 27 51 19 3
57 29 42 13 2 72 30 58 16 3 54 34 15 04 1 52 41 15 01 1
71 30 59 21 3 64 31 55 18 3 60 30 48 18 3 63 29 56 18 3
49 24 33 10 2 56 27 42 13 2 57 30 42 12 2 55 42 14 02 1
49 31 15 02 1 77 26 69 23 3 60 22 50 15 3 54 39 17 04 1
66 29 46 13 2 52 27 39 14 2 60 34 45 16 2 50 34 15 02 1
44 29 14 02 1 50 20 35 10 2 55 24 37 10 2 58 27 39 12 2
47 32 13 02 1 46 31 15 02 1 69 32 57 23 3 62 29 43 13 2
74 28 61 19 3 59 30 42 15 2 51 34 15 02 1 50 35 13 03 1
56 28 49 20 3 60 22 40 10 2 73 29 63 18 3 67 25 58 18 3
49 31 15 01 1 67 31 47 15 2 63 23 44 13 2 54 37 15 02 1
56 30 41 13 2 63 25 49 15 2 61 28 47 12 2 64 29 43 13 2
51 25 30 11 2 57 28 41 13 2 65 30 58 22 3 69 31 54 21 3
54 39 13 04 1 51 35 14 03 1 72 36 61 25 3 65 32 51 20 3
61 29 47 14 2 56 29 36 13 2 69 31 49 15 2 64 27 53 19 3
68 30 55 21 3 55 25 40 13 2 48 34 16 02 1 48 30 14 01 1
45 23 13 03 1 57 25 50 20 3 57 38 17 03 1 51 38 15 03 1
55 23 40 13 2 66 30 44 14 2 68 28 48 14 2 54 34 17 02 1
51 37 15 04 1 52 35 15 02 1 58 28 51 24 3 67 30 50 17 2
63 33 60 25 3 53 37 15 02 1
;

proc hpneural data=iris;
   input SepalLength SepalWidth PetalLength PetalWidth;
   target Species / level=nom;
   hidden 2;
   train;
   score out=scores_iris;
run;
```

Figure 9.1 displays the SAS Log output. This shows the percentage of validation observations that were misclassified by the trained network. If there had been any interval targets, the log would have shown the absolute average percentage error and the absolute maximum percentage error for each interval target.

Figure 9.1 SAS Log Output

```
NOTE: Misclassification Error for target Species:   5.2632%
NOTE: There were 150 observations read from the data set WORK.IRIS.
NOTE: The data set WORK.SCORES_IRIS has 150 observations and 2 variables.
```

Figure 9.2 displays the “Model Information,” “Performance Information,” and “Number of Observations” tables. The HPNEURAL procedure creates a neural network model for the classification variable Species. Of the 150 observations, 38 are used as a validation subset, which consists of the first observation and every fourth observation thereafter. The other 112 observations make up the training subset.

The procedure executes in client mode. That is, the model is trained on the machine where the SAS session executes.

Figure 9.2 Model Information, Performance Information, and Number of Observations Tables

```
                          Fisher (1936) Iris Data

                           The HPNEURAL Procedure

                             Model Information

              Data Source                    WORK.IRIS
              Architecture                   One Hidden Layer
              Optimization Technique         Limited Memory BFGS
              Number of Input Variables      4
              Number of Target Variables     1
              Number of Hidden Neurons       2
              Number of Weights              31

                          Performance Information

                        Execution Mode       On client
                        Number of Threads    2

                   Number of Observations Read             150
                   Number of Observations Used             150
                   Number of Validation Obs.                38
```

Figure 9.3 displays the “Misclassification Table.” It shows the results of scoring the validation subset using the neural network model trained on the training subset. This example shows two incorrect classifications: two observations whose target value was “Virginica” were incorrectly classified as “Versicolor.”

Figure 9.3 Misclassification Table for Species

```
                    Misclassification Table for Species

                  Class:      Setosa    Versicolor    Virginica

                Setosa            13             0            0
                Versicolor         0            11            0
                Virginica          0             2           12
```

Syntax: HPNEURAL Procedure

The following statements are available in the HPNEURAL procedure:

> **PROC HPNEURAL** < **DATA=** >*SAS-data-set*
> < **DISTR=***data-distribution-options* >
> < **NOPRINT** > **;**
> **PERFORMANCE** *performance-options* **;**
> **ARCHITECTURE** *architecture-option* **;**
> **INPUT** *variables* < / **LEVEL=***level-options* > **;**
> **ID** *variables* **;**
> **WEIGHT** *variable* | **_INVERSE_PRIORS_ ;**
> **HIDDEN** *number* **;**
> **TARGET** *variables* < / **LEVEL=***level-options* > **;**
> **TRAIN** < **NUMTRIES=***number*> < **MAXITER=***number*> < **VALID=***variable* | **_NONE_** > **;**
> **SCORE OUT=***SAS-data-set* **;**

The PROC, INPUT, TARGET, HIDDEN, and TRAIN statements are required.

PROC HPNEURAL Statement

> **PROC HPNEURAL** < **DATA=** >*SAS-data-set*
> < **DISTR=***data-distribution-options* >
> < **NOPRINT** > **;**

The PROC HPNEURAL statement invokes the procedure. You can specify the following options in the PROC HPNEURAL statement:

DATA=*SAS-data-set*

names the SAS data set that contains the training and validation observations to be used by PROC HPNEURAL to train the neural network. The default is the most recently created data set. Each observation must contain the input, weight, id, and target variables specified in the associated INPUT, WEIGHT, ID, and TARGET statements.

By default, every fourth observation, starting with the first observation, is used as a validation observation. Any observations with missing values for input, weight, or target variables are discarded and are not used for training or validation error calculation.

If the procedure executes in a distributed environment, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. For information about the alongside-the-database model, see the section "Alongside-the-Database Execution" on page 15.

DISTR=ALL | SPLIT

specifies whether the input data set is to be replicated in the memory of each node in a distributed computing environment. If this option is not specified, PROC HPNEURAL makes this decision automatically based on the size of the input data set. This option is ignored if PROC HPNEURAL is not running in a distributed computing environment.

When using a distributed computing environment, PROC HPNEURAL usually divides the input data set among all the nodes to minimize the time it takes to optimize each try. However, if the input data set is small, dividing the data in this way might be inefficient because of the interconnect delay (the time it takes to send partial results between nodes). It might be more efficient to have each node have a complete copy of the data and run each try in parallel on separate nodes. Each try might take longer because it uses only a single node, but because the tries are running in parallel, it could take less time to finish all the tries.

You can force the data to be redistributed so that each node has a complete in-memory copy by specifying DISTR=ALL. You can prevent the data from being redistributed by specifying DISTR=SPLIT.

NOPRINT
specifies that no ODS tables be created.

ARCHITECTURE Statement

ARCHITECTURE *architecture-option* **;**

The ARCHITECTURE statement specifies the architecture of the neural network. The *architecture-option* must be one of the following:

LAYER1
specifies a multilayer perceptron with a single hidden layer.

LAYER1SKIP
specifies a multilayer perceptron with a single hidden layer and additional connections between each input and each target neuron.

LAYER2
specifies a multilayer perceptron with two hidden layers. The number of hidden neurons is split equally between the first and second layer. If the number of hidden neurons is odd, the first hidden layer has the extra neuron.

LAYER2SKIP
specifies a multilayer perceptron with two hidden layers and additional connections between each input and each target neuron. The number of hidden neurons is split equally between the first and second layer. If the number of hidden neurons is odd, the first hidden layer has the extra neuron.

The ARCHITECTURE statement is optional. The default is LAYER1.

HIDDEN Statement

HIDDEN *number* **;**

The HIDDEN statement specifies the *number* of hidden neurons in the network. The *number* must be an integer greater than 1. For two-layer architectures (LAYER2 and LAYER2SKIP), the hidden neurons are split between the first and second layer. In this case, if the number of hidden neurons is odd, the first hidden layer has the extra neuron.

All hidden neurons use a hyperbolic-tangent activation function.

You must include exactly one HIDDEN statement.

ID Statement

ID *variables* **;**

The ID statement lists one or more *variables* from the input data set that are transferred to the output data set, which is specified in the SCORE statement.

For documentation about the common ID statement in High-Performance Analytics procedures, see the section "ID Statement" on page 23 in Chapter 2, "Shared Concepts and Topics."

The ID statement is optional.

INPUT Statement

INPUT *variables* < / **LEVEL= INT | NOM** > **;**

The INPUT statement identifies *variables* in the input data set that are inputs to the neural network.

LEVEL = INT | NOM
specifies whether the *variables* are interval variables (INT), which must be numeric, or nominal variables (NOM), also known as classification variables, which can be numeric or character. The default for the LEVEL option is INT.

You must include one or more INPUT statements. You need more than one INPUT statement when you have both interval and classification input variables.

All interval input variables are automatically standardized to the range [–1, 1].

If an observation has missing values for any of the specified input variables, the observation is not used for training or for computing validation error.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing. The PERFORMANCE statement for High-Performance Analytics procedures is documented in the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics."

The PERFORMANCE statement is optional.

SCORE Statement

SCORE OUT=*SAS-data-set* **;**

The SCORE statement causes the HPNEURAL procedure to write the network's target and predicted output for each observation in the input data set to the output data set that is specified by the OUT option, along with any variables from the input data set that are specified in the ID statement.

The SCORE statement is optional.

TARGET Statement

TARGET *variables* < / **LEVEL= INT | NOM** > **;**

The TARGET statement identifies *variables* in the input data set that the network is to predict.

LEVEL = INT | NOM

specifies whether the *variables* are interval variables (INT), which must be numeric, or nominal variables (NOM), also known as classification variables, which can be numeric or character. The default for the LEVEL option is INT.

You must include one or more TARGET statements. You need more than one TARGET statement when you have both interval and classification target variables.

All target variables are automatically standardized to the range $[0.1, 0.9]$. The procedure automatically converts them back to their original scale before it computes fit statistics and writes predictions to the scoring data set.

For interval variables, all target neurons use a sigmoid activation function.

Classification variables have one target neuron per class level. Each of these neurons uses the softmax activation function to ensure that the sum of the outputs for all neurons is 1.0. The output of each neuron can then be interpreted as the probability that the variable is the corresponding class level.

If an observation has missing values for any of the specified target variables, the observation is not used for training or for computing validation error.

You cannot specify the same variable in both an INPUT statement and a TARGET statement.

TRAIN Statement

TRAIN < **NUMTRIES=***number*> < **MAXITER=***number*> < **VALID=***variable* | **_NONE_** > **;**

The TRAIN statement causes the HPNEURAL procedure to use the training data specified in the PROC HP-NEURAL statement to train a neural network model whose structure is specified in the ARCHITECTURE, INPUT, TARGET, and HIDDEN statements. The goal of training is to determine a set of network weights that best predicts the targets in the training data while still doing a good job of predicting targets of unseen data (that is, generalizing well and not overfitting).

Training starts with a pseudorandomly generated initial set of weights. PROC HPNEURAL then computes the objective function for the training subset (the sum of the squared differences between the target values of each observation and the outputs of the network), and the minimization algorithm adjusts the weights. This process is repeated until any one of the following conditions is met:

- The objective function that is computed using the training subset stops improving.
- The objective function that is computed using the validation subset stops improving.
- The process has been repeated the *number* of time specified in the MAXITER= option.

The weights that result in the smallest value of the objective function for the validation subset are saved and used for calculating fit statistics and for scoring.

You must include exactly one TRAIN statement.

NUMTRIES=*number*

specifies the *number* of times the network is to be trained using a different pseudorandomly generated starting point. Specifying the NUMTRIES= option helps ensure that the optimizer finds the set of weights that truly minimizes the objective function and does not return a local minimum. The value of *number* must be an integer between 1 and 99, 999. The default is 5.

MAXITER=*number*

specifies the maximum number of iterations (weight adjustments) for the optimizer to make before terminating.

Setting *number* to a large value does not mean that the optimizer actually iterates that many times. Often, training or validation error stops improving much sooner.

When you are training using large data sets, you can do a training run with MAXITER=1 to determine approximately how long each iteration will take.

The default is 50.

VALID=*variable* | _NONE_

specifies which observations are members of the validation subset. By default, every fourth observation (starting with the first observation) is selected as a member of the validation subset. If you specify VALID=*variable*, an observation is selected as a member of the validation subset if the value of *variable* is nonzero. If you specify VALID=_NONE_, there is no validation subset and training can stop only if the training error stops improving or if the number of iterations specified in the MAXITER= option have completed.

WEIGHT Statement

WEIGHT *variable* | _INVERSE_PRIORS_ ;

The WEIGHT statement usually identifies a numeric *variable* in the input data set that contains the weight to be placed on the prediction error (the difference between the output of the network and the target value specified in the input data set) for each observation during training.

If, instead of specifying a *variable*, you specify the keyword _INVERSE_PRIORS_, PROC HPNEURAL calculates the weight applied to the prediction error of each nominal target variable as the total number of observations divided by the number of observations whose target class is the same as the current observation (in other words, the inverse of the fraction of the time the target class occurs in the input data set).

If *variable* is less than or equal to 0 or is missing, the observation is not used for training or for computing validation error. When validation error is computed during training, the weights on the validation observations are used even though weights are not used when scoring.

The WEIGHT statement is optional. If a WEIGHT statement is not included, all observations are assigned a weight of 1.

Details: HPNEURAL Procedure

Computational Method

PROC HPNEURAL trains a multilayer perceptron neural network with a single hidden layer. For more information about multilayer perceptron neural networks, see Bishop (1995).

All activation functions for both hidden and target neurons are hyperbolic tangents. Because the output of the hyperbolic tangent function is limited to the range $[-1, 1]$, continuous target variables must be scaled so as not to exceed this range. PROC HPNEURAL automatically scales target variables to the range $[-0.9, 0.9]$ so that the network can predict any target variable without forcing the output neurons into their saturation region.

In addition, all continuous input variables are scaled to be in the range $[-1, 1]$. This scaling makes it easier to generate initial values of the weight variables that do not immediately cause the hidden neurons to be driven into their saturation region.

The error function for the network is the sum of the squared error of the target and the network output for each observation. This is a scalar function of the network weights. This function defines an error surface on which the optimization algorithm attempts to locate a minimum. Optimization is done in two parts: the limited memory Broyden–Fletcher–Goldfarb–Shanno (LBFGS) algorithm (Nocedal and Liu 1989) computes a direction along the error surface; then the Moré Thuente line search algorithm (Moré and Thuente 1992) finds a new minimum of the error function on the surface along that direction. If a sufficient decrease in the error function can be found, the weights that generated the new minimum are then used by the LBFGS algorithm to calculate a new descent direction. The process is repeated until a sufficient decrease in the error function cannot be obtained.

Both the LBFGS search direction algorithm and the Moré Thuente line search algorithm need to know the gradient of the error surface at several different points (sets of weights). This gradient is calculated by using the algorithm described in Wilamowski and Hao Yu (2010).

Besides terminating due to the inability to improve the error function, the optimization algorithm also stops if the validation error (which is calculated after each line search) lacks improvement 40 times in a row. The validation error is computed as the sum, over each validation observation, of the absolute difference between the target and the network output. Every fourth observation is used as a validation observation starting with the first observation. Validation observations are not used in any other way by the optimization algorithm.

Multithreading

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

The number of threads spawned by the HPNEURAL procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, the following statement causes the HPNEURAL procedure to determine threading as if it executed on a system with four CPUs, regardless of the actual CPU count:

  ```
  options cpucount=4;
  ```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This option overrides the system option. Specify NTHREADS=1 to force single-threaded execution.

- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement. This option overrides the CPUCOUNT system option and instructs the HPNEURAL procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output. By default, the HPNEURAL procedure allocates one thread per CPU.

The tasks multithreaded by the HPNEURAL procedure are primarily defined by dividing the data processed on a single machine among the threads; that is, the HPNEURAL procedure implements multithreading through a data-parallel model. For example, if the input data set has $1,000$ observations and PROC HPNEURAL is running with four threads, then 250 observations are associated with each thread.

Output Data Set

The output data set is specified by the OUT= option in the SCORE statement. If there is no SCORE statement, then no output data set is created, but fit statistics are still displayed in ODS tables.

Table 9.2 describes the columns of the output data set.

Table 9.2 Output Data Set Columns

Column	**Name**
Target value of the variable from the input data set	Same as the name of the variable in the input data set
Predicted value of the variable from the trained network	Same as the name of the variable in the input data set with a two-character string added as a prefix. The prefixes are as follows: • "P_" for an interval variable • "I_" for a classification variable If the resulting name is longer than 32 characters, it is truncated. If the name is no longer unique, a number is appended so it is unique.
For nominal values, additional columns for the raw network output for each level of each nominal variable. Because nominal variables use the softmax activation function, the raw value for a specific level is usually interpreted as the probability that the target variable is that level.	Prefix "P_", followed by the variable name, followed by the class level name. If the resulting name is longer than 32 characters, it is truncated. Special characters in the level name are replaced with "_". If the name is no longer unique, a number is appended so it is unique.

Displayed Output

PROC HPNEURAL displays basic fit statistics in the SAS Log and more detailed information in several ODS tables.

The statistics displayed in the log are based on the network's prediction accuracy on the validation subset. For interval variables, the procedure displays the average absolute percentage error and maximum absolute percentage error. The percentage is the percentage of the range of the variable across the entire input data set (not just the validation observations). For classification variables, the procedure displays the percentage of observations that were misclassified.

In addition to information displayed in the SAS log, the procedure generates ODS tables that give detailed information on the model structure, input data, detailed training results, and timings.

Other fit statistics can be computed from information in the score data set.

ODS Table Names

Each table created by the HPNEURAL procedure has a name associated with it, and 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 9.3.

Table 9.3 ODS Tables Produced by PROC HPNEURAL

Table Name	Description	Required Statement / Option
ClassLevels	Level information for input classification variables	INPUT with LEVEL=NOM
Details	Detailed real times for each phase of the procedure	PERFORMANCE with DETAILS option
ErrorSummary	Average and maximum errors for interval targets	TARGET with LEVEL=INT (the default)
Iteration	Training and validation error for each iteration of the best try	Default output
Misclassification	Misclassification matrix for target classification variables	TARGET with LEVEL=NOM
ModelInfo	Information about the modeling environment	Default output
Nobs	Number of observations read and used; number of validation observations used	Default output
PerformanceInfo	Information about the high-performance computing environment	Default output
Training	Training error, validation error, and reason for stopping for each try	Default output

References

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press.

Moré, J. J. and Thuente, D. J. (1992), “Line Search Algorithms with Guaranteed Sufficient Decrease,” *ACM Transactions on Mathematical Software*, 20, 286–307.

Nocedal, J. and Liu, D. C. (1989), “On the Limited Memory BFGS Method for Large Scale Optimization,” *Mathematical Programming*, 45, 503–528.

Wilamowski, B. M. and Hao Yu (2010), “Neural Network Learning without Backpropagation,” *IEEE Transactions on Neural Networks*, 21, 1793–1803.

Chapter 10

The HPNLIN Procedure

Contents

Overview: HPNLIN Procedure

The HPNLIN procedure is a high-performance procedure that uses either nonlinear least squares or maximum likelihood to fit nonlinear regression models on the SAS appliance. PROC HPNLIN enables you to specify the model with SAS programming statements, which gives you greater flexibility in modeling the relationship between the response variable and independent (regressor) variables than SAS procedures that use a more structured MODEL statement.

With the HPNLIN procedure you can read and write data in distributed form and perform analyses in parallel in symmetric multiprocessing (SMP) or massively parallel processing (MPP) mode. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details.

PROC HPNLIN Features

The HPNLIN procedure does the following:

- can perform analysis on a massively parallel SAS high-performance appliance
- reads input data in parallel and writes output data in parallel when the data source is the appliance database
- is highly multithreaded during all phases of analytic execution
- computes analytical derivatives of user-provided expressions for more robust parameter estimations
- evaluates user-provided expressions and their confidence limits with the ESTIMATE and PREDICT statements
- estimates parameters without specifying a particular distribution function by using the least squares method
- estimates parameters by using the maximum likelihood method when either a built-in distribution function is specified or a likelihood function is provided

PROC HPNLIN Contrasted with Other SAS Procedures

For general contrasts between SAS High Performance Analytics procedures and other SAS procedures, see the section "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10. The following remarks contrast the HPNLIN procedure with the NLIN and NLMIXED procedures in SAS/STAT software:

Like the NLIN procedure, the HPNLIN procedure estimates parameters by using least squares minimization for models that are specified by SAS programming statements. However, PROC HPNLIN can also perform maximum likelihood estimation when information about the response variable's distribution is provided. PROC HPNLIN also has a RESTRICT statement for specifying restrictions on parameter estimates that are more general than those that are available in PROC NLIN. Because the HPNLIN and NLIN procedures use different optimization techniques, the available options that control the estimation process and resulting parameter estimates can differ between these procedures when equivalent models and data are analyzed.

Although it does not support the specification of random effects, PROC HPNLIN is similar to PROC NLMIXED. Both procedures perform maximum likelihood estimation by using the same programming syntax and set of distributions to specify the model's mean term. Additionally, both PROC HPNLIN and PROC NLMIXED use the same optimization techniques and options. However, PROC NLMIXED does not support least squares parameter estimation.

Getting Started: HPNLIN Procedure

The most common use of the HPNLIN procedure is to estimate the parameters in a model in which the response variable is a nonlinear function of one or more of the parameters.

Least Squares Model

The Michaelis-Menten model of enzyme kinetics (Ratkowsky 1990, p. 59) relates a substrate's concentration to its catalyzed reaction rate. The Michaelis-Menten model can be analyzed using a least squares estimation because it does not specify how the reaction rate is distributed about its predicted value. The relationship between reaction rate and substrate concentration is

$$f(\mathbf{x}, \boldsymbol{\theta}) = \frac{\theta_1 x_i}{\theta_2 + x_i}, \quad \text{for } i = 1, 2, \ldots, n$$

where x_i represents the concentration for n trials and $f(\mathbf{x}, \boldsymbol{\theta})$ is the reaction rate. The vector $\boldsymbol{\theta}$ contains the rate parameters.

For this model with experimental measurements of reaction rate and concentration stored in the enzyme data set, the following SAS statements estimate the parameters θ_1 and θ_2:

```
proc hpnlin data=enzyme;
   parms theta1=0 theta2=0;
   model rate ~ residual(theta1*conc / (theta2 + conc));
run;
```

The least squares estimation performed by PROC HPNLIN for this enzyme kinetics problem produces the analysis of variance table displayed in Figure 10.1. The table displays the degrees of freedom, sums of squares, and mean squares along with the model F test.

Figure 10.1 Nonlinear Least Squares Analysis of Variance

```
                              The HPNLIN Procedure

                              Analysis of Variance

                                     Sum of        Mean                Approx
   Source                     DF    Squares      Square    F Value    Pr > F

   Model                       2     290116      145058    88537.2    <.0001
   Error                      12    19.6606      1.6384
   Uncorrected Total          14     290135

                    An intercept was not specified for this model.
```

Finally, Figure 10.2 displays the parameter estimates, standard errors, *t* statistics, and 95% confidence intervals for θ_1 and θ_2.

Figure 10.2 Parameter Estimates and Approximate 95% Confidence Intervals

```
                              Parameter Estimates

                         Standard                                Approximate 95%
   Parameter  Estimate      Error    DF  t Value  Pr > |t|     Confidence Limits

   theta1       158.10     0.6737    12   234.67    <.0001     156.64    159.57
   theta2      0.07413   0.003129    12    23.69    <.0001    0.06731   0.08095
```

In the enzyme kinetics model, no information was supplied concerning the distribution of the reaction rate about the model's mean value. Therefore, the *residual* model distribution was specified to perform a least squares parameter fit.

Binomial Model

In Example 62.3 (*SAS/STAT User's Guide*) cancer remission is modeled by expressing the maximum likelihood function for a binary distribution as a nonlinear least squares optimization in PROC NLIN. The following statements show an equivalent formulation of this model that uses PROC HPNLIN and specifies the binary distribution explicitly:

```
proc hpnlin data=remiss corr;
   parms int=-10 a=-2 b=-1 c=6;
   linp = int + a*cell + b*li + c*temp;
   p = probnorm(linp);
   model remiss ~ binary(1-p);
run;
```

This binary distribution model displays information about the quality of the estimation that is different from the information displayed in the section "Least Squares Model" on page 253. No analysis of variance table is produced for this model, and fit statistics based on the value of the likelihood function are displayed in Figure 10.3.

Figure 10.3 Nonlinear Likelihood Function Statistics

```
                           The HPNLIN Procedure

                              Fit Statistics

                 -2 Log Likelihood                  21.9002
                 AIC (smaller is better)            29.9002
                 AICC (smaller is better)           31.7183
                 BIC (smaller is better)            35.0835
```

Parameter estimates for the binary distribution model are displayed in Figure 10.4 using the same quantities as are used in the section "Least Squares Model" on page 253.

Figure 10.4 Parameter Estimates and Approximate 95% Confidence Intervals

```
                           Parameter Estimates

                       Standard                                Approximate 95%
   Parameter  Estimate    Error   DF  t Value  Pr > |t|     Confidence Limits

   int        -36.7548  32.3607   27    -1.14    0.2660    -103.15    29.6439
   a           -5.6298   4.6376   27    -1.21    0.2353   -15.1454     3.8858
   b           -2.2513   0.9790   27    -2.30    0.0294    -4.2599    -0.2426
   c           45.1815  34.9095   27     1.29    0.2065   -26.4469     116.81
```

Syntax: HPNLIN Procedure

The following statements are available in the HPNLIN procedure:

PROC HPNLIN < *options* > ;
  BOUNDS *inequality* < *,. . . ,inequality* > ;
  ESTIMATE *'label' expression* < *options* > ;
  MODEL *model specification* ;
  PARAMETERS < *parameter-specification* > < *,. . . , parameter-specification* > ;
  PERFORMANCE < *performance-options* > ;
  PREDICT *'label' expression keyword=names* < *. . . keyword=names* > < *options* > ;
  RESTRICT *restriction1* < *, restriction2 . . .* > ;
  Programming Statements ;

The PROC HPNLIN statement and exactly one MODEL statement are required.

PROC HPNLIN Statement

PROC HPNLIN < *options* > **;**

The PROC HPNLIN statement invokes the procedure. Table 10.1 summarizes important options in the PROC HPNLIN statement by function. These and other options in the PROC HPNLIN statement are then described fully in alphabetical order.

Table 10.1 PROC HPNLIN Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
OUT=	Specifies the output data set
Options Related to Output	
CORR	Specifies the correlation matrix
COV	Specifies the covariance matrix
ECORR	Specifies the correlation matrix of additional estimates
ECOV	Specifies the covariance matrix of additional estimates
DF	Specifies the default degrees of freedom
NOPRINT	Suppresses ODS output
NOITPRINT	Suppresses output concerning iterations within the optimization process
Options Related to Optimization	
ABSCONV=	Tunes an absolute function convergence criterion
ABSFCONV=	Tunes an absolute difference function convergence criterion
ABSGCONV=	Tunes the absolute gradient convergence criterion
FCONV=	Tunes the relative function convergence criterion
GCONV=	Tunes the relative gradient convergence criterion
MAXITER=	Chooses the maximum number of iterations in any optimization
MAXFUNC=	Specifies the maximum number of function evaluations in any optimization
MAXTIME=	Specifies the upper limit seconds of CPU time for any optimization
MINITER=	Specifies the minimum number of iterations in any optimization
TECHNIQUE=	Selects the optimization technique
Tolerances	
SINGULAR=	Tunes the general singularity criterion
User-defined Formats	
FMTLIBXML=	Specifies a file reference for a format stream
XMLFORMAT=	Specifies a file name for a format stream

You can specify the following options in the PROC HPNLIN statement.

ABSCONV=*r*

ABSTOL=*r*

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

ABSFCONV=*r* < *n*>

ABSFTOL=*r*< *n*>

specifies an absolute difference function convergence criterion. For all techniques except the Nelder-Mead simplex (NMSIMP) technique, termination requires a small change of the function value in successive iterations:

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

Here, $\boldsymbol{\psi}$ denotes the vector of parameters that participate in the optimization, and $f(\cdot)$ is the objective function. The same formula is used for the NMSIMP technique, but $\boldsymbol{\psi}(k)$ is defined as the vertex with the lowest function value, and $\boldsymbol{\psi}^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is $r = 0$. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

ABSGCONV=*r* < *n*>

ABSGTOL=*r*< *n*>

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

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

Here, $\boldsymbol{\psi}$ denotes the vector of parameters that participate in the optimization, and $g_j(\cdot)$ is the gradient of the objective function with respect to the jth parameter. This criterion is not used by the NMSIMP technique. The default value is $r =$1E−5. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

ALPHA=*α*

specifies the level of significance α used in the construction of $100(1-\alpha)\%$ confidence intervals. The value must be strictly between 0 and 1; the default value of $\alpha = 0.05$ results in 95% intervals. This value is used as the default confidence level for limits computed in the “Parameter Estimates” table and is used in the LOWER and UPPER options in the PREDICT statement.

CORR

requests the approximate correlation matrix for the parameter estimates.

COV

requests the approximate covariance matrix for the parameter estimates.

DATA=*SAS-data-set*

names the SAS data set to be used by PROC HPNLIN. The default is the most recently created data set.

If the procedure executes in MPP mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. In the latter case, the procedure reads the data alongside the distributed database. For more information about the various execution modes, see the section "SMP and MPP Modes" on page 10; for more information about on the alongside-the-database model, see the section "Alongside-the-Database Execution" on page 15.

DF=*n*

specifies the default number of degrees of freedom to use in the calculation of p-values and confidence limits for additional parameter estimates.

ECORR

requests the approximate correlation matrix for all expressions that are specified in ESTIMATE statements.

ECOV

requests the approximate covariance matrix for all expressions that are specified in ESTIMATE statements.

FCONV=*r< n>*

FTOL=*r< n>*

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

$$\frac{|f(\boldsymbol{\psi}^{(k)}) - f(\boldsymbol{\psi}^{(k-1)})|}{|f(\boldsymbol{\psi}^{(k-1)})|} \le r$$

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

The default is $r = 10^{-\text{FDIGITS}}$, where FDIGITS is by default $-\log_{10}\{\epsilon\}$ and ϵ is the machine precision. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

FMTLIBXML=*file-ref*

specifies the file reference for the XML stream that contains the user-defined format definitions. User-defined formats are handled differently in a distributed computing environment than they are handled in other SAS products. See the section "Working with Formats" on page 18 for details about how to generate a XML stream for your formats.

GCONV=*r< n>*

GTOL=*r< n>*

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

$$\frac{\mathbf{g}(\boldsymbol{\psi}^{(k)})'[\mathbf{H}^{(k)}]^{-1}\mathbf{g}(\boldsymbol{\psi}^{(k)})}{|f(\boldsymbol{\psi}^{(k)})|} \le r$$

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

$$\frac{\| \mathbf{g}(\boldsymbol{\psi}^{(k)}) \|_2^2 \quad \| \mathbf{s}(\boldsymbol{\psi}^{(k)}) \|_2}{\| \mathbf{g}(\boldsymbol{\psi}^{(k)}) - \mathbf{g}(\boldsymbol{\psi}^{(k-1)}) \|_2 \, |f(\boldsymbol{\psi}^{(k)})|} \le r$$

This criterion is not used by the NMSIMP technique. The default value is r =1E−8. The optional integer value *n* specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

MAXFUNC=*n*

MAXFU=*n*

specifies the maximum number *n* of function calls in the optimization process. The default values are as follows, depending on the optimization technique (which you specify in the TECHNIQUE= option):

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

Optimization can terminate only after completing a full iteration. Therefore, the number of function calls that are actually performed can exceed the number that is specified by the MAXFUNC= option.

MAXITER=*n*

MAXIT=*n*

specifies the maximum number *n* of iterations in the optimization process. The default values are as follows, depending on the optimization technique (which you specify in the TECHNIQUE= option):

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

These default values also apply when *n* is specified as a missing value.

MAXTIME=*r*

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

MINITER=*n*

MINIT=*n*

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

NOITPRINT

suppresses the display of the "Iteration History" table.

NOPRINT

suppresses the generation of ODS output.

OUT=*SAS-data-set*

names the SAS data set to be created when one or more PREDICT statements are specified. A single OUT= data set is created to contain all predicted values when more than one PREDICT statement is specified. An error message is produced if a PREDICT statement is specified and an OUT= data set is not specified. See the section "Output Data Sets" on page 18 for more information about output data sets in SAS High-Performance Analytics procedures.

SINGULAR=*number*

tunes the general singularity criterion applied by the HPNLIN procedure in sweeps and inversions. The default is 1E4 times the machine epsilon; this product is approximately 1E−12 on most computers.

TECHNIQUE=*keyword*

TECH=*keyword*

specifies the optimization technique for obtaining maximum likelihood estimates. You can choose from the following techniques by specifying the appropriate *keyword*:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
LEVMAR	performs a Levenberg-Marquardt optimization.
NEWRAP	performs a Newton-Raphson optimization that combines a line-search algorithm with ridging.
NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	performs no optimization.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a quasi-Newton optimization.
TRUREG	performs a trust-region optimization.

The default value is TECHNIQUE=LEVMAR for least squares regression models and TECHNIQUE=NRRIDG for models where the distribution is specified.

XMLFORMAT=*filename*

specifies the file name for the XML stream that contains the user-defined format definitions. User-defined formats are handled differently in a distributed computing environment than they are handled in other SAS products. See the section "Working with Formats" on page 18 for details about how to generate a XML stream for your formats.

BOUNDS Statement

BOUNDS *constraint* < *, constraint . . .* > **;**

where *constraint* represents

< *number operator* > *parameter-list* < *operator number* >

Boundary constraints are specified with a BOUNDS statement. One- or two-sided boundary constraints are allowed. The list of boundary constraints are separated by commas. For example:

```
bounds 0 <= a1-a9 X <= 1, -1 <= c2-c5;
bounds b1-b10 y >= 0;
```

You can specify more than one BOUNDS statement. If you specify more than one lower (or upper) bound for the same parameter, the maximum (or minimum) of these is taken.

If the maximum l_j of all lower bounds is larger than the minimum of all upper bounds u_j for the same parameter θ_j, the boundary constraint is replaced by $\theta_j := l_j := \min(u_j)$ defined by the minimum of all upper bounds specified for θ_j.

ESTIMATE Statement

ESTIMATE *'label' expression* < *options* > **;**

The ESTIMATE statement enables you to compute an additional estimate that is a function of the parameter values. You must provide a quoted string to identify the estimate and then provide a valid SAS expression. Multiple ESTIMATE statements are permitted, and results from all statements are listed in a common table. PROC HPNLIN computes approximate standard errors for the estimates by using the delta method (Billingsley 1986). It uses these standard errors to compute corresponding t statistics, p-values, and confidence limits.

The ECOV option in the PROC HPNLIN statement produces a table that contains the approximate covariance matrix of all the additional estimates you specify. The ECORR option produces the corresponding correlation matrix.

The following options are available in the ESTIMATE statement:

ALPHA=α

specifies the alpha level to be used in computing confidence limits. The default value corresponds to the ALPHA= option in the PROC HPNLIN statement.

DF=*d*

specifies the degrees of freedom to be used in computing *p*-values and confidence limits. The default value corresponds to the DF= option in the PROC HPNLIN statement.

MODEL Statement

MODEL *dependent-variable* ∼ *distribution* **;**

The MODEL statement is the mechanism for either using a distribution specification to specify the distribution of the data or using the RESIDUAL distribution to specify a predicted value. You must specify a single dependent variable from the input data set, a tilde (∼), and then a distribution with its parameters. You can specify the following values for *distribution*:

RESIDUAL(*m*) or **LS**(*m*)	specifies no particular distribution. Instead minimize the sum of squares of the differences between *m* and the dependent variable.
NORMAL(*m*, *v*)	specifies a normal (Gaussian) distribution with mean *m* and variance *v*.
BINARY(*p*)	specifies a binary (Bernoulli) distribution with probability *p*.
BINOMIAL(*n*, *p*)	specifies a binomial distribution with count *n* and probability *p*.
GAMMA(*a*, *b*)	specifies a gamma distribution with shape *a* and scale *b*.
NEGBIN(*n*, *p*)	specifies a negative binomial distribution with count *n* and probability *p*.
POISSON(*m*)	specifies a Poisson distribution with mean *m*.
GENERAL(*ll*)	specifies a general log-likelihood function that you construct by using SAS programming statements.

The MODEL statement must follow any SAS programming statements you specify for computing parameters of the preceding distributions. See the section "Built-in Log-Likelihood Functions" on page 270 for expressions of the built-in log-likelihood functions.

PARAMETERS Statement

PARAMETERS < *parameter-specification* > < *,. . . , parameter-specification* > < */ options* > **;**

PARMS < *parameter-specification* > < *,. . . , parameter-specification* > < */ options* > **;**

The purpose of the PARAMETERS (PARMS) statement is to provide starting values for the HPNLIN procedure. You can provide values that define a single point in the parameter space or a set of points. For more information about *parameter-specification*, see the section "Assigning Starting Values with Parameter-Specification" on page 263.

You can specify the following *options* in the PARMS statement after the slash (/).

BEST=$i > 0$
specifies the maximum number of parameter grid points and corresponding objective function values to display in the “Parameters” table. If the BEST= option is specified, parameter grid points are listed in ascending order of objective function value. By default, all parameter grid points are displayed.

PDATA=*SAS-data-set*
DATA=*SAS-data-set*
specifies the data set used to provide parameter starting values.

START=*value*
DEFSTART=*value*
specifies a default starting value for all parameters.

There are four methods available for providing starting values to the optimization process. In descending order of precedence, the methods are as follows:

1. Specify values directly in the PARMS statement.
2. Specify values in the PARMS / PDATA= data set.
3. Specify a single value for all parameters by using the START= option.
4. Use the default value 1.0.

The names assigned to parameters must be valid SAS names and must not coincide with names of variables in the input data set (see the DATA= option in the PROC HPNLIN statement). Parameters that are assigned starting values through the PARAMETERS statement can be omitted from the estimation if the expression in the MODEL statement does not depend on them.

Assigning Starting Values with Parameter-Specification

A *parameter-specification* has the following general form where *name* identifies the parameter and *value-list* provides the set of starting values for the parameter:

name **=** *value-list*

Very often the *value-list* contains only a single value, but more general and flexible list specifications such as the following are possible:

m	a single value
m1, *m2*, ..., *mn*	several values
m TO *n*	a sequence in which *m* equals the starting value, *n* equals the ending value, and the increment equals 1
m TO *n* BY *i*	a sequence in which *m* equals the starting value, *n* equals the ending value, and the increment is *i*
m1, *m2* TO *m3*	mixed values and sequences

When you specify more than one value for a parameter, PROC HPNLIN sorts the values in ascending sequence and removes duplicate values from the parameter list before forming the grid for the parameter search. If you specify several values for each parameter, PROC HPNLIN evaluates the model at each point on the grid. The iterations then commence from the point on the grid that yields the smallest objective function value.

For example, the following PARMS statement specifies five parameters and sets their possible starting values as shown in the table:

```
parms  b0 = 0
       b1 = 4 to 8
       b2 = 0 to .6 by .2
       b3 = 1, 10, 100
       b4 = 0, .5, 1 to 4;
```

Possible Starting Values

B0	B1	B2	B3	B4
0	4	0.0	1	0.0
	5	0.2	10	0.5
	6	0.4	100	1.0
	7	0.6		2.0
	8			3.0
				4.0

The objective function values are calculated for each of the $1 \times 5 \times 4 \times 3 \times 6 = 360$ combinations of possible starting values. Each grid point's objective function value is computed by using the execution mode specified in the PERFORMANCE statement.

If you specify a starting value with a *parameter-specification*, any starting values provided for this parameter through the PDATA= data set are ignored. The *parameter-specification* overrides the information in the PDATA= data set.

Assigning Starting Values from a SAS Data Set

The PDATA= option in the PARAMETERS statement enables you to assign starting values for parameters through a SAS data set. The data set must contain at least two variables: a character variable named Parameter or Parm that identifies the parameter and a numeric variable named Estimate or Est that contains the starting values. For example, the PDATA= option enables you to use the contents of the "ParameterEstimates" table from one PROC HPNLIN run to supply starting values for a subsequent run, as follows:

```
proc hpnlin data=d(obs=30);
   parameters alpha=100 beta=3 gamma=4;
   Switch = 1/(1+gamma*exp(beta*log(dose)));
   model y ~ residual(alpha*Switch);
   ods output ParameterEstimates=pest;
run;

proc hpnlin data=d;
```

```
      parameters / pdata=pest;
      Switch = 1/(1+gamma*exp(beta*log(dose)));
      model y ~ residual(alpha*Switch);
   run;
```

You can specify multiple values for a parameter in the PDATA= data set, and the parameters can appear in any order. The starting values are collected by parameter and arranged in ascending order, and duplicate values are removed. The parameter names in the PDATA= data set are not case sensitive. For example, the following DATA step defines starting values for three parameters and a starting grid with $1 \times 3 \times 1 = 3$ points:

```
   data test;
      input Parameter $ Estimate;
      datalines;
   alpha  100
    BETA    4
    beta    4.1
   beta     4.2
   beta     4.1
    gamma  30
   ;
```

PERFORMANCE Statement

PERFORMANCE < *performance-options*> **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

With the PERFORMANCE statement, you can also control whether a SAS High-Performance Analytics procedure executes in symmetric multiprocessing or massively parallel processing mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

PREDICT Statement

PREDICT *'label' expression* < *options*> **;**

PREDICT *'label'* **MEAN** < *options*> **;**

The PREDICT statement enables you to construct predictions of an expression across all of the observations in the input data set. Multiple PREDICT statements are permitted. You must provide a quoted string to identify the predicted expression and then provide the predicted value. You can specify the predicted value either with a SAS programming expression that involves the input data set variables and parameters or by using the keyword MEAN. If you specify the keyword MEAN, the predicted mean value for the distribution specified in the MODEL statement is used. Predicted values are computed using the final parameter estimates. Standard errors of prediction are computed using the delta method (Billingsley 1986; Cox 1998). Results for all PREDICT statements are placed in the output data set that you specify with the OUT= option in the PROC HPNLIN statement. See the section "Output Data Sets" on page 18 for more details about output data sets in High-Performance Analytics procedures.

The following options are available in the PREDICT statement.

ALPHA=*α*

specifies the alpha level to be used in computing confidence limits. The default value corresponds to the ALPHA= option in the PROC HPNLIN statement.

DF=*d*

specifies the degrees of freedom to be used in computing confidence limits. The default value corresponds to the DF= option in the PROC HPNLIN statement.

LOWER=*name*

specifies a variable that contains the lower confidence limit of the predicted value.

PRED=*name*

specifies a variable that contains the predicted value.

PROBT=*name*

specifies a variable that contains the p-value of the predicted value.

STDERR=*name*

specifies a variable that contains the standard error of the predicted value.

TVALUE=*name*

specifies a variable that contains the t statistic for the predicted value.

UPPER=*name*

specifies a variable that contains the upper confidence limit of the predicted value.

RESTRICT Statement

RESTRICT *restriction1* < , *restriction2* . . . > **;**

The RESTRICT statement imposes linear restrictions on the model's parameters estimates. You can specify any number of RESTRICT statements.

Each *restriction* is written as an expression, optionally followed by an equality operator (=) or an inequality operator (<, >, <=, >=), followed by a second expression as follows:

expression **<** *operator expression* **>**

The *operator* can be =, <, >, <= , or >=. The operator and second expression are optional. When they are omitted, the *operator* defaults to = and the second *expression* defaults to the value 0.

Restriction expressions can be composed of parameter names, arithmetic operators, functions, and constants. Comparison operators (such as = or <) and logical operators (such as &) cannot be used in RESTRICT statement expressions. Parameters named in restriction expressions must be among the parameters estimated by the model. Restriction expressions cannot refer to other variables defined in the program or the DATA= data set. The restriction expressions must be linear functions of the parameters.

The following example illustrates how to use the RESTRICT statement to impose a linear constraint on parameters:

```
 proc hpnlin;
    parms alpha beta;
    f = (x/alpha + beta)**2
    model y ~ residual(f);
    restrict beta < 2*(alpha + constant('pi'));
run;
```

The preceding RESTRICT statement represents the following model constraint:

$$\beta < 2(\alpha + \pi)$$

Programming Statements

Programming statements define the arguments of the MODEL, ESTIMATE, and PREDICT statements in PROC HPNLIN. Most of the programming statements that can be used in the SAS DATA step can also be used in the HPNLIN procedure. See *SAS Language Reference: Concepts* for a description of SAS programming statements. The following are valid programming statements:

ABORT;
CALL *name* **[(** *expression* **[,** *expression* **. . .])];**
DELETE;
DO[*variable* **=** *expression*
  [TO *expression***] [BY** *expression***]**
  [, *expression* **[TO** *expression***] [BY** *expression* **] . . .]**
  ]
  [WHILE *expression* **] [UNTIL** *expression* **] ;**
END;
GOTO *statement_label***;**
IF *expression***;**
IF *expression* **THEN** *program_statement***;**
  ELSE *program_statement***;**
variable **=** *expression***;**
variable **+** *expression***;**
LINK *statement_label***;**
PUT [*variable***] [=] [...];**
RETURN;
SELECT[(*expression***)];**
STOP;
SUBSTR(*variable***,** *index***,** *length* **)=** *expression***;**
WHEN (*expression***)** *program_statement***;**
    OTHERWISE *program_statement***;**

For the most part, the SAS programming statements work the same as they do in the SAS DATA step, as documented in *SAS Language Reference: Concepts*. However, they differ as follows differences:

- The ABORT statement does not allow any arguments.
- The DO statement does not allow a character index variable. Thus

  ```
  do i = 1,2,3;
  ```

  is supported, but the following statement is not supported:

  ```
  do i = 'A','B','C';
  ```

- In contrast to other procedures that share PROC HPNLIN's programming syntax, PROC HPNLIN does not support the LAG function. Because observations are not processed sequentially when High-Performance Analytics procedures perform the parameter optimization, information for computing lagged values is not available.
- The PUT statement, used mostly for program debugging in PROC HPNLIN, supports only some of the features of the DATA step PUT statement, and it has some new features that the DATA step PUT statement does not have:
  - The PROC HPNLIN PUT statement does not support line pointers, factored lists, iteration factors, overprinting, _INFILE_, the colon (:) format modifier, or "$".

  - The PROC HPNLIN PUT statement supports expressions, but the expression must be enclosed in parentheses. For example, the following statement displays the square root of x:

    ```
    put  (sqrt(x));
    ```

  - The PROC HPNLIN PUT statement supports the item _PDV_ to display a formatted listing of all variables in the program. For example, the following statement displays a much more readable listing of the variables than the _ALL_ print item:

    ```
    put _pdv_;
    ```

- The WHEN and OTHERWISE statements enable you to specify more than one target statement. That is, DO/END groups are not necessary for multiple statement WHENs. For example, the following syntax is valid:

```
select;
   when (exp1) stmt1;
               stmt2;
   when (exp2) stmt3;
               stmt4;
end;
```

When you code your programming statements, avoid defining variables that begin with an underscore (_) because they might conflict with internal variables that are created by PROC HPNLIN. The MODEL statement must come after any SAS programming statements that define or modify terms that are used to specify the model.

Details: HPNLIN Procedure

Least Squares Estimation

The most general models estimated by PROC HPNLIN can be represented using the equations

$$\begin{aligned}\mathbf{Y} &= \mathbf{f}(\boldsymbol{\beta}; \mathbf{z}_1, \cdots, \mathbf{z}_k) + \boldsymbol{\epsilon} \\ \mathrm{E}[\boldsymbol{\epsilon}] &= \mathbf{0} \\ \mathrm{Var}[\boldsymbol{\epsilon}] &= \sigma^2 \mathbf{I}\end{aligned}$$

where

$\mathbf{Y}$	is the $(n \times 1)$ vector of observed responses
$\mathbf{f}$	is the nonlinear prediction function of parameters and regressor variables
$\boldsymbol{\beta}$	is the vector of model parameters to be estimated

$\mathbf{z}_1, \cdots, \mathbf{z}_k$	are the $(n \times 1)$ vectors for each of the k regressor variables
$\boldsymbol{\epsilon}$	is the $(n \times 1)$ vector of residuals
σ^2	is the variance of the residuals

In these models, the distribution of the residuals is not specified and the model parameters are estimated using the least squares method. For the standard errors and confidence limits in the "ParameterEstimates" table to apply, the errors are assumed to be homoscedastic, uncorrelated, and have zero mean.

Built-in Log-Likelihood Functions

For models in which the distribution of model errors is specified, the HPNLIN procedure estimates parameters by maximizing the value of a log-likelihood function for the specified distribution. The log-likelihood functions used by PROC HPNLIN for the supported error distributions are as follows:

$Y \sim \text{normal}(m, v)$

$$\begin{aligned} l(m, v; y) &= -\frac{1}{2}\left(\log\{2\pi\} + \frac{(y-m)^2}{v} + \log\{v\}\right) \\ \text{E}[Y] &= m \\ \text{Var}[Y] &= v \\ v &> 0 \end{aligned}$$

$Y \sim \text{binary}(p)$

$$\begin{aligned} l_1(p; y) &= \begin{cases} y \ \log\{p\} & y > 0 \\ 0 & \text{otherwise} \end{cases} \\ l_2(p; y) &= \begin{cases} (1-y) \ \log\{1-p\} & y < 1 \\ 0 & \text{otherwise} \end{cases} \\ l(p; y) &= l_1(p; y) + l_2(p; y) \\ \text{E}[Y] &= p \\ \text{Var}[Y] &= p\,(1-p) \\ 0 < p &< 1 \end{aligned}$$

$Y \sim \text{binomial}(n, p)$

$$\begin{aligned} l_c &= \log\{\Gamma(n+1)\} - \log\{\Gamma(y+1)\} - \log\{\Gamma(n-y+1)\} \\ l_1(n, p; y) &= \begin{cases} y \ \log\{p\} & y > 0 \\ 0 & \text{otherwise} \end{cases} \\ l_2(n, p; y) &= \begin{cases} (n-y) \ \log\{1-p\} & n-y > 0 \\ 0 & \text{otherwise} \end{cases} \\ l(n, p; y) &= l_c + l_1(n, p; y) + l_2(n, p; y) \\ \text{E}[Y] &= n\,p \\ \text{Var}[Y] &= n\,p\,(1-p) \\ 0 < p &< 1 \end{aligned}$$

$Y \sim \text{gamma}(a, b)$

$$\begin{aligned} l(a, b; y) &= -a \log\{b\} - \log\{\Gamma(a)\} + (a-1)\log\{y\} - y/b \\ \mathrm{E}[Y] &= ab \\ \mathrm{Var}[Y] &= ab^2 \\ a &> 0 \\ b &> 0 \end{aligned}$$

This parameterization of the gamma distribution differs from the parameterization used in the GLIMMIX and GENMOD procedures. The scale parameter in PROC HPNLIN is expressed as the inverse of the scale parameter in PROC GLIMMIX and PROC GENMOD. The PROC HPNLIN parameter represents the scale of the magnitude of the residuals. The scale parameter in PROC GLIMMIX can be estimated by using the following statements:

```
proc glimmix;
   model y = x / dist=gamma s;
run;
```

The following statements show how to estimate the equivalent scale parameter using PROC HPNLIN:

```
proc hpnlin;
   parms b0=1 b1=0 scale=14;
   linp = b0 + b1*x;
   mu   = exp(linp);
   b    = mu*scale;
   model y ~ gamma(1/scale,b);
run;
```

$Y \sim \text{negbin}(n, p)$

$$\begin{aligned} l(n, p; y) &= \log\{\Gamma(n+y)\} - \log\{\Gamma(n)\} - \log\{\Gamma(y+1)\} \\ &\quad + n\log\{p\} + y\log\{1-p\} \\ \mathrm{E}[Y] &= n\left(\frac{1-p}{p}\right) \\ \mathrm{Var}[Y] &= n\left(\frac{1-p}{p^2}\right) \\ n &\geq 0 \\ 0 &< p < 1 \end{aligned}$$

The parameter n can be real-numbered; it does not have to be integer-valued.

$Y \sim \text{Poisson}(m)$

$$\begin{aligned} l(m; y) &= y\log\{m\} - m - \log\{\Gamma(y+1)\} \\ \mathrm{E}[Y] &= m \\ \mathrm{Var}[Y] &= m \\ m &> 0 \end{aligned}$$

Computational Method

Multithreading

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

The number of threads spawned by the HPNLIN procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statement, the HPNLIN procedure determines threading as if it executed on a system with four CPUs, regardless of the actual CPU count:

  ```
  options cpucount=4;
  ```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This specification overrides the system option. Specify NTHREADS=1 to force single-threaded execution.

- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement. Specifying this option overrides the CPUCOUNT system option and instructs the HPNLIN procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output. The HPNLIN procedure allocates one thread per CPU.

The HPNLIN procedure divides the data processed on a single machine among the threads—that is, the HPNLIN procedure implements multithreading by parallelizing computations across the data. For example, if the input data set has $1,000$ observations and you are running with four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. These operations include the following:

- calculation of objective function values for the initial parameter grid
- objective function calculation
- gradient calculation
- Hessian calculation
- scoring of observations

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

Choosing an Optimization Algorithm

First- or Second-Order Algorithms

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

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

Table 10.3 shows which derivatives are required for each optimization technique.

Table 10.3 Derivatives Required

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

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

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

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

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

The LEVMAR method is only appropriate for least squares optimization problems.

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

By default, the HPNLIN procedure applies the NRRIDG algorithm because it can take advantage of mutli-threading in Hessian computations and inversions. If the number of parameters becomes large, specifying TECHNIQUE=QUANEW, which is a first-order method with good overall properties, is recommended.

Algorithm Descriptions

The following subsections provide details about each optimization technique and follow the same order as Table 10.3.

Trust Region Optimization (TRUREG)

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

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

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

Newton-Raphson Optimization with Line Search (NEWRAP)

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

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

In each iteration, a line search is performed along the search direction to find an approximate optimum of the objective function. The default line-search method uses quadratic interpolation and cubic extrapolation (LIS=2).

Newton-Raphson Ridge Optimization (NRRIDG)

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

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

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

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

Quasi-Newton Optimization (QUANEW)

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

The QUANEW technique implemented by the HPNLIN procedure is the dual quasi-Newton algorithm, which updates the Cholesky factor of an approximate Hessian.

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

The QUANEW algorithm uses its own line-search technique.

Double-Dogleg Optimization (DBLDOG)

The double-dogleg optimization method combines the ideas of the quasi-Newton and trust region methods. In each iteration, the double-dogleg algorithm computes the step $\mathbf{s}^{(k)}$ as the linear combination of the steepest descent or ascent search direction $\mathbf{s}_1^{(k)}$ and a quasi-Newton search direction $\mathbf{s}_2^{(k)}$:

$$\mathbf{s}^{(k)} = \alpha_1 \mathbf{s}_1^{(k)} + \alpha_2 \mathbf{s}_2^{(k)}$$

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

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

Conjugate Gradient Optimization (CONGRA)

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

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

Nelder-Mead Simplex Optimization (NMSIMP)

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

The original Nelder-Mead simplex algorithm is implemented and extended to boundary constraints. This algorithm does not compute the objective for infeasible points, but it changes the shape of the simplex adapting to the nonlinearities of the objective function, which contributes to an increased speed of convergence. It uses a special termination criterion.

Displayed Output

The following sections describe the output that PROC HPNLIN produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Performance Information

The “Performance Information” table is produced by default and displays information about the grid host for distributed execution and about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The numbers of compute nodes and threads are also displayed, depending on the environment.

Specifications

The “Specifications” table displays basic information about the model such as the data source, the dependent variable, the distribution being modeled, and the optimization technique.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data set and the number of observations used in the analysis.

Dimensions

The “Dimensions” table displays the number of parameters estimated in the model and the number of upper and lower bounds which are imposed on the parameters.

Parameters

The “Parameters” table displays the initial values of parameters used to start the estimation process. This information can be limited by using the BEST= option in the PARAMETERS statement when a large number of initial parameter value combinations are specified. The parameter combinations and their corresponding objective function values are listed in increasing order of objective function value.

Iteration History

The “Iteration History” table displays for each iteration of the optimization the number of function evaluations (including gradient and Hessian evaluations), the value of the objective function, the change in the objective function from the previous iteration, and the absolute value of the largest (projected) gradient element.

Convergence Status

The convergence status table is a small ODS table that follows the “Iteration History” table in the default output. In the listing it appears as a message that identifies whether the optimization succeeded and which convergence criterion was met. If the optimization fails, the message indicates the reason for the failure. If you save the convergence status table to an output data set, a numeric Status variable is added that enables you to assess convergence programmatically. The values of the Status variable encode the following:

0	Convergence was achieved or an optimization was not performed (because of TECHNIQUE=NONE).
1	The objective function could not be improved.
2	Convergence was not achieved because of a user interrupt or because a limit was exceeded, such as the maximum number of iterations or the maximum number of function evaluations. To modify these limits, see the MAXITER=, MAXFUNC=, and MAXTIME= options in the PROC HPNLIN statement.
3	Optimization failed to converge because function or derivative evaluations failed at the starting values or during the iterations or because a feasible point that satisfies the parameter constraints could not be found in the parameter space.

Linear Constraints

The "Linear Constraints" table summarizes the linear constraints that were applied to the model using the RESTRICT statements. All the constraints specified in the model are listed in the "Linear Constraints" table together with information about whether each constraint represents an inequality or equality condition and whether that constraint is active for the final parameter estimates.

Fit Statistics

The "Fit Statistics" table displays a variety of measures of fit depending on whether the model was estimated using least squares or maximum likelihood. In both cases, smaller values of the fit statistics indicate better fit.

For least squares estimations, the "Fit Statistics" table displays the sum of squares of errors and the variance of errors.

For maximum likelihood estimations, the table displays information criteria using the following formulas, where p denotes the number of effective parameters, n denotes the number of observations used, and l is the log likelihood evaluated at the converged estimates

$$\text{AIC} = -2l + 2p$$

$$\text{AICC} = \begin{cases} -2l + 2pn/(n-p-1) & f > p+2 \\ -2l + 2p(p+2) & \text{otherwise} \end{cases}$$

$$\text{BIC} = -2l + p\log(f)$$

The information criteria values displayed in the "Fit Statistics" table are not based on a normalized log-likelihood function.

ANOVA

The "Analysis of Variance" table is displayed only for least squares estimations. The ANOVA table displays the number of degrees of freedom and the sum of squares attributed to the model, the error, and the total.

The ANOVA table also reports the variance of the model and the errors, the F statistic, and its probability for the model.

Parameter Estimates

The parameter estimates, their estimated (asymptotic) standard errors t statistics, and associated p-values for the hypothesis that the parameter is 0 are presented in the “Parameter Estimates” table. Confidence limits are displayed for each parameter based on the value of the ALPHA= option in the PROC HPNLIN statement.

Additional Estimates

The “Additional Estimates” table displays identical information to the “Parameter Estimates” table for the expressions that appear in the optional ESTIMATE statements. The table is generated when one or more ESTIMATE statements are specified. Because a separate ALPHA= option can be specified for each ESTIMATE statement, the “Additional Estimates” table also includes a column that indicates each confidence interval’s corresponding significance level.

Covariance

The “Covariance” table appears when the COV option is specified in the PROC HPNLIN statement. The “Covariance” table displays a matrix of covariances between each pair of estimated parameters.

Correlation

The “Correlation” table appears when the CORR option is specified in the PROC HPNLIN statement. The “Correlation” table displays the correlation matrix for the estimated parameters.

Additional Estimates Covariance

The “Covariance of Additional Estimates” table appears when the ECOV option is specified in the PROC HPNLIN statement. The “Covariance of Additional Estimates” table displays a matrix of covariances between each pair of expressions specified in ESTIMATE statements.

Additional Estimates Correlation

The “Correlation of Additional Estimates” table appears when the ECORR option is specified in the PROC HPNLIN statement. The “Correlation of Additional Estimates” table displays the correlation matrix for the expressions specified in ESTIMATE statements.

Procedure Task Timing

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a "Procedure Task Timing" table in which elapsed times (absolute and relative) for the main tasks of the procedure are displayed.

ODS Table Names

Each table created by the HPNLIN procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 10.4.

Table 10.4 ODS Tables Produced by PROC HPNLIN

Table Name	Description	Required Statement / Option
AdditionalEstimates	Functions of estimated parameters and their associated statistics	ESTIMATE statement
ANOVA	Least squares analysis of variance information	RESIDUAL distribution in the MODEL statement
Constraints	Information about the model's linear constraints	RESTRICT statement
ConvergenceStatus	Optimization success and convergence information	Default output
CorrB	Parameter correlation matrix	CORR option in the PROC HPNLIN statement
CovB	Parameter covariance matrix	COV option in the PROC HPNLIN statement
Dimensions	Number of parameters and their bounds	Default output
ECorrB	Additional estimates' correlation matrix	ECORR option in the PROC HPNLIN statement
ECovB	Additional estimates' covariance matrix	ECOV option in the PROC HPNLIN statement
FitStatistics	Statistics concerning the quality of the fit	Default output
IterHistory	Optimizer iteration information	Default output
NObs	Number of observations read and used	Default output
ParameterEstimates	Parameter estimates and associated statistics	Default output
Parameters	Initial parameter values	Default output

Table 10.4 *continued*

Table Name	Description	Required Statement / Option
PerformanceInfo	Information about high-performance computing environment	Default output
Specifications	Basic model characteristics	Default output

Examples: HPNLIN Procedure

Example 10.1: Segmented Model

Suppose you are interested in fitting a model that consists of two segments that connect in a smooth fashion. For example, the following model states that for values of x less than x_0 the mean of Y is a quadratic function in x, and for values of x greater than x_0 the mean of Y is constant:

$$\mathrm{E}[Y|x] = \begin{cases} \alpha + \beta x + \gamma x^2 & \text{if } x < x_0 \\ c & \text{if } x \geq x_0 \end{cases}$$

In this model equation α, β, and γ are the coefficients of the quadratic segment, and c is the plateau of the mean function. The HPNLIN procedure can fit such a segmented model even when the join point, x_0, is unknown.

You also want to impose conditions on the two segments of the model. First, the curve should be continuous—that is, the quadratic and the plateau section need to meet at x_0. Second, the curve should be smooth—that is, the first derivative of the two segments with respect to x need to coincide at x_0.

The continuity condition requires that

$$c = \mathrm{E}[Y|x_0] = \alpha + \beta x_0 + \gamma x_0^2$$

The smoothness condition requires that

$$\frac{\partial \mathrm{E}[Y|x_0]}{\partial x} = \beta + 2\gamma x_0 \equiv 0$$

If you solve for x_0 and substitute into the expression for c, the two conditions jointly imply that

$$\begin{aligned} x_0 &= -\beta/2\gamma \\ c &= \alpha - \beta^2/4\gamma \end{aligned}$$

Although there are five unknowns, the model contains only three independent parameters. The continuity and smoothness restrictions together completely determine two parameters given the other three.

The following DATA step creates the SAS data set for this example:

```
data a;
   input y x @@;
   datalines;
.46 1  .47  2 .57  3 .61  4 .62  5 .68  6 .69  7
.78 8  .70  9 .74 10 .77 11 .78 12 .74 13 .80 13
.80 15 .78 16
 ;
```

The following PROC HPNLIN statements fit this segmented model:

```
proc hpnlin data=a out=b;
   parms alpha=.45 beta=.05 gamma=-.0025;

   x0 = -.5*beta / gamma;

   if (x < x0) then
        yp = alpha + beta*x  + gamma*x*x;
   else
        yp = alpha + beta*x0 + gamma*x0*x0;

   model y ~ residual(yp);

   estimate 'join point' -beta/2/gamma;
   estimate 'plateau value c' alpha - beta**2/(4*gamma);
   predict 'predicted' yp pred=yp;
   predict 'response' y pred=y;
   predict 'x' x pred=x;
run;
```

The parameters of the model are α, β, and γ. They are represented in the PROC HPNLIN statements by the variables alpha, beta, and gamma, respectively. In order to model the two segments, a conditional statement assigns the appropriate expression to the mean function depending on the value of x_0. ESTIMATE statements compute the values of x_0 and c. The PREDICT statement computes predicted values for plotting and saves them to data set b.

The results from fitting this model with PROC HPNLIN are shown in Output 10.1.1 through Output 10.1.3. The iterative optimization converges after six iterations (Output 10.1.1). Output 10.1.2 shows the estimated parameters. Output 10.1.3 indicates that the join point is 12.7477 and the plateau value is 0.7775.

Output 10.1.1 Nonlinear Least Squares Iterative Phase

```
                        Quadratic Model with Plateau

                           The HPNLIN Procedure

                             Iteration History

                                     Objective                           Max
      Iteration     Evaluations       Function          Change      Gradient

              0               5   0.0035144531           .           7.184063
              1               2   0.0007352716      0.00277918       2.145337
              2               2   0.0006292751      0.00010600       0.032551
              3               2   0.0006291261      0.00000015       0.002952
              4               2   0.0006291244      0.00000000       0.000238
              5               2   0.0006291244      0.00000000       0.000023
              6               2   0.0006291244      0.00000000       2.313E-6

                  Convergence criterion (GCONV=1E-8) satisfied.
```

Output 10.1.2 Least Squares Analysis for the Quadratic Model

```
                              Analysis of Variance

                                      Sum of         Mean                Approx
    Source                    DF     Squares       Square    F Value     Pr > F

    Model                      2      0.1769       0.0884     114.22     <.0001
    Error                     13      0.0101     0.000774
    Corrected Total           15      0.1869

                               Parameter Estimates

                           Standard                               Approximate 95%
    Parameter  Estimate       Error     DF   t Value   Pr > |t|    Confidence Limits

    alpha        0.3921     0.02667     13     14.70     <.0001     0.3345     0.4497
    beta        0.06046    0.008423     13      7.18     <.0001    0.04227    0.07866
    gamma      -0.00237    0.000551     13     -4.30     0.0009   -0.00356   -0.00118
```

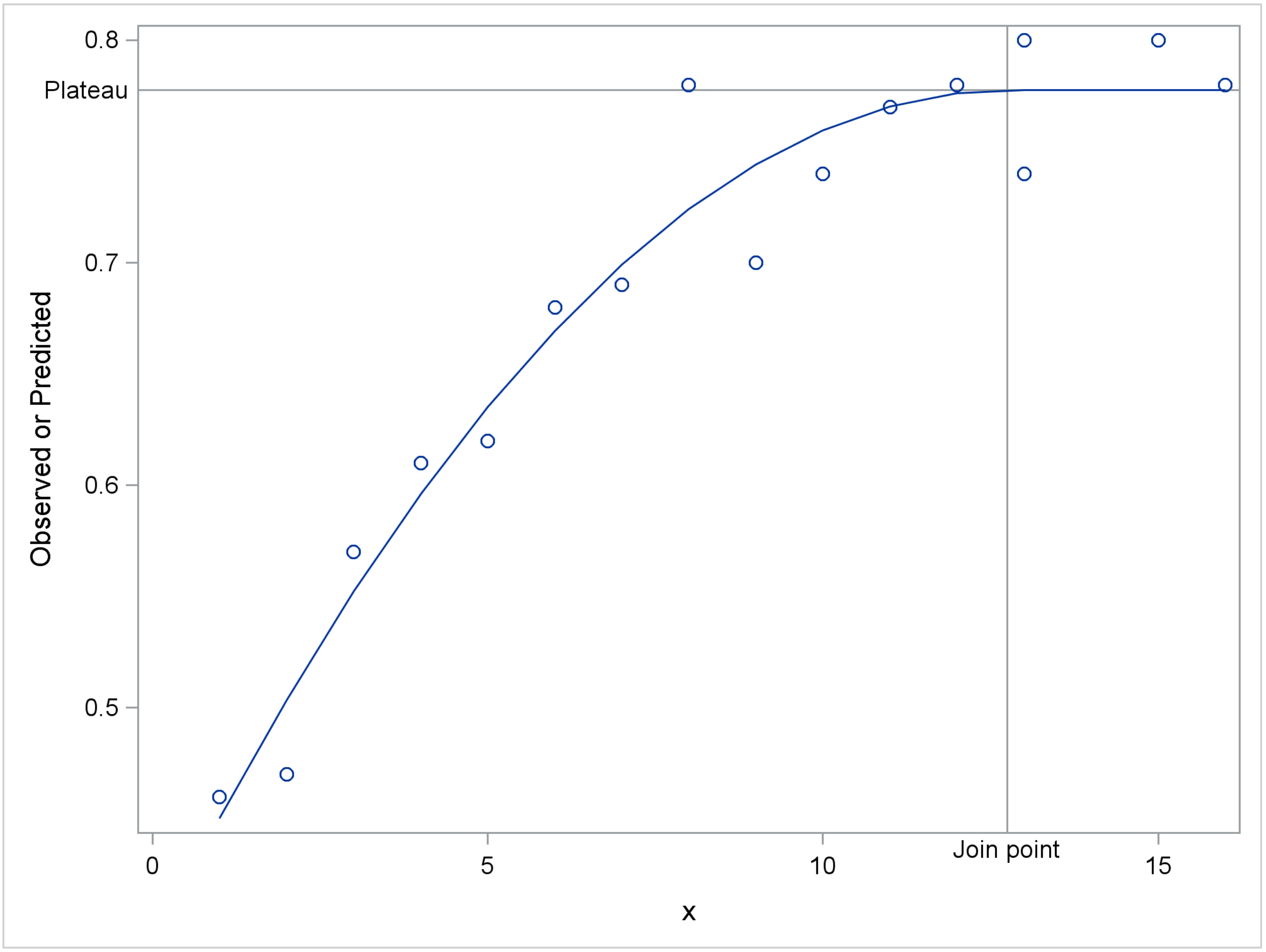
Output 10.1.3 Additional Estimates for the Quadratic Model

```
                              Additional Estimates

                               Standard
Label          Estimate           Error       DF     t Value     Pr > |t|      Alpha

join point      12.7477          1.2781       13        9.97       <.0001       0.05
plateau          0.7775         0.01232       13       63.11       <.0001       0.05

                               Additional Estimates

                     Label         Approximate Confidence Limits

                     join point       9.9864      15.5089
                     plateau          0.7509       0.8041
```

The following statements produce a graph of the observed and predicted values with reference lines for the join point and plateau estimates (Output 10.1.4):

```
proc sgplot data=b noautolegend;
   yaxis label='Observed or Predicted';
   refline 0.7775  / axis=y label="Plateau"    labelpos=min;
   refline 12.7477 / axis=x label="Join point" labelpos=min;
   scatter y=y  x=x;
   series  y=yp x=x;
run;
```

Output 10.1.4 Observed and Predicted Values for the Quadratic Model



References

Billingsley, P. (1986), *Probability and Measure*, Second Edition, New York: John Wiley & Sons.

Cox, C. (1998), "Delta Method," *Encyclopedia of Biostatistics*, 1125–1127.

Dennis, J. E., Gay, D. M., and Welsch, R. E. (1981), "An Adaptive Nonlinear Least-Squares Algorithm," *ACM Transactions on Mathematical Software*, 7, 348–368.

Dennis, J. E. and Mei, H. H. W. (1979), "Two New Unconstrained Optimization Algorithms Which Use Function and Gradient Values," *Journal of Optimization Theory Applications*, 28, 453–482.

Eskow, E. and Schnabel, R. B. (1991), "Algorithm 695: Software for a New Modified Cholesky Factorization," *ACM Transactions on Mathematical Software*, 17, 306–312.

Fletcher, R. (1987), *Practical Methods of Optimization*, Second Edition, Chichester, UK: John Wiley & Sons.

Gay, D. M. (1983), “Subroutines for Unconstrained Minimization,” *ACM Transactions on Mathematical Software*, 9, 503–524.

Moré, J. J. and Sorensen, D. C. (1983), “Computing a Trust-Region Step,” *SIAM Journal on Scientific and Statistical Computing*, 4, 553–572.

Ratkowsky, D. (1990), *Handbook of Nonlinear Regression Models*, New York: Marcel Dekker.

Chapter 11

The HPREDUCE Procedure

Contents

Overview: HPREDUCE Procedure

The HPREDUCE procedure is a high-performance procedure that performs both supervised and unsupervised variable selection on the SAS appliance. With the HPREDUCE procedure you can read data in distributed form and perform variable selection in parallel in symmetric multiprocessing (SMP) or massively parallel processing (MPP) mode. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details about how to configure the execution mode of SAS High-Performance Analytics procedures.

The HPREDUCE procedure performs unsupervised variable selection by identifying a set of variables that jointly explain the maximum amount of data variance. Unlike principal component analysis (PCA), which reduces dimensionality by generating a set of new variables (variable extraction), the HPREDUCE procedure reduces dimensionality by selecting a subset of the original variables (variable selection). Thus, this technique preserves model interpretation.

The HPREDUCE procedure performs supervised variable selection by identifying a set of variables that jointly explain the maximum amount of variance contained in the response variables. The HPREDUCE procedure supports variable selection in both the regression setting and the classification (categorization) setting.

The HPREDUCE procedure can also be used to output the sums of squares and crossproducts (SSCP) matrix, the correlation (CORR) matrix, or the covariance (COV) matrix for exploratory data analysis and direct input to statistical procedures that accept that form. This step saves time by eliminating redundant matrix aggregations.

PROC HPREDUCE Features

The HPREDUCE procedure conducts a variance analysis and reduces dimensionality by selecting the variables that contribute the most to the overall variance of the data (or the dependent variables). The following list summarizes the basic features of the HPREDUCE procedure:

- Variable selection is based on covariance analysis.
- Analysis can be performed on a massively parallel SAS high-performance appliance.
- Input data can be read in parallel when the data source is the appliance database.
- Computation of the CORR, COV, or SSCP matrix is distributed.
- Computation of the variable selection steps is distributed.
- All phases of analytic execution use of high degree of multithreading.
- Both supervised and unsupervised variable selection are supported.
- Multiple response variables are supported in variable selection for regression.

- The CLASS statement supports categorical inputs.
- The REDUCE statement supports main and interaction effects.
- The OUTCP statement supports outputting a CORR, COV, or SSCP matrix.

PROC HPREDUCE Contrasted with Other SAS Procedures

For general contrasts between High-Performance Analaytics procedures and other SAS procedures, see the session "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10. The following remarks compare the HPREDUCE procedure with the FACTOR, PRINCOMP, GLMSELECT, and DISCRIM procedures in SAS/STAT software.

When PROC HPREDUCE performs unsupervised variable selection, it conducts variance analysis and reduces dimensionality by forward selection of the variables that contribute the most to the overall data variance. The output lists the variables in order of their contribution to data variance and can be used directly for reporting or for selecting variables for model building procedures. In contrast, principal component analysis (PCA) conducts a variance analysis and then projects the data space to an orthogonal set of axes by a linear combination of the original variables. These new principal components best explain the data variance and can be used as input to model building procedures. In either case, the number of inputs to the modeling procedure has been reduced from the original set. PCA can be done through the SAS/STAT FACTOR and PRINCOMP procedures. The primary difference between PCA and PROC HPREDUCE is that PCA generates new variables, while PROC HPREDUCE reduces data dimensionality by selecting a subset of the original variables. This feature of PROC HPREDUCE is beneficial in applications where retaining the original variables is important for model exploration and interpretation.

When PROC HPREDUCE performs supervised variable selection, it conducts variance analysis and reduces dimensionality by forward selection of the variables that contribute the most to explaining the overall variance of the response variables (targets). The output lists the variables in order of their contribution to explaining response variance. The output can be used directly for reporting or for selecting variables for model building procedures. When PROC HPREDUCE is used to perform supervised variable selection, it most resembles the GLMSELECT procedure. However, PROC HPREDUCE allows multiple response variables, which is not supported by PROC GLMSELECT. When the response variable is a classification variable and its levelization is done in a special format, PROC HPREDUCE conducts variance analysis in the same way as linear discriminant analysis (LDA) does. LDA can be done through the SAS/STAT DISCRIM procedure. Like PCA, LDA generates new variables by linearly combining all original variables, while PROC HPREDUCE reduces data dimensionality by selecting a subset of the original variables.

Getting Started: HPREDUCE Procedure

The following DATA step contains 100 observations with one character variable (C), one classification variable (y), and 10 continuous variables (x1–x10). This data set is used for both of the getting-started examples in the following sections.

```
data getStarted;
   input C$ y x1-x10;
   datalines;
   D  0  10.2  6  1.6  38  15  2.4  20  0.8  8.5  3.9
   F  1  12.2  6  2.6  42  61  1.5  10  0.6  8.5  0.7
   D  1   7.7  1  2.1  38  61    1  90  0.6  7.5  5.2
   J  1  10.9  7  3.5  46  42  0.3   0  0.2    6  3.6
   E  0  17.3  6  3.8  26  47  0.9  10  0.4  1.5  4.7
   A  0  18.7  4  1.8   2  34  1.7  80    1  9.5  2.2
   B  0   7.2  1  0.3  48  61  1.1  10  0.8  3.5    4
   D  0   0.1  3  2.4   0  65  1.6  70  0.8  3.5  0.7
   H  1   2.4  4  0.7  38  22  0.2  20    0    3  4.2
   J  0  15.6  7  1.4   0  98  0.3   0    1    5  5.2
   J  0  11.1  3  2.4  42  55  2.2  60  0.6  4.5  0.7
   F  0     4  6  0.9   4  36  2.1  30  0.8    9  4.6
   A  0   6.2  2  1.8  14  79  1.1  70  0.2    0  5.1
   H  0   3.7  3  0.8  12  66  1.3  40  0.4  0.5  3.3
   A  1   9.2  3  2.3  48  51  2.3  50    0    6  5.4
   G  0    14  3    2  18  12  2.2   0    0    3  3.4
   E  1  19.5  6  3.7  26  81  0.1  30  0.6    5  4.8
   C  0    11  3  2.8  38   9  1.7  50  0.8  6.5  0.9
   I  0  15.3  7  2.2  20  98  2.7 100  0.4    7  0.8
   H  1   7.4  4  0.5  28  65  1.3  60  0.2  9.5  5.4
   F  0  11.4  2  1.4  42  12  2.4  10  0.4    1  4.5
   C  1  19.4  1  0.4  42   4  2.4  10    0  6.5  0.1
   G  0   5.9  4  2.6  12  57  0.8  50  0.4    2  5.8
   G  1  15.8  6  3.7  34   8  1.3  90  0.6  2.5  5.7
   I  0    10  3  1.9  16  80    3  90  0.4  9.5  1.9
   E  0  15.7  1  2.7  32  25  1.7  20  0.2  8.5    6
   G  0    11  5  2.9  48  53  0.1  50    1  3.5  1.2
   J  1  16.8  0  0.9  14  86  1.4  40  0.8    9    5
   D  1    11  4  3.2  48  63  2.8  90  0.6    0  2.2
   J  1   4.8  7  3.6  24   1  2.2  20    1  8.5  0.5
   J  1  10.4  5    2  42  56    1  20    0  3.5  4.2
   G  0  12.7  7  3.6   8  56  2.1  70    1  4.5  1.5
   G  0   6.8  1  3.2  30  27  0.6   0  0.8    2  5.6
   E  0   8.8  0  3.2   2  67  0.7  10  0.4    1    5
   I  1   0.2  0  2.9  10  41  2.3  60  0.2    9  0.3
   J  1   4.6  7  3.9  50  61  2.1  50  0.4    3  4.9
   J  1   2.3  2  3.2  36  98  0.1  40  0.6  4.5  4.3
   I  0  10.8  3  2.7  28  58  0.8  80  0.8    3    6
   B  0   9.3  2  3.3  44  44  0.3  50  0.8  5.5  0.4
   F  0   9.2  6  0.6   4  64  0.1   0  0.6  4.5  3.9
   D  0   7.4  0  2.9  14   0  0.2  30  0.8  7.5  4.5
   G  0  18.3  3  3.1   8  60  0.3  60  0.2    7  1.9
```

```
F  0   5.3  4  0.2  48  63  2.3  80  0.2    8  5.2
C  0   2.6  5  2.2  24   4  1.3  20    0    2  1.4
F  0  13.8  4  3.6   4   7  1.1  10  0.4  3.5  1.9
B  1  12.4  6  1.7  30  44  1.1  60  0.2    6  1.5
I  0   1.3  1  1.3   8  53  1.1  70  0.6    7  0.8
F  0  18.2  7  1.7  26  92  2.2  30    1  8.5  4.8
J  0   5.2  2  2.2  18  12  1.4  90  0.8    4  4.9
G  1   9.4  2  0.8  22  86  0.4  30  0.4    1  5.9
J  1  10.4  2  1.7  26  31  2.4  10  0.2    7  1.6
J  0    13  1  1.8  14  11  2.3  50  0.6  5.5  2.6
A  0  17.9  4  3.1  46  58  2.6  90  0.6  1.5  3.2
D  1  19.4  6    3  20  50  2.8 100  0.2    9  1.2
I  0  19.6  3  3.6  22  19  1.2   0  0.6    5  4.1
I  1     6  2  1.5  30  30  2.2  20  0.4  8.5  5.3
G  0  13.8  1  2.7   0  52  2.4  20  0.8    6    2
B  0  14.3  4  2.9  30  11  0.6  90  0.6  0.5  4.9
E  0  15.6  0  0.4  38  79  0.4  80  0.4    1  3.3
D  0    14  2    1  22  61    3  90  0.6    2  0.1
C  1   9.4  5  0.4  12  53  1.7  40    0    3  1.1
H  0  13.2  1  1.6  40  15  0.7  40  0.2    9  5.5
A  0  13.5  5  2.4  18  89  1.6  20  0.4  9.5  4.7
E  0   2.6  4  2.3  38   6  0.8  20  0.4    5  5.3
E  0  12.4  3  1.3  26   8  2.8  10  0.8    6  5.8
D  0   7.6  2  0.9  44  89  1.3  50  0.8    6  0.4
I  0  12.7  1  2.3  42   6  2.4  10  0.4    1    3
C  1  10.7  4  3.2  28  23  2.2  90  0.8  5.5  2.8
H  0  10.1  2  2.3  10  62  0.9  50  0.4  2.5  3.7
C  1  16.6  1  0.5  12  88  0.1  20  0.6  5.5  1.8
I  1   0.2  3  2.2   8  71  1.7  80  0.4  0.5  5.5
C  0  10.8  4  3.5  30  70  2.3  60  0.4  4.5  5.9
F  0   7.1  4    3  14  63  2.4  70    0    7  3.1
D  0  16.5  1  3.3  30  80  1.6  40    0  3.5  2.7
H  0  17.1  7  2.1  30  45  1.5  60  0.6  0.5  2.8
D  0   4.3  1  1.5  24  44    0  70    0    5  0.5
H  0    15  2  0.2  14  87  1.8  50    0  4.5  4.7
G  0  19.7  3  1.9  36  99  1.5  10  0.6    3  1.7
H  1   2.8  6  0.6  34  21    2  60    1    9  4.7
G  0  16.6  3  3.3  46   1  1.4  70  0.6  1.5  5.3
E  0  11.7  5  2.7  48   4  0.9  60  0.8  4.5  1.6
F  0  15.6  3  0.2   4  79  0.5   0  0.8  1.5  2.9
C  1   5.3  6  1.4   8  64    2  80  0.4    9  4.2
B  1   8.1  7  1.7  40  36  1.4  60  0.6    6  3.9
I  0  14.8  2  3.2   8  37  0.4  10    0  4.5    3
D  0   7.4  4    3  12   3  0.6  60  0.6    7  0.7
D  0   4.8  3  2.3  44  41  1.9  60  0.2    3  3.1
A  0   4.5  0  0.2   4  48  1.7  80  0.8    9  4.2
D  0   6.9  6  3.3  14  92  0.5  40  0.4  7.5    5
B  0   4.7  4  0.9  14  99  2.4  80    1  0.5  0.7
I  1   7.5  4  2.1  20  79  0.4  40  0.4  2.5  0.7
C  0   6.1  0  1.4  38  18  2.3  60  0.8  4.5  0.7
C  0  18.3  1    1  26  98  2.7  20    1  8.5  0.5
F  0  16.4  7  1.2  32  94  2.9  40  0.4  5.5  2.1
I  0   9.4  2  2.3  32  42  0.2  70  0.4  8.5  0.3
F  1  17.9  4  1.3  32  42    2  40  0.2    1  5.4
```

```
      H  0  14.9  3  1.6  36  74  2.6  60  0.2    1  2.3
      C  0  12.7  0  2.6   0  88  1.1  80  0.8  0.5  2.1
      F  0   5.4  4  1.5   2   1  1.8  70  0.4  5.5  3.6
      J  1  12.1  4  1.8  20  59  1.3  60  0.4    3  3.8
   ;
```

Unsupervised Variable Selection with the HPREDUCE Procedure

The following statements use PROC HPREDUCE for unsupervised variable selection. The statements specify that the technique used for variable selection is variance analysis. The maximum number of variables to select is 5, and the maximum percentage of the total variance to explain is 95%. The procedure stops when either of two conditions is satisfied.

```
proc hpreduce data=getstarted technique=VarianceAnalysis;
   class C;
   reduce unsupervised C x1-x10 / maxeffects=5 varexp=0.95;
   performance details;
run;
```

The output from this analysis is presented in Figure 11.1 through Figure 11.5.

Figure 11.1 shows the "Performance Information" table, which indicates that the procedure executes in client mode. That is, the procedure runs on the machine where the SAS system is running. The table also shows that two threads are used for computing.

Figure 11.1 Performance Information

```
                    Performance Information

                 Execution Mode       On client
                 Number of Threads    2
```

Figure 11.2 displays the "Model Information" and "Number of Observations" tables. The "Model Information" table shows that the HPREDUCE procedure is used for unsupervised variable selection. The CLASS variable C is parameterized in the general linear model (GLM) parameterization, which is the default. The total number of variables is 11. The technique used for variable selection is variance analysis. The maximum number of variables to select is 5, and the maximum percentage of the total variance to explain is 95%. The "Number of Observations" table shows that all 100 observations in the data set are used in the analysis.

Figure 11.2 Model Information and Number of Observations

```
                         Model Information

              Data Source                        WORK.GETSTARTED
              Model Type                         Unsupervised
              Class Parameterization             GLM
              Selection Technique                Variance Analysis
              Number of Variables                11
              Number of Variables to Select      5
              Variance to Explain                0.95


                 Number of Observations Read             100
                 Number of Observations Used             100
```

Figure 11.3 shows the “Class Level Information” table, which indicates that the CLASS variable C has 10 unique formatted levels.

Figure 11.3 Class Level Information and Response Profile

```
                       Class Level Information

                   Class    Levels    Values

                   C            10    A B C D E F G H I J
```

Figure 11.4 shows the “Selection Summary” and “Selected Effects” tables. The “Selection Summary” table shows which variable (or level for CLASS variables) is selected in each step, in addition to the total variance that is explained by the variables selected so far. The “Selected Effects” table presents all the selected variables and their corresponding variable types.

Figure 11.4 Selection Summary and Selected Effects

```
                         Selection Summary

                    Selected                   Variance
      Iteration     Variables     Levels      Explained

              1     x3              -            0.0681
              2     x7              -            0.1326
              3     x10             -            0.1933
              4     C             I              0.2529
              5     C             F               0.312
              6     C             J               0.369
              7     C             B              0.4257
              8     C             G              0.4813
              9     C             D              0.5351
             10     C             H              0.5889
             11     C             A              0.6407
             12     x5              -            0.6916


                          Selected Effects

                        Selected      Variable
             Number     Variables     Type

                  1     x3            INTERVAL
                  2     x7            INTERVAL
                  3     x10           INTERVAL
                  4     C             CLASS
                  5     x5            INTERVAL
```

Figure 11.5 shows the “Procedure Task Timing” table, which provides details about how much time is used by each processing step.

Figure 11.5 Procedure Task Timing

```
                       Procedure Task Timing

                                                 Time
  Task                                          (sec.)

  Data read and variable levelization             0.03     91.2%
  Effect Levelization                             0.00     0.00%
  Cross-product accumulation                      0.00     5.88%
  Variable selection                              0.00     2.94%
```

Supervised Variable Selection with HPREDUCE Procedure

The following statements use PROC HPREDUCE for supervised variable selection. The statements specify that y is the response variable, the technique used for variable selection is discriminant analysis, and the maximum number of variables to select is 5.

```
proc hpreduce data=getstarted technique=DiscriminantAnalysis;
    class C y;
    reduce supervised y = C x1-x10 / maxeffects=5;
    performance details;
run;
```

The output from this analysis is presented in Figure 11.6 through Figure 11.10.

Figure 11.6 shows the "Performance Information" table, which indicates that the procedure executes in client mode. That is, the procedure runs on the machine where the SAS system is running. The table also shows that two threads are used for computing.

Figure 11.6 Performance Information

```
                    Performance Information

                 Execution Mode       On client
                 Number of Threads    2
```

Figure 11.7 displays the "Model Information" and "Number of Observations" tables. The "Model Information" table shows that HPREDUCE procedure performed supervised variable selection. The CLASS variables are parameterized in the general linear model (GLM) parameterization, which is the default. The total number of variables is 12. The technique used for variable selection is discriminant analysis, and the maximum number of variables to select is 5. The "Number of Observations" table shows that all 100 observations in the data set are used in the analysis.

Figure 11.7 Model Information and Number of Observations

```
                         Model Information

         Data Source                      WORK.GETSTARTED
         Model Type                       Supervised
         Class Parameterization           GLM
         Selection Technique              Discriminant Analysis
         Number of Variables              12
         Number of Variables to Select    5


              Number of Observations Read             100
              Number of Observations Used             100
```

Figure 11.8 shows the "Class Level Information" table, which indicates that the CLASS variable C has 10 unique formatted levels and the CLASS variable y has two levels.

Figure 11.8 Class Level Information and Response Profile

```
                         Class Level Information

                 Class     Levels     Values

                 C             10     A B C D E F G H I J
                 y              2     0 1
```

Figure 11.9 shows the “Selection Summary” and “Selected Effects” tables. The “Selection Summary” table shows which variable (or level for CLASS variables) is selected in each step, in addition to the total variance that is explained by the variables selected so far. The “Selected Effects” table presents all the selected variables and their corresponding variable types.

Figure 11.9 Selection Summary and Selected Effects

```
                            Selection Summary

                         Selected                Variance
              Iteration    Variables    Levels    Explained

                      1    C            J            0.0811
                      2    x8            -           0.1323
                      3    x2            -           0.1687
                      4    C            C            0.1992
                      5    x4            -           0.2184
                      6    x9            -           0.2369


                             Selected Effects

                             Selected     Variable
                     Number    Variables    Type

                          1    C            CLASS
                          2    x8           INTERVAL
                          3    x2           INTERVAL
                          4    x4           INTERVAL
                          5    x9           INTERVAL
```

Figure 11.10 shows the “Procedure Task Timing” table, which provides details about how much time is used by each processing step.

Figure 11.10 Procedure Task Timing

```
                              Procedure Task Timing

                                                                 Time
        Task                                                    (sec.)

        Data read and variable levelization                      0.04      70.7%
        Effect Levelization                                      0.00      0.00%
        Data preparation for discriminant analysis               0.00      0.00%
        Cross-product accumulation                               0.02      27.6%
        Variable selection                                       0.00      1.72%
```

Syntax: HPREDUCE Procedure

The following statements are available in the HPREDUCE procedure:

PROC HPREDUCE < *options* > ;
  CLASS *variable* < *(options)* >. . . < *variable* < *(options)* > > < / *global-options* > ;
  REDUCE UNSUPERVISED *effects* < / *reduce-options* > ;
  REDUCE SUPERVISED *response* . . . < *response* > **=** *effects* < / *reduce-options* > ;
  PERFORMANCE *performance-options* ;

The PROC HPREDUCE statement and the REDUCE statement are required. The CLASS statement can appear multiple times. If a CLASS statement is specified, it must precede the REDUCE statement.

PROC HPREDUCE Statement

PROC HPREDUCE < *options* > ;

The PROC HPREDUCE statement invokes the procedure. Table 11.1 summarizes the important options in the PROC HPREDUCE statement by function. The options are then described fully in alphabetical order.

Table 11.1 PROC HPREDUCE Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
NAMELEN=	Limits the length of effect names
TECHNIQUE=	Selects the variable selection technique
Options Related to Output	
NOPRINT	Suppresses ODS output
NOCLPRINT	Limits or suppresses the display of CLASS levels

Table 11.1 *continued*

Option	Description
NOSUMPRINT	Suppresses generation of the selection summary table
TIMEPRINT	Prints the time used by each variable selection iteration
OUTCP=	Outputs the CORR, COV, or SSCP matrix
Pearson Correlation Statistics	
COV	Computes covariances
CORR	Computes correlations (default)
SSCP	Computes sums of squares and crossproducts
User-Defined Formats	
FMTLIBXML=	Specifies the file reference for a format stream

You can specify the following *options* in the PROC HPREDUCE statement.

CORR

selects variables based on the correlation matrix. Assuming that X and Y are two variables, the correlation between X and Y is computed by:

$$Corr(X,Y) = \frac{E\left((X - E(X))(Y - E(Y))\right)}{\sqrt{E\left(X - E(X)\right)^2 E\left(Y - E(Y)\right)^2}}$$

This is the default option for computing the Pearson correlation statistics in PROC HPREDUCE.

COV

selects variables based on the covariance matrix. Assuming that X and Y are two variables, the covariance between X and Y is computed by:

$$Cov(X,Y) = E\left((X - E(X))(Y - E(Y))\right)$$

DATA=*SAS-data-set*

names the input SAS data set to be used by PROC HPREDUCE. The default is the most recently created data set. If the procedure executes in MPP mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. When data are already distributed, the procedure reads the data alongside the distributed database. See the sections “SMP and MPP Modes” on page 10 and “Alongside-the-Database Execution” on page 15.

FMTLIBXML=*file-ref*

specifies the file reference for the XML stream that contains the user-defined format definitions. In a distributed computing environment, user-defined formats are handled differently than they are in other SAS products. See the section “Working with Formats” on page 18 in Chapter 2, “Shared Concepts and Topics,” for details about how to generate an XML stream for your formats.

NAMELEN=*number*

specifies the length to which long effect names are shortened ($20 \leq$ *number* ≤ 128). The default and minimum value is 20.

NOCLPRINT<=*number*>

suppresses the display of the "Class Level Information" table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps to reduce the size of the "Class Level Information" table if some classification variables have a large number of levels.

NOPRINT

suppresses the generation of ODS output.

NOSUMPRINT

suppresses the generation of the "Selection Summary" table.

OUTCP=*SAS-data-set*</LIST<(EPS = *number*)>>

creates both a data set that contains a symmetric matrix that depicts the relationships among variables and also a set of statistics about the input data set and variables. Depending on the Pearson correlation statistics option specified in the PROC HPREDUCE statement, the symmetric matrix can be a correlation (CORR) matrix, a covariance (COV) matrix, or a sums of squares and crossproducts (SSCP) matrix.

When the LIST option is specified, the symmetric matrix is output in the list-of-list (LIL) format. In this format, the matrix is represented as a set of tuples $(i,\ j,\ x)$, where x is an entry in the matrix and i and j denote its row and column indices, respectively. LIL format can be used when the output contains too many columns to fit in a data set. For example, in most database systems the maximum number of columns in a table is usually limited to several thousand. If an output matrix contains more columns than the limit, you must use the LIST option to avoid errors that would arise from writing too many columns to the table. When LIL format is used, all 0 entries in the matrix are ignored in the output.

When EPS= *number* is specified in the LIST option, matrix entries that have an absolute value smaller than *number* are ignored in the output. This feature helps omit unreliable estimations and generate a compact representation for the matrix. When the EPS= option is not specified, only the 0 entries in the matrix are ignored in the output.

SSCP

selects variables based on the sums of squares and crossproducts matrix. Assuming that X and Y are two variables and that $\mathbf{x}$ and $\mathbf{y}$ are their corresponding variable vectors, the SSCP between X and Y is computed by:

$$SSCP(X,Y) = \mathbf{x}^\top \mathbf{y}$$

TECHNIQUE=*keyword*

TECH=*keyword*

specifies the variable selection technique. You can specify the following *keyword*s:

VARIANCEANALYSIS | VAR	performs variance analysis for variable selection.
DISCRIMINANTANALYSIS | DSC	performs discriminant analysis for variable selection.

The default value is TECHNIQUE=VAR.

You can use variance analysis for both supervised and unsupervised variable selection. You can use discriminant analysis only for supervised variable selection with one classification variable as the response. For more information, see the section "Variable Selection for Classification" on page 303.

TIMEPRINT

prints the time (in seconds) used by each variable selection iteration in the "Selection Summary" table. If this option does not appear in the PROC HPREDUCE statement, the time information is not printed.

CLASS Statement

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

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. The CLASS statement must precede the REDUCE statement.

The CLASS statement for High-Performance Analytics procedures is documented in the section "CLASS Statement" on page 20 of Chapter 2, "Shared Concepts and Topics."

The HPREDUCE procedure does not support the SPLIT option in the CLASS statement. For CLASS variable parameterization, the HPREDUCE procedure only support the GLM method.

REDUCE Statement

REDUCE UNSUPERVISED *effects < / reduce-options >* **;**

REDUCE SUPERVISED *response . . . < response > = effects < / reduce-options >* **;**

PROC HPREDUCE can be used for both supervised and unsupervised variable selection. In unsupervised case, the REDUCE statement specifies the effects to be considered in the variable selection process. An effect can be an original variable in the input data set or a variable constructed from the original variables. In the supervised case, you need to specify both the effects and the response variables. A response variable can be an original variable in the input data set or a variable constructed from the original variables. For the regression case, you can specify more than one response variable.

Table 11.2 summarizes the *reduce-options*, which control the number of variables to be selected.

Table 11.2 *reduce-options*

Option	Description
MAXSTEPS=	Specifies the maximum number of steps to take for variable selection; the number must be greater than or equal to 1.
MAXEFFECTS=	Specifies the number of effects to select; the number must be greater than or equal to 1.

Table 11.2 *continued*

Option	Description
VARIANCEEXPLAINED | VAREXP=	Specifies the fraction of the total variance to be explained; the value must be between 0 and 1.
MINVARIANCEINCREMENT | VARINC=	Specifies the minimum increment of explained variance (in a fraction of the total variance); the value must be between 0 and 1.

The *reduce-options* determine the number of variables to be selected. You can specify the following *reduce-options*. When any of the options is satisfied, the HPREDUCE procedure stops.

MAXSTEPS=*n*

stops PROC HPREDUCE after it runs *n* steps.

MAXEFFECTS=*n*

stops PROC HPREDUCE after *n* effects have been selected. Since individual levels of one classification variable can be selected in different steps of the variable selection process, PROC HPREDUCE might take more than *n* steps to select *n* effects.

VARIANCEEXPLAINED=*fraction*

VAREXP=*fraction*

stops PROC HPREDUCE when the *fraction* of the total variance can be explained by the selected variables.

MINVARIANCEINCREMENT=*fraction*

VARINC=*fraction*

stops PROC HPREDUCE when the minimum increment of the explained variance is less than *fraction* of the total variance.

PERFORMANCE Statement

PERFORMANCE < *performance-options*> **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of the HPREDUCE procedure.

With the PERFORMANCE statement, you can also control whether the HPREDUCE procedure executes in SMP or MPP mode.

The PERFORMANCE statement for High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

Details: HPREDUCE Procedure

The performance of a learning model usually decreases in terms of accuracy and efficiency when the dimensionality of the input data is high. The problem is known as the "curse of dimensionality." Variable selection techniques can reduce the dimensionality of a data set by removing irrelevant and redundant variables (Liu and Motoda 1998).

The HPREDUCE procedure performs both supervised and unsupervised variable selection. It selects variables by identifying a set of variables that can jointly explain the maximum amount of data variance.

Unsupervised Variable Selection

When no response variable is specified, PROC HPREDUCE conducts unsupervised variable selection. Assume that k variables need to be selected. Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be a data set that contains n samples and m variables; let $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$, where $\mathbf{X}_1 \in \mathbb{R}^{n \times k}$ is the data set that contains the k selected variables and $\mathbf{X}_2 \in \mathbb{R}^{n \times (m-k)}$ contains the remaining $m - k$ variables. PROC HPREDUCE selects the variables by minimizing the equation:

$$\min Trace \left(\mathbf{X}_2^\top \left(\mathbf{I} - \mathbf{X}_1 \left(\mathbf{X}_1^\top \mathbf{X}_1 \right)^{-1} \mathbf{X}_1^\top \right) \mathbf{X}_2 \right)$$

$\left(\mathbf{I} - \mathbf{X}_1 \left(\mathbf{X}_1^\top \mathbf{X}_1 \right)^{-1} \mathbf{X}_1^\top \right)^{\frac{1}{2}} \mathbf{X}_2$ projects $\mathbf{X}_2$ to the null space of $\mathbf{X}_1$. Therefore, the above equation measures the data variance that resides in the null space of $\mathbf{X}_1$, which is the data variance that cannot be explained by the variables in $\mathbf{X}_1$. Minimizing this equation leads to the selection of the variables that jointly explain the maximum amount of the variance in the original data.

Let $\mathbf{C}_{11} = \mathbf{X}_1^\top \mathbf{X}_1$, $\mathbf{C}_{12} = \mathbf{X}_1^\top \mathbf{X}_2$, and $\mathbf{C}_{21} = \mathbf{X}_2^\top \mathbf{X}_1$. The following equations hold:

$$\mathbf{C} = \mathbf{X}^\top \mathbf{X} = \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix}$$

$$\mathbf{X}_2^\top \left(\mathbf{I} - \mathbf{X}_1 \left(\mathbf{X}_1^\top \mathbf{X}_1 \right)^{-1} \mathbf{X}_1^\top \right) \mathbf{X}_2 = \mathbf{C}_{22} - \mathbf{C}_{21} \mathbf{C}_{11}^{-1} \mathbf{C}_{21}$$

When all the variables are centralized to have a zero mean, $\mathbf{C}$ is the covariance matrix. This corresponds to setting the COV option in the PROC HPREDUCE statement, which specifies that the covariance matrix be used for variable selection. Similarly, if variables need to be both centralized and normalized to have unit length, you should specify the CORR option in the PROC HPREDUCE statement, which leads to the use of the correlation matrix for variable selection. If neither centralization nor normalization should be applied, you need to specify the SSCP option in the PROC HPREDUCE statement.

Principal component analysis (PCA) (Jolliffe 2002) also reduces dimensionality by preserving data variance. The key difference between PCA and PROC HPREDUCE is that PCA generates a small set of new variables (variable extraction) by linearly combining the original variables, while PROC HPREDUCE selects a small set of the original variables (variable selection). The variables returned by PROC HPREDUCE are the original variables. This feature is very important in applications where retaining the original variables is important for model exploration or interpretation (for example, genetic analysis, and text mining).

Supervised Variable Selection

When response variables are specified in a REDUCE statement, PROC HPREDUCE conducts supervised variable selection. The procedure supports variable selection both in a regression context and in a classification (categorization) context.

Variable Selection for Regression

In a regression setting, all response variables should be numerical. When a classification variable is in the response, this variable needs to be levelized to multiple dummy variables, with each dummy variable corresponding to a level of the classification variable. You can achieve this levelization by adding this variable to the variable list of the CLASS statement.

Let $\mathbf{Y} \in \mathbb{R}^{n \times t}$ be the response data that contain t response variables. Assume that k variables need to be selected. Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be a data set that contains n samples and m variables; let $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$, where $\mathbf{X}_1 \in \mathbb{R}^{n \times k}$ is the data set that contains the k selected variables and $\mathbf{X}_2 \in \mathbb{R}^{n \times (m-k)}$ contains the remaining $m - k$ variables. PROC HPREDUCE selects the variables by minimizing the following equation:

$$\min Trace\left(\mathbf{Y}^\top \left(\mathbf{I} - \mathbf{X}_1 \left(\mathbf{X}_1^\top \mathbf{X}_1\right)^{-1} \mathbf{X}_1^\top\right) \mathbf{Y}\right)$$

$\left(\mathbf{I} - \mathbf{X}_1 \left(\mathbf{X}_1^\top \mathbf{X}_1\right)^{-1} \mathbf{X}_1^\top\right)^{\frac{1}{2}} \mathbf{Y}$ projects $\mathbf{Y}$ to the null space of $\mathbf{X}_1$. Therefore, the equation measures the response variance that resides in the null space of $\mathbf{X}_1$, which is the variance of the response variables that cannot be explained by the variables in $\mathbf{X}_1$. Minimizing the equation leads to the selection of the variables that jointly explain the maximum amount of the variance of the response variables.

Variable Selection for Classification

In a classification setting, one classification variable is specified as the response, with each of its levels corresponding to a category of the classification problem. Let the classification variable be $\mathbf{y}$ with c levels $\{1, \ldots, c\}$. Then $\mathbf{y}$ can be levelized in a special way to generate a response data $\mathbf{Y} \in \mathbb{R}^{n \times c}$ as:

$$\mathbf{Y}_{i,j} = \begin{cases} \frac{1}{\sqrt{n}}\left(\sqrt{\frac{n}{n_j}} - \sqrt{\frac{n_j}{n}}\right), & \mathbf{y}_i = j \\ -\frac{1}{\sqrt{n}}\sqrt{\frac{n_j}{n}} & , \quad \mathbf{y}_i \neq j \end{cases}$$

By using this $\mathbf{Y}$ in the variance analysis, PROC HPREDUCE selects variables by using the discriminant criterion specified in linear discriminant analysis (LDA) (Fisher 1936; Cooley and Lohnes 1971). LDA also reduces dimensionality. The key difference between LDA and PROC HPREDUCE is that LDA generates a small set of new variables (variable extraction) by linearly combining the original variables, while PROC HPREDUCE selects a small set of the original variables (variable selection).

Computational Method

Given m variables, finding the k variables that minimize the proposed equations is a combinatorial problem, which is NP-hard (nondeterministic polynomial-time hard). To select k variables, PROC HPREDUCE applies k steps of a greedy search to generate a suboptimal solution for the problem.

Assume that q features have been selected, that $\mathbf{X}_1$ contains the q selected variables, and that $\mathbf{X}_2$ contains the remaining variables. PROC HPREDUCE selects the $q + 1$ variable, F, by minimizing the equation

$$arg \min_{F} Trace\left(\hat{\mathbf{X}}_2^{\top}\left(\mathbf{I}-\hat{\mathbf{X}}_1\left(\hat{\mathbf{X}}_1^{\top}\hat{\mathbf{X}}_1\right)^{-1}\hat{\mathbf{X}}_1^{\top}\right)\hat{\mathbf{X}}_2\right)$$

where $\hat{\mathbf{X}}_1$ is the data set that contains the feature F and the q selected variables, and $\hat{\mathbf{X}}_2$ is the data set that contains the remaining variables. Minimizing the preceding problem is equivalent to maximizing the following problem:

$$\frac{\left\|\mathbf{X}_2^{\top}\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)\mathbf{f}\right\|_2^2}{\left\|\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)^{\frac{1}{2}}\mathbf{f}\right\|_2^2}$$

In the preceding equation, $\left\|\mathbf{X}_2^{\top}\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)\mathbf{f}\right\|_2^2$ is the summation of the squares of the covariance between the variable $\mathbf{f}$ and all the unselected variables in the null space of $\mathbf{X}_1$. And $\left\|\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)^{\frac{1}{2}}\mathbf{f}\right\|_2^2$ is the square of the variance of $\mathbf{f}$ in the null space of $\mathbf{X}_1$, which is used as a normalization factor.

This problem can be solved efficiently. Assuming that $m \gg k$, the time complexity for solving it is

$$O\left(m^2\left(n+k^2\right)\right)$$

where m is the number of variables, n is the number of samples, and k is the number of selected variables. In the equation, m^2n corresponds to the time for computing the covariance (or correlation, or SSCP) matrix. And m^2k^2 corresponds to the time for selecting k variables out of m.

Similar analysis also applies to supervised variable selection with PROC HPREDUCE. In this case, the following problem is maximized for variable selection:

$$\frac{\left\|\mathbf{Y}^{\top}\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)\mathbf{f}\right\|_2^2}{\left\|\left(\mathbf{I}-\mathbf{X}_1\left(\mathbf{X}_1^{\top}\mathbf{X}_1\right)^{-1}\mathbf{X}_1^{\top}\right)^{\frac{1}{2}}\mathbf{f}\right\|_2^2}$$

Here, $\mathbf{Y}$ is the response data. Let c be the number of columns in $\mathbf{Y}$. The time complexity for selecting k variables by solving the preceding problem is

$$O\left(k^2\left(c+k\right)m+m^2n\right)$$

Note that for most data of very high dimensionality, $c + k \ll m$.

PROC HPREDUCE is fully threaded and distributed. When there are p machines used for computing, the time complexity for unsupervised variable selection is

$$CPU\left(\frac{m^2\left(n+k^2\right)}{p} + m^2 \log p\right) + NET\left(m^2 \log p\right)$$

And the time complexity for supervised variable selection is

$$CPU\left(\frac{k^2\left(c+k\right)m + m^2 n}{p} + m^2 \log p\right) + NET\left(m^2 \log p\right)$$

where CPU corresponds to the time used for computing and NET corresponds to the time used for communication among computers.

Displayed Output

The following sections describe the output that PROC HPREDUCE produces by default. The output is organized into various tables, which are discussed in the order of appearance.

Performance Information

The “Performance Information” table is produced by default. It displays information about the grid host for distributed execution and information about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also displayed, depending on the environment.

Model Information

The “Model Information” table displays basic information about the model, such as the data source, the selection technique, the number of selected variables, and the execution mode that the HPREDUCE procedure determines based on your input and options. If you want to know whether the procedure executed on the client machine or in distributed form or whether data were sent from the client or processed alongside the database, check the execution mode entry of this table.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data set and the number of observations used in the analysis.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the CLASS statement. You should check this information to make sure that the data are correct. You can adjust the order of the CLASS variable levels with the ORDER= option in the CLASS statement.

If the classification variables are in reference parameterization, the “Class Level Information” table also displays the reference value for each variable.

Selection Summary

The “Selection Summary” table displays for each iteration the name of the selected effect, the name of the selected level, and the total variance explained after the iteration. If you specify the TIMEPRINT option in the PROC HPREDUCE statement, information about the time used by each iteration is added to the “Selection Summary” table.

Selection Effects

The “Selected Effects” table summarizes which effects were selected in the selection process. It also provides information about the variable type of each selected effect.

Procedure Task Timing

If you specify the DETAILS option in the PERFORMANCE statement, the procedure produces a “Procedure Task Timing” table, which displays the elapsed time (absolute and relative) for the main tasks of the procedure.

ODS Table Names

Each table created by the HPREDUCE procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 11.3.

Table 11.3 ODS Tables Produced by PROC HPREDUCE

Table Name	Description	Required Statement / Option
ClassLevels	Level information from the CLASS statement	CLASS
ModelInfo	Information about modeling environment	Default output
NObs	Number of observations read and used; number of events and trials, if applicable	Default output
SelectedEffects	Summary of selected variables	Default output
SelectionSummary	Selection summary	Default output
Timing	Absolute and relative times for tasks performed by the procedure	DETAILS option in the PERFORMANCE statement

Examples: HPREDUCE Procedure

Example 11.1: Running in Soloist Mode

This example first generates a data set, which has 2,000 observations and contains both interval variables (x1–x10) and CLASS variables (b1–b3, c1–c10). Then PROC HPREDUCE is run to select variables. When a host for distributed computing is not specified or the NODES option in the PERFORMANCE statement is not specified, PROC HPREDUCE runs automatically in soloist mode.

```
data one;
  array x{10};
  array c{10};
  do i=1 to 2000;
    do j=1 to 10;
       x{j}=ranuni(1);
       c{j}=int(ranuni(1)*4);
    end;
    if c{1} eq 0 then b1 = 'aa';
    if c{1} eq 1 then b1 = 'bb';
    if c{1} eq 2 then b1 = 'cc';
    if c{1} eq 3 then b1 = 'dd';
    if c{1} eq 4 then b1 = 'ee';

    if c{2} eq 0 then b2 = 'ff';
    if c{2} eq 1 then b2 = 'gg';
    if c{2} eq 2 then b2 = 'hh';
    if c{2} eq 3 then b2 = 'ii';
    if c{2} eq 4 then b2 = 'jj';

    if c{3} eq 0 then b3 = 'kk';
    if c{3} eq 1 then b3 = 'll';
    if c{3} eq 2 then b3 = 'mm';
    if c{3} eq 3 then b3 = 'nn';
    if c{3} eq 4 then b3 = 'oo';
    output;
  end;
run;

proc hpreduce data=one;
    class b1-b3 c1-c3;
    reduce unsupervised b1 b1*b2 b3 c1-c3 x1-x10/maxsteps=5;
    performance details;
run;
```

Output 11.1.1 shows the results for PROC HPREDUCE running in soloist mode. Notice that the “Performance Information” table shows that the “Execution mode” is “On client.”

Output 11.1.1 PROC HPREDUCE Running in Soloist Mode

```
                      Performance Information

                   Execution Mode       On client
                   Number of Threads    2

                         Model Information

            Data Source                 WORK.ONE
            Model Type                  Unsupervised
            Class Parameterization      GLM
            Selection Technique         Variance Analysis
            Number of Variables         16
            Maximal Number of Steps     5

            Number of Observations Read            2000
            Number of Observations Used            2000

                      Class Level Information

                  Class    Levels    Values
                  b1            4    aa bb cc dd
                  b2            4    ff gg hh ii
                  b3            4    kk ll mm nn
                  c1            4    0 1 2 3
                  c2            4    0 1 2 3
                  c3            4    0 1 2 3

                         Selection Summary

                     Selected                   Variance
           Iteration    Variables    Levels    Explained
                   1    b1           cc           0.0821
                   2    b1           bb           0.1637
                   3    b1           aa           0.2443
                   4    b3           kk           0.3036
                   5    b3           mm           0.3619


                          Selected Effects

                          Selected     Variable
                  Number    Variables    Type
                       1    b1           CLASS
                       2    b3           CLASS


                       Procedure Task Timing
                                                    Time
    Task                                           (sec.)
    Data read and variable levelization              0.03     56.5%
    Effect Levelization                              0.00     8.70%
    Cross-product accumulation                       0.02     34.8%
    Variable selection                               0.00     0.00%
```

Example 11.2: Running in Distributed Mode

When a host for distributed computing is specified and the NODES option in the PERFORMANCE statement is specified, PROC HPREDUCE uses the specified host for computing and runs in distributed mode.

```
option set=GRIDHOST="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";

data one;
    array x{10};
    array c{10};
    do i=1 to 2000;
        do j=1 to 10;
            x{j}=ranuni(1);
            c{j}=int(ranuni(1)*4);
        end;
        y=int(ranuni(1)*2);
        output;
    end;
run;

proc hpreduce data=one tech=var;
    class c1 c2 c3;
    reduce supervised y = c1-c3 x1-x10/maxsteps=5;
    performance nodes=2;
run;
```

To run the preceding example successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to the macro variables in the example with the appropriate values.

Output 11.2.1 shows the results for PROC HPREDUCE running in distributed mode. Notice that the “Performance Information” table shows that the “Execution mode” is “Distributed.”

Output 11.2.1 PROC HPREDUCE Running in Distributed Mode

```
                    Performance Information

         Host Node                      << your grid host >>
         Install Location               << your grid install location >>
         Execution Mode                 Distributed
         Number of Compute Nodes        2
         Number of Threads per Node     8


                       Model Information

             Data Source                 WORK.ONE
             Model Type                  Supervised
             Class Parameterization      GLM
             Selection Technique         Variance Analysis
             Number of Variables         14
             Maximal Number of Steps     5


             Number of Observations Read            2000
             Number of Observations Used            2000


                    Class Level Information

                       Class    Levels    Values

                       c1            4    0 1 2 3
                       c2            4    0 1 2 3
                       c3            4    0 1 2 3


                       Selection Summary

                          Selected                Variance
                Iteration    Variables    Levels    Explained

                        1    c2           0            0.0017
                        2    c3           0            0.0026
                        3    c3           3            0.0034
                        4    x2             -          0.0042
                        5    x8             -          0.0045


                        Selected Effects

                              Selected     Variable
                       Number    Variables    Type

                            1    c2           CLASS
                            2    c3           CLASS
                            3    x2           INTERVAL
                            4    x8           INTERVAL
```

Example 11.3: Output a Correlation Matrix to a SAS Data File

This example shows how to output a correlation matrix to a SAS data file. The OUTCP option creates an output data set named corr.

```
data one;
    array x{2};
    array c{2};
    do i=1 to 2000;
        do j=1 to 2;
            x{j}=ranuni(1);
            c{j}=int(ranuni(1)*2);
        end;
        output;
    end;
run;

title  "Output the Correlation Matrix";

proc hpreduce data=one corr outcp=corr;
    class a;
    reduce unsupervised a x1-x2 /maxsteps=4;
run;

proc print data=corr;
run;
```

Output 11.3.1 shows the content of the data file generated by PROC HPREDUCE.

Output 11.3.1 Output the Correlation Matrix

```
Obs _ID_  _TYPE_     _VAR_ _LEV_ _vID_        v1        v2        v3        v4

 1    1   MEAN/FREQ                        979.00   1021.00      0.49      0.50
 2    2   N                               2000.00   2000.00   2000.00   2000.00
 3    3   CORR        a      0     v1        1.00     -1.00      0.03      0.00
 4    4   CORR        a      1     v2       -1.00      1.00     -0.03     -0.00
 5    5   CORR        x1           v3        0.03     -0.03      1.00      0.00
 6    6   CORR        x2           v4        0.00     -0.00      0.00      1.00
```

The values in the column _VAR_ are the name of the variables. The _LEV_ column shows the name of a CLASS variable's levels, but is empty for interval variables. Assuming that you have n effects (the total number of interval variables and the levels of CLASS variables), the _vID_ column contains n markers, v1 to vn, where vi denotes the ith effect. The column _TYPE_ defines the role of each row. When the _TYPE_ column shows MEAN/FREQ, the corresponding row contains either the mean for an interval variable or the frequency for a level of a CLASS variable. When the _TYPE_ column shows N, the corresponding row contains the number of samples. And when the _TYPE_ column shows CORR, COV, or SSCP, the corresponding row contains a row of the CORR, COV, or SSCP matrix. In this example, the CORR matrix is 4×4, and it resides in the table in row 3 through row 6 and column 7 through column 10.

Example 11.4: Output the Correlation Matrix in LIL Format

By using the LIST option in the OUTCP option, you can output a correlation matrix in LIL format.

```
data one;
    array x{2};
    do i=1 to 2000;
        a=int(ranuni(1)*2);
        do j=1 to 2;
            x{j}=ranuni(1);
        end;
        output;
    end;
run;

title  "Output the Correlation Matrix in LIL format";

proc hpreduce data=one corr outcp=corr_lil/list(eps=0.01);
    class a;
    reduce unsupervised a x1-x2 /maxsteps=4;
run;

proc print data=corr_lil;
run;
```

Output 11.4.1 shows the correlation matrix in LIL format.

Output 11.4.1 Output the Correlation Matrix in LIL Format

```
Obs    _TYPE_    _ID_    _NAME1_    _NAME2_       _VAL_

  1      S         1     samples                2000.00
  2      S         2     nVar                      3.00
  3      S         3     nEff                      4.00
  4      F         1     a             0         979.00
  5      F         2     a             1        1021.00
  6      M         3     x1                        0.49
  7      M         4     x2                        0.50
  8      R         1     1             1           1.00
  9      R         2     2             1          -1.00
 10      R         3     2             2           1.00
 11      R         4     3             1           0.03
 12      R         5     3             2          -0.03
 13      R         6     3             3           1.00
 14      R        10     4             4           1.00
```

The column _TYPE_ defines the type of each row:

- When the _TYPE_ column shows S, the corresponding row contains the statistics of the data set. More specifically, when the _TYPE_ column shows S and the _NAME1_ column shows samples, the

VAL column in the corresponding row contains the number of samples in the data set. Similarly, when the _TYPE_ column shows S and the _NAME1_ column shows nVar, the _VAL_ column contains the number of variables. And when the _TYPE_ column show S and the _NAME1_ column shows nEff, the _VAL_ column in the corresponding row contains the number of effects.

- When the _TYPE_ column shows F, the row contains the frequency of a level of a CLASS variable. In this case, the _NAME1_ column contains the name of the CLASS variable and the _NAME2_ column contains the name of the corresponding level.
- When the _TYPE_ column shows M, the row contains the mean of an interval variable. In this case, the _NAME1_ column contains the name of the variable and the _NAME2_ column is empty.
- When the _TYPE_ column shows R, the row contains an entry in the correlation matrix. In this case, the _NAME1_ column contains the row ID, the _NAME2_ column contains the column ID, and the _VAL_ column contains the value.
- When the _TYPE_ column shows V or P, the corresponding row contains an entry of a COV matrix or an SSCP matrix, respectively.

Only entries in the lower triangle of the correlation matrix are written to the file, because the correlation matrix is symmetric. Also any entry of the matrix that has a value smaller than 0.01 is ignored in the output (EPS = 0.01), which saves storage space.

Example 11.5: Output an ODS Table as A Local Data File

The ODS output of PROC HPREDUCE can be stored in a local data file. The following example shows how the “Iteration History” table can be stored as a local file named IterHist by using the ODS output statement:

```
data one;
  array x{10};
  array c{10};
  do i=1 to 2000;
    do j=1 to 10;
       x{j}=ranuni(1);
       c{j}=int(ranuni(1)*4);
    end;

    if c{1} eq 0 Then b1 = 'aa';
    if c{1} eq 1 Then b1 = 'bb';
    if c{1} eq 2 then b1 = 'cc';
    if c{1} eq 3 then b1 = 'dd';
    if c{1} eq 4 then b1 = 'ee';

    if c{2} eq 0 Then b2 = 'ff';
    if c{2} eq 1 Then b2 = 'gg';
    if c{2} eq 2 then b2 = 'hh';
    if c{2} eq 3 then b2 = 'ii';
    if c{2} eq 4 then b2 = 'jj';
```

```
        if c{3} eq 0 Then b3 = 'kk';
        if c{3} eq 1 Then b3 = 'll';
        if c{3} eq 2 then b3 = 'mm';
        if c{3} eq 3 then b3 = 'nn';
        if c{3} eq 4 then b3 = 'oo';
        output;
      end;
    run;

    proc hpreduce data=one;
        ods output SelectionSummary=Summary;
        class b1-b3 c1-c3;
        reduce  unsupervised b1 b1*b2 b3 c1-c3 x1-x10 /maxsteps =5;
        performance details;
    run;
```

References

Cooley, W. W. and Lohnes, P. R. (1971), *Multivariate Data Analysis*, New York: John Wiley & Sons.

Fisher, R. A. (1936), “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Eugenics*, 7, 179–188.

Jolliffe, I. T. (2002), *Principal Component Analysis*, New York: Springer-Verlag.

Liu, H. and Motoda, H. (1998), *Feature Selection for Knowledge Discovery and Data Mining*, Norwell, MA: Kluwer Academic.

Chapter 12

The HPREG Procedure

Contents

Overview: HPREG Procedure

The HPREG procedure is a high-performance procedure that fits and performs model selection for ordinary linear least squares models. The models supported are standard independently and identically distributed general linear models, which can contain main effects that consist of both continuous and classification variables and interaction effects of these variables. The procedure offers extensive capabilities for customizing the model selection with a wide variety of selection and stopping criteria, from traditional and computationally efficient significance-level-based criteria to more computationally intensive validation-based criteria. PROC HPREG also provides a variety of regression diagnostics that are conditional on the selected model.

With the HPREG procedure you can read and write data in distributed form and perform analyses in parallel in symmetric multiprocessing (SMP) or massively parallel processing (MPP) mode. See the section “SMP and MPP Modes” on page 10 in Chapter 2, “Shared Concepts and Topics,” for details.

PROC HPREG Features

The main features of the HPREG procedure are as follows:

- **High-performance features**
  - can perform analysis on a massively parallel SAS high-performance appliance
  - reads input data in parallel and writes output data in parallel when the data source is the appliance database
  - is highly multithreaded during all phases of analytic execution

- **Model specification**
  - supports GLM and reference parameterization for classification effects
  - supports any degree of interaction (crossed effects) and nested effects
  - supports hierarchy among effects
  - supports partitioning of data into training, validation, and testing roles
  - supports a FREQ statement for grouped analysis
  - supports a WEIGHT statement for weighted analysis
- **Selection control**
  - provides multiple effect-selection methods
  - enables selection from a very large number of effects (tens of thousands)
  - offers selection of individual levels of classification effects
  - provides effect selection based on a variety of selection criteria
  - provides stopping rules based on a variety of model evaluation criteria
  - supports stopping and selection rules based on external validation and leave-one-out cross validation
- **Display and output**
  - produces output data sets that contain predicted values, residuals, studentized residuals, confidence limits, and influence statistics

The HPREG procedure supports the following effect selection methods. For a more detailed description of these methods, see the section "Methods" on page 44 in Chapter 2, "Shared Concepts and Topics."

FORWARD	The forward selection method starts with no effects in the model and adds effects.
BACKWARD	The backward elimination method starts with all effects in the model and deletes effects.
STEPWISE	The stepwise regression method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.
FORWARDSWAP	The forward swap selection method is a modification of forward selection where before any addition step, all pairwise swaps of effects in and out of the current model that improve the selection criterion are made. When the selection criterion is R square, this method coincides with the MAXR method in the REG procedure in SAS/STAT software.
LAR	The least angle regression method, like forward selection, starts with no effects in the model and adds effects. The parameter estimates at any step are "shrunk" when compared to the corresponding least squares estimates.
LASSO	The lasso method adds and deletes parameters based on a version of ordinary least squares in which the sum of the absolute regression coefficients is constrained. PROC HPREG also supports adaptive lasso selection where weights are applied to each of the parameters in forming the lasso constraint.

Hybrid versions of LAR and LASSO methods are also supported. They use LAR or LASSO to select the model, but then estimate the regression coefficients by ordinary weighted least squares.

PROC HPREG Contrasted with Other SAS Procedures

For general contrasts between SAS High-Performance Analytics procedures and other SAS procedures, see the section "SAS High-Performance Analytics Procedures Contrasted with Other SAS Procedures" on page 10 in Chapter 2, "Shared Concepts and Topics." The following remarks contrast the HPREG procedure with the GLMSELECT, GLM, and REG procedures in SAS/STAT software.

A major functional difference between the HPREG and REG procedures is that the HPREG procedure enables you to specify general linear models that include classification variables. In this respect it is similar to the GLM and GLMSELECT procedures. In terms of the supported model selection methods, the HPREG procedure most resembles the GLMSELECT procedure. Like the GLMSELECT procedure but different from the REG procedure, the HPREG procedure supports the LAR and LASSO methods, the ability to use external validation data and cross validation as selection criteria, and extensive options to customize the selection process. The HPREG procedure does not support the MAXR and MINR methods that are available in the REG procedure. Nor does the HPREG procedure include any support for the all-subset-based methods that you can find in the REG procedure.

The CLASS statement in the HPREG procedure permits two parameterizations: the GLM-type parameterization and a reference parameterization. In contrast to the GLMSELECT, GENMOD, LOGISTIC, and other procedures that permit multiple parameterizations, the HPREG procedure does not mix parameterizations across the variables in the CLASS statement. In other words, all classification variables are in the same parameterization, and this parameterization is either the GLM or reference parameterization.

Like the REG procedure but different from the GLMSELECT procedure, the HPREG procedure does not perform model selection by default. If you request model selection by using the SELECTION statement then the default selection method is stepwise selection based on the SBC criterion. This default matches the default method used in PROC GLMSELECT.

As with the REG procedure but not supported with the GLMSELECT procedure, you can request observation-wise residual and influence diagnostics in the OUTPUT statement and variance inflation and tolerance statistics for the parameter estimates. If the fitted model has been obtained by performing model selection, then these statistics are conditional on the selected model and do not take the variability introduced by the selection process into account.

Getting Started: HPREG Procedure

The following example is closely modeled on the example in the section "Getting Started: GLMSELECT Procedure" in the *SAS/STAT User's Guide*. The data set contains salary and performance information for Major League Baseball players (excluding pitchers) who played at least one game in both the 1986 and 1987 seasons. The salaries are for the 1987 season (*Sports Illustrated,* April 20, 1987) and the performance measures are from 1986 (Collier Books, *The 1987 Baseball Encyclopedia Update*).

```
data baseball;
  length name $ 18;
  length team $ 12;
```

```
   input name $ 1-18 nAtBat nHits nHome nRuns nRBI nBB
         yrMajor crAtBat crHits crHome crRuns crRbi crBB
         league $ division $ team $ position $ nOuts nAssts
         nError salary;
   label name="Player's Name"
      nAtBat="Times at Bat in 1986"
      nHits="Hits in 1986"
      nHome="Home Runs in 1986"
      nRuns="Runs in 1986"
      nRBI="RBIs in 1986"
      nBB="Walks in 1986"
      yrMajor="Years in the Major Leagues"
      crAtBat="Career times at bat"
      crHits="Career Hits"
      crHome="Career Home Runs"
      crRuns="Career Runs"
      crRbi="Career RBIs"
      crBB="Career Walks"
      league="League at the end of 1986"
      division="Division at the end of 1986"
      team="Team at the end of 1986"
      position="Position(s) in 1986"
      nOuts="Put Outs in 1986"
      nAssts="Assists in 1986"
      nError="Errors in 1986"
      salary="1987 Salary in $ Thousands";
      logSalary = log(Salary);
   datalines;
Allanson, Andy       293    66     1    30    29    14
                       1   293    66     1    30    29    14
                    American East Cleveland C 446 33 20 .
Ashby, Alan          315    81     7    24    38    39
                      14  3449   835    69   321   414   375
                    National West Houston C 632 43 10 475

   ... more lines ...

Wilson, Willie       631   170     9    77    44    31
                      11  4908  1457    30   775   357   249
                    American West KansasCity CF 408 4 3 1000
;
```

Suppose you want to investigate whether you can model the players' salaries for the 1987 season based on performance measures for the previous season. The aim is to obtain a parsimonious model that does not overfit this particular data, making it useful for prediction. This example shows how you can use PROC HPREG as a starting point for such an analysis. Since the variation of salaries is much greater for the higher salaries, it is appropriate to apply a log transformation to the salaries before doing the model selection.

The following statements select a model with the default settings for stepwise selection:

```
proc hpreg data=baseball;
  class league division;
  model logSalary = nAtBat nHits nHome nRuns nRBI nBB
                    yrMajor crAtBat crHits crHome crRuns crRbi
                    crBB league division nOuts nAssts nError;
  selection method=stepwise;
run;
```

The default output from this analysis is presented in Figure 12.1 through Figure 12.5.

Figure 12.1 Performance, Model, and Selection Information

```
                         The HPREG Procedure

                       Performance Information

                    Execution Mode       On client
                    Number of Threads    4

                          Model Information

               Data Source                 WORK.BASEBALL
               Dependent Variable          logSalary
               Class Parameterization      GLM

                        Selection Information

                Selection Method              Stepwise
                Select Criterion              SBC
                Stop Criterion                SBC
                Effect Hierarchy Enforced     None
                Stop Horizon                  3
```

Figure 12.1 displays the “Performance Information,” “Model Information,” and “Selection Information” tables. The “Performance Information” table shows that procedure executes in client mode—that is, the model is fit on the machine where the SAS session executes. This run of the HPREG procedure was performed on a multicore machine with four CPUs; one computational thread was spawned per CPU.

The “Model Information” table identifies the data source and response and shows that the CLASS variables are parameterized in the GLM parameterization, which is the default.

The “Selection Information” provides details about the method and criteria used to perform the model selection. The requested selection method is a variant of the traditional stepwise selection where the decisions about what effects to add or drop at any step and when to terminate the selection are both based on the Schwarz Bayesian information criterion (SBC). The effect in the current model whose removal yields the maximal decrease in the SBC statistic is dropped provided this lowers the SBC value. When no further decrease in the SBC value can be obtained by dropping an effect in the model, the effect whose addition to the model yields the lowest SBC statistic is added and the whole process is repeated. The method terminates when dropping or adding any effect increases the SBC statistic.

Figure 12.2 displays the "Number of Observations," "Class Levels," and "Dimensions" tables. The "Number of Observations" table shows that of the 322 observations in the input data, only 263 observations are used in the analysis because there are observations with incomplete data. The "Class Level Information" table lists the levels of the classification variables "division" and "league." When you specify effects that contain classification variables, the number of parameters is usually larger than the number of effects. The "Dimensions" table shows the number of effects and the number of parameters considered.

Figure 12.2 Number of Observations, Class Levels, and Dimensions

```
                   Number of Observations Read             322
                   Number of Observations Used             263


                             Class Level Information

                     Class       Levels    Values

                     league           2    American National
                     division         2    East West


                                    Dimensions

                         Number of Effects             19
                         Number of Parameters          21
```

The "Stepwise Selection Summary" table in Figure 12.3 shows the effect that was added or dropped at each step of the selection process together with fit statistics for the model at each step. In this case, both selection and stopping are based on the SBC statistic.

Figure 12.3 Selection Summary Table

```
                              The HPREG Procedure

                               Selection Summary

                         Effect     Effect       Number
                 Step    Entered    Removed  Effects In           SBC

                    0    Intercept                    1      -57.2041
           -----------------------------------------------------------
                    1    crRuns                       2     -194.3166
                    2    nHits                        3     -252.5794
                    3    yrMajor                      4     -262.7322
                    4               crRuns            3     -262.8353
                    5    nBB                          4     -269.7804*

                          * Optimal Value of Criterion
```

Figure 12.4 displays the "Stop Reason," "Selection Reason," and "Selected Effects" tables. Note that these tables are displayed without any titles. The "Stop Reason" table indicates that selection stopped because adding or removing any effect would worsen the SBC value that is used as the selection criterion. In this case, because no CHOOSE= criterion is specified in the SELECTION statement, the final model is the selected model; this is indicated in the “Selection Reason” table. The “Selected Effects” table lists the effects in the selected model.

Figure 12.4 Stopping and Selection Reasons

```
Stepwise selection stopped because adding or removing an effect does not improve
the SBC criterion.

                         The model at step 5 is selected.


                  Selected Effects: Intercept nHits nBB yrMajor
```

The “Analysis of Variance,” “Fit Statistics,” and “Parameter Estimates” tables shown in Figure 12.5 give details of the selected model.

Figure 12.5 Details of the Selected Model

```
                              Analysis of Variance

                                    Sum of           Mean
 Source                  DF        Squares         Square    F Value    Pr > F

 Model                    3      120.52553       40.17518     120.12    <.0001
 Error                  259       86.62820        0.33447
 Corrected Total        262      207.15373

                            Root MSE         0.57834
                            R-Square         0.58182
                            Adj R-Sq         0.57697
                            AIC            -19.06903
                            AICC           -18.83557
                            SBC           -269.78041
                            ASE              0.32938

                              Parameter Estimates

                                                Standard
    Parameter     DF         Estimate              Error    t Value    Pr > |t|

    Intercept      1         4.013911           0.111290      36.07      <.0001
    nHits          1         0.007929           0.000994       7.98      <.0001
    nBB            1         0.007280           0.002049       3.55      0.0005
    yrMajor        1         0.100663           0.007551      13.33      <.0001
```

You might want to examine regression diagnostics for the selected model to investigate whether collinearity among the selected parameters or the presence of outlying or high leverage observations might be impacting the fit produced. The following statements include some options and statements to obtain these diagnostics:

```
proc hpreg data=baseball;
  id name;
  class league division;
  model logSalary = nAtBat nHits nHome nRuns nRBI nBB
                    yrMajor crAtBat crHits crHome crRuns crRbi
                    crBB league division nOuts nAssts nError / vif clb;
  selection method=stepwise;
  output out=baseballOut p=predictedLogSalary r h cookd rstudent;
run;
```

The VIF and CLB options in the MODEL statement request variance inflation factors and 95% confidence limits for the parameter estimates. Figure 12.6 shows the "Parameter Estimates" with these requested statistics. The variance inflation factors (VIF) measure the inflation in the variances of the parameter estimates due to collinearities that exist among the regressor (independent) variables. Although there are no formal criteria for deciding whether a VIF is large enough to affect the predicted values, the VIF values for the selected effects in this example are small enough to indicate that there are no collinearity issues among the selected regressors.

Figure 12.6 Parameter Estimates with Additional Statistics

```
                          The HPREG Procedure
                            Selected Model

                          Parameter Estimates

                                     Standard                          Variance
Parameter    DF       Estimate          Error    t Value    Pr > |t|    Inflation

Intercept     1       4.013911       0.111290      36.07      <.0001            0
nHits         1       0.007929       0.000994       7.98      <.0001      1.49642
nBB           1       0.007280       0.002049       3.55      0.0005      1.52109
yrMajor       1       0.100663       0.007551      13.33      <.0001      1.02488

                          Parameter Estimates

                      Parameter      95% Confidence Limits

                      Intercept       3.79476       4.23306
                      nHits           0.00597       0.00989
                      nBB             0.00325       0.01131
                      yrMajor         0.08579       0.11553
```

By default, High-Performance Analytics procedures do not include all variables from the input data set in output data sets. The ID statement specifies that the variable name in the input data set be added as an identification variable in the baseballOut data set that is produced by the OUTPUT statement. In addition to this variable, the OUTPUT statement requests that predicted values, raw residuals, leverage values, Cook's D statistics, and studentized residuals be added in the output data set. Note that default names are used for these statistics except for the predicted values for which a specified name, predictedLogSalary, is supplied. The following statements use PROC PRINT to display the first five observations of this output data set:

```
proc print data=baseballOut(obs=5);
run;
```

Figure 12.7 First 5 Observations of the baseballOut Data Set

```
                              predicted
 Obs  name                    LogSalary  Residual       H          COOKD  RSTUDENT

   1  Allanson, Andy           4.73980      .       0.016087   .               .
   2  Ashby, Alan              6.34935   -0.18603   0.012645   .000335535  -0.32316
   3  Davis, Alan              5.89993    0.27385   0.019909   .001161794   0.47759
   4  Dawson, Andre            6.50852   -0.29392   0.011060   .000730178  -0.51031
   5  Galarraga, Andres        5.12344   -0.60711   0.009684   .002720358  -1.05510
```

Syntax: HPREG Procedure

The following statements are available in the HPREG procedure:

```
PROC HPREG < options > ;
    CLASS variables ;
    MODEL dependent = < effects > < / model-options > ;
    OUTPUT < OUT=SAS-data-set >
              < keyword < =name > >. . .
              < keyword < =name > > < / options > ;
    PARTITION < partition-options > ;
    PERFORMANCE performance-options ;
    SELECTION selection-options ;
    FREQ variable ;
    ID variables ;
    WEIGHT variable ;
```

The PROC HPREG statement and a single MODEL statement are required. All other statements are optional. The CLASS statement can appear multiple times. If a CLASS statement is specified, it must precede the MODEL statement.

PROC HPREG Statement

PROC HPREG < *options* > **;**

The PROC HPREG statement invokes the procedure. Table 12.1 summarizes the options in the PROC HPREG statement by function.

Table 12.1 PROC HPREG Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
NAMELEN=	Limits the length of effect names
Options Related to Output	
NOPRINT	Suppresses ODS output
NOCLPRINT	Limits or suppresses the display of class levels
User-Defined Formats	
FMTLIBXML=	Specifies a file reference for a format stream
Other Options	
ALPHA=	Sets the significance level used for the construction of confidence intervals
SEED=	Sets the seed used for pseudorandom number generation

Following are explanations of the *options* that you can specify in the PROC HPREG statement (in alphabetical order):

ALPHA=*number*

sets the significance level used for the construction of confidence intervals. The value must be between 0 and 1; the default value of 0.05 results in 95% intervals. This option affects the OUTPUT statement keywords LCL, LCLM, UCL, and UCLM, and the CLB option in the MODEL statement.

DATA=*SAS-data-set*

names the input SAS data set to be used by PROC HPREG. The default is the most recently created data set.

If the procedure executes in MPP mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. In that case the procedure reads the data alongside the distributed database. See the section "SMP and MPP Modes" on page 10 about the various execution modes and the section "Alongside-the-Database Execution" on page 15 about the alongside-the-database model. Both sections are in Chapter 2, "Shared Concepts and Topics."

FMTLIBXML=*file-ref*

specifies the file reference for the XML stream that contains the user-defined format definitions. User-defined formats are handled differently in a distributed computing environment than they are in other

SAS products. See the section "Working with Formats" on page 18 in Chapter 2, "Shared Concepts and Topics," for details about how to generate a XML stream for your formats.

NAMELEN=*number*

specifies the length to which long effect names are shortened. The default and minimum value is 20.

NOCLPRINT< =*number* **>**

suppresses the display of the "Class Level Information" table if you do not specify *number*. If you specify *number*, the values of the classification variables are displayed for only those variables whose number of levels is less than *number*. Specifying a *number* helps to reduce the size of the "Class Level Information" table if some classification variables have a large number of levels.

NOPRINT

suppresses the generation of ODS output.

SEED=*number*

specifies an integer used to start the pseudorandom number generator for random partitioning of data for training, testing, and validation. If you do not specify a seed, or if you specify a value less than or equal to 0, the seed is generated from reading the time of day from the computer's clock.

CLASS Statement

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

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. The CLASS statement must precede the MODEL statement.

The CLASS statement for SAS High-Performance Analytics procedures is documented in the section "CLASS Statement" on page 20 of Chapter 2, "Shared Concepts and Topics." The HPREG procedure also supports the following *global-option* in the CLASS statement:

UPCASE

uppercases the values of character-valued CLASS variables before levelizing them. For example, if the UPCASE option is in effect and a CLASS variable can take the values 'a', 'A', and 'b', then 'a' and 'A' represent the same level and the CLASS variable is treated as having only two values: 'A' and 'B'.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. SAS High-Performance Analytics procedures that support the FREQ statement treat each observation as if it appeared f times, where f is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID *variables* **;**

The ID statement lists one or more variables from the input data set that are transferred to output data sets created by SAS High-Performance Analytics procedures, provided that the output data set produces one (or more) records per input observation.

For documentation on the common ID statement in SAS High-Performance Analytics procedures, see the section "ID Statement" on page 23 in Chapter 2, "Shared Concepts and Topics."

MODEL Statement

MODEL *dependent=< effects > / < options >* **;**

The MODEL statement names the dependent variable and the explanatory effects, including covariates, main effects, interactions, and nested effects. If you omit the explanatory effects, the procedure fits an intercept-only model.

After the keyword MODEL, the dependent (response) variable is specified, followed by an equal sign. The explanatory effects follow the equal sign. For information about constructing the model effects, see the section "Specification and Parameterization of Model Effects" on page 35, of Chapter 2, "Shared Concepts and Topics."

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

CLB

requests the $100(1-\alpha)\%$ upper and lower confidence limits for the parameter estimates. By default, the 95% limits are computed; the ALPHA= option in the PROC HPREG statement can be used to change the α level. The CLB option is not supported when you request METHOD=LAR or METHOD=LASSO in the SELECTION statement.

INCLUDE=*n*

INCLUDE=*single-effect*

INCLUDE=(*effects***)**

forces effects to be included in all models. If you specify INCLUDE=*n*, then the first *n* effects listed in the MODEL statement are included in all models. If you specify INCLUDE=*single-effect* or if you specify a list of effects within parentheses, then the specified effects are forced into all models. The effects that you specify in the INCLUDE= option must be explanatory effects defined in the MODELstatement before the slash (/). The INCLUDE= option is not available when you specify METHOD=LAR or METHOD=LASSO in the SELECTION statement.

NOINT

suppresses the intercept term that is otherwise included in the model.

ORDERSELECT

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

START=*n*

START=*single-effect*

START=(*effects*)

is used to begin the selection process in the FORWARD, FORWARDSWAP, and STEPWISE selection methods from the initial model that you designate. If you specify START=*n*, then the starting model consists of the first *n* effects listed in the MODEL statement. If you specify START=*single-effect* or if you specify a list of effects within parentheses, then the starting model consists of these specified effects. The effects that you specify in the START= option must be explanatory effects defined in the MODELstatement before the slash (/). The START= option is not available when you specify METHOD=BACKWARD, METHOD=LAR, or METHOD=LASSO in the SELECTION statement.

STB

produces standardized regression coefficients. A standardized regression coefficient is computed by dividing a parameter estimate by the ratio of the sample standard deviation of the dependent variable to the sample standard deviation of the regressor.

TOL

produces tolerance values for the estimates. Tolerance for a parameter is defined as $1 - R^2$, where R^2 is obtained from the regression of the parameter on all other parameters in the model. The TOL option is not supported when you request METHOD=LAR or METHOD=LASSO in the SELECTION statement.

VIF

produces variance inflation factors with the parameter estimates. Variance inflation is the reciprocal of tolerance. The VIF option is not supported when you request METHOD=LAR or METHOD=LASSO in the SELECTION statement.

PARTITION Statement

PARTITION < *partition-options* > **;**

The PARTITION statement specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing. Either you can designate a variable in the input data set and a set of formatted values of that variable to determine the role of each observation, or you can specify proportions to use for random assignment of observations for each role.

The following mutually exclusive *partition-options* are available:

ROLEVAR | ROLE=*variable***(< TEST='***value***' > < TRAIN='***value***' > < VALIDATE='***value***' >)**

names the variable in the input data set whose values are used to assign roles to each observation. The formatted values of this variable that are used to assign observations roles are specified in the TEST=, TRAIN=, and VALIDATE= suboptions. If you do not specify the TRAIN= suboption, then all observations whose role is not determined by the TEST= or VALIDATE= suboptions are assigned to training.

FRACTION(< TEST=*fraction* **> < VALIDATE=***fraction* **>)**

requests that specified proportions of the observations in the input data set be randomly assigned training and validation roles. You specify the proportions for testing and validation by using the TEST= and VALIDATE= suboptions. If you specify both the TEST= and the VALIDATE= suboptions, then the sum of the specified fractions must be less than 1 and the remaining fraction of the observations are assigned to the training role.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

With the PERFORMANCE statement you can also control whether a SAS High-Performance Analytics procedure executes in SMP or MPP mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

SELECTION Statement

SELECTION < *options* > **;**

The SELECTION statement performs variable selection. The statement is fully documented in the section "SELECTION Statement" on page 24 in Chapter 2, "Shared Concepts and Topics."

The HPREG procedure supports the following variable selection methods in the METHOD= option in the SELECTION statement:

NONE	No model selection.
FORWARD	The forward selection method starts with no effects in the model and adds effects.
BACKWARD	The backward elimination method starts with all effects in the model and deletes effects.
STEPWISE	The stepwise regression method is similar to the FORWARD method except that effects already in the model do not necessarily stay there.

FORWARDSWAP	The forward-swap selection method is an extension of the forward selection method. Before any addition step, PROC HPREG makes all pairwise swaps of effects in and out of the current model that improve the selection criterion. When the selection criterion is R square, this method is the same as the MAXR method in the REG procedure in SAS/STAT software.
LAR	The least angle regression method, like forward selection, starts with no effects in the model and adds effects. The parameter estimates at any step are "shrunk" when compared to the corresponding least squares estimates. If the model contains classification variables, then these classification variables are split. See the SPLIT option in the CLASS statement for details.
LASSO	The lasso method adds and deletes parameters based on a version of ordinary least squares where the sum of the absolute regression coefficients is constrained. If the model contains classification variables, then these classification variables are split. See the SPLIT option in the CLASS statement for details.

The DETAILS=ALL and DETAILS=STEPS options produce the "ANOVA," "Fit Statistics," and "Parameter Estimates" tables, which provide information about the model that is selected at each step of the selection process.

OUTPUT Statement

OUTPUT < **OUT=***SAS-data-set*>
    < *keyword* < *=name* > >. . . < *keyword* < *=name* > > **;**

The OUTPUT statement creates a data set that contains observationwise statistics, which are computed after fitting the model. The variables in the input data set are *not* included in the output data set to avoid data duplication for large data sets.

If the input data are in distributed form, where access of data in a particular order cannot be guaranteed, the HPREG procedure copies the distribution or partition key to the output data set so that its contents can be joined with the input data.

The output statistics are computed based on the parameter estimates for the selected model.

You can specify the following syntax elements in the OUTPUT statement:

OUT=*SAS-data-set*
DATA=*SAS-data-set*

specifies the name of the output data set. If the OUT= (or DATA=) option is omitted, the procedure uses the DATA*n* convention to name the output data set.

keyword **<** *=name* **>**

specifies the statistics to include in the output data set and optionally names the new variables that contain the statistics. Specify a keyword for each desired statistic (see the following list of keywords), followed optionally by an equal sign and a variable to contain the statistic.

If you specify *keyword=name*, the new variable that contains the requested statistic has the specified name. If you omit the optional *=name* after a *keyword*, then a default name is used.

The following are valid values for *keyword* to request statistics that are available with all selection methods:

PREDICTED

PRED

P

requests predicted values for the response variable. The default name is `pred`.

RESIDUAL

RESID

R

requests the residual, calculated as ACTUAL–PREDICTED. The default name is `residual`.

ROLE

requests a numeric variable that indicates the role played by each observation in fitting the model. The default name is `role`. For each observation the interpretation of this variable is shown in Table 12.2:

Table 12.2 Role Interpretation

Value	Observation Role
0	Not used
1	Training
2	Validation
3	Testing

If you do not partition the input data by using a PARTITION statement, then the role variable value is 1 for observations used in fitting the model, and 0 for observations that have at least one missing or invalid value for the response, regressors, frequency or weight variables.

In addition to the preceding statistics, you can also use the *keywords* listed in Table 12.3 in the OUTPUT statement to obtain additional statistics. These statistics are not available if you use METHOD=LAR or METHOD=LASSO in the SELECTION statement, unless you also specify the LSCOEFFS option. See the section "Diagnostic Statistics" on page 336 for computational formulas. All the statistics available in the OUTPUT statement are conditional on the selected model and do not take into account the variability introduced by doing model selection.

Table 12.3 Keywords for OUTPUT Statement

Keyword	Description
COOKD	Cook's D influence statistic
COVRATIO	Standard influence of observation on covariance of betas
DFFIT	Standard influence of observation on predicted value
H	Leverage, $x_i(\mathbf{X}'\mathbf{X})^{-1}x_i'$

Table 12.3 *continued*

Keyword	Description
LCL	Lower bound of a $100(1-\alpha)\%$ confidence interval for an individual prediction. This includes the variance of the error, as well as the variance of the parameter estimates.
LCLM	Lower bound of a $100(1-\alpha)\%$ confidence interval for the expected value (mean) of the dependent variable
PRESS	ith residual divided by $(1-h)$, where h is the leverage, and where the model has been refit without the ith observation
RSTUDENT	A studentized residual with the current observation deleted
STDI	Standard error of the individual predicted value
STDP	Standard error of the mean predicted value
STDR	Standard error of the residual
STUDENT	Studentized residuals, which are the residuals divided by their standard errors
UCL	Upper bound of a $100(1-\alpha)\%$ confidence interval for an individual prediction
UCLM	Upper bound of a $100(1-\alpha)\%$ confidence interval for the expected value (mean) of the dependent variable

WEIGHT Statement

WEIGHT *variable* **;**

The *variable* in the WEIGHT statement is used as a weight to perform a weighted analysis of the data. Observations with nonpositive or missing weights are not included in the analysis. If a WEIGHT statement is not included, all observations used in the analysis are assigned a weight of 1.

Details: HPREG Procedure

Criteria Used in Model Selection

The HPREG procedure supports a variety of fit statistics that you can specify as criteria for the CHOOSE=, SELECT=, and STOP= options in the SELECTION statement. The following statistics are available:

ADJRSQ	Adjusted R-square statistic (Darlington 1968; Judge et al. 1985)
AIC	Akaike's information criterion (Akaike 1969; Judge et al. 1985)
AICC	Corrected Akaike's information criterion (Hurvich and Tsai 1989)

BIC | SBC	Schwarz Bayesian information criterion (Schwarz 1978; Judge et al. 1985)
CP	Mallows C_p statistic (Mallows 1973; Hocking 1976)
PRESS	Predicted residual sum of squares statistic
RSQUARE	R-square statistic (Darlington 1968; Judge et al. 1985)
SL	Significance used to assess an effect's contribution to the fit when it is added to or removed from a model
VALIDATE	Average square error over the validation data

When you use SL as a criterion for effect selection, the definition depends on whether an effect is being considered as a drop or an add candidate. If the current model has p parameters excluding the intercept, and if you denote its residual sum of squares by RSS_p and you add an effect with k degrees of freedom and denote the residual sum of squares of the resulting model by RSS_{p+k}, then the F statistic for entry with k numerator degrees of freedom and $n-(p+k)-1$ denominator degrees of freedom is given by

$$F = \frac{(\text{RSS}_p - \text{RSS}_{p+k})/k}{\text{RSS}_{p+k}/(n-(p+k)-1)}$$

where n is number of observations used in the analysis. The significance level for entry is the p-value of this F statistic, and is deemed significant if it is smaller than the SLENTRY limit. Among several such add candidates, the effect with the smallest p-value (most significant) is deemed best.

If you drop an effect with k degrees of freedom and denote the residual sum of squares of the resulting model by RSS_{p-k}, then the F statistic for removal with k numerator degrees of freedom and $n-p-k$ denominator degrees of freedom is given by

$$F = \frac{(\text{RSS}_{p-k} - \text{RSS}_p)/k}{\text{RSS}_p/(n-p-k)}$$

where n is number of observations used in the analysis. The significance level for removal is the p-value of this F statistic, and the effect is deemed not significant if this p-value is larger than the SLSTAY limit. Among several such removal candidates, the effect with the largest p-value (least significant) is deemed the best removal candidate.

It is known that the "F-to-enter" and "F-to-delete" statistics do not follow an F distribution (Draper, Guttman, and Kanemasu 1971).. Hence the SLENTRY and SLSTAY values cannot reliably be viewed as probabilities. One way to address this difficulty is to replace hypothesis testing as a means of selecting a model with information criteria or out-of-sample prediction criteria. While Harrell (2001) points out that information criteria were developed for comparing only prespecified models, Burnham and Anderson (2002) note that AIC criteria have routinely been used for several decades for performing model selection in time series analysis.

Table 12.4 provides formulas and definitions for these fit statistics.

Table 12.4 Formulas and Definitions for Model Fit Summary Statistics

Statistic	Definition or Formula
n	Number of observations
p	Number of parameters including the intercept
$\hat{\sigma}^2$	Estimate of pure error variance from fitting the full model

Table 12.4 *continued*

Statistic	Definition or Formula
SST	Total sum of squares corrected for the mean for the dependent variable
SSE	Error sum of squares
ASE	$\frac{\text{SSE}}{n}$
MSE	$\frac{\text{SSE}}{n-p}$
R^2	$1 - \frac{\text{SSE}}{\text{SST}}$
ADJRSQ	$1 - \frac{(n-1)(1-R^2)}{n-p}$
AIC	$n \ln\left(\frac{\text{SSE}}{n}\right) + 2p$
AICC	$1 + \ln\left(\frac{\text{SSE}}{n}\right) + \frac{2(p+1)}{n-p-2}$
CP (C_p)	$\frac{\text{SSE}}{\hat{\sigma}^2} + 2p - n$
PRESS	$\sum_{i=1}^{n} \frac{r_i^2}{(1-h_i)^2}$ where $r_i =$ residual at observation i and $h_i =$ leverage of observation $i = \mathbf{x}_i(\mathbf{X}'\mathbf{X})^{-}\mathbf{x}_i'$
RMSE	$\sqrt{\text{MSE}}$
SBC	$n \ln\left(\frac{\text{SSE}}{n}\right) + p\ln(n)$

Diagnostic Statistics

This section gathers the formulas for the statistics available in the OUTPUT statement. All the statistics available in the OUTPUT statement are conditional on the selected model and do not take into account the variability introduced by doing model selection.

The model to be fit is $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, and the parameter estimate is denoted by $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-}\mathbf{X}'\mathbf{Y}$. The subscript i denotes values for the ith observation, and the parenthetical subscript (i) means that the statistic is computed by using all observations except the ith observation.

The ALPHA= option in the PROC HPREG statement is used to set the α value for the confidence limit statistics.

Table 12.5 contains the diagnostic statistics and their formulas. Each statistic is computed for each observation.

Table 12.5 Formulas and Definitions for Diagnostic Statistics

MODEL Option or Statistic	Formula
PRED ($\widehat{\mathbf{Y}}_i$)	$\mathbf{X}_i\mathbf{b}$
RES (r_i)	$\mathbf{Y}_i - \widehat{\mathbf{Y}}_i$
H (h_i)	$\mathbf{x}_i(\mathbf{X}'\mathbf{X})^{-}\mathbf{x}_i'$
STDP	$\sqrt{h_i\widehat{\sigma}^2}$
STDI	$\sqrt{(1+h_i)\widehat{\sigma}^2}$
STDR	$\sqrt{(1-h_i)\widehat{\sigma}^2}$
LCL	$\widehat{Y}_i - t_{\frac{\alpha}{2}}\text{STDI}$
LCLM	$\widehat{Y}_i - t_{\frac{\alpha}{2}}\text{STDP}$
UCL	$\widehat{Y}_i + t_{\frac{\alpha}{2}}\text{STDI}$
UCLM	$\widehat{Y}_i + t_{\frac{\alpha}{2}}\text{STDP}$
STUDENT	$\frac{r_i}{\text{STDR}_i}$
RSTUDENT	$\frac{r_i}{\hat{\sigma}_{(i)}\sqrt{1-h_i}}$
COOKD	$\frac{1}{p}\text{STUDENT}^2\frac{\text{STDP}^2}{\text{STDR}^2}$
COVRATIO	$\frac{\det(\hat{\sigma}^2_{(i)}(\mathbf{x}'_{(i)}\mathbf{x}_{(i)})^{-1}}{\det(\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1})}$
DFFITS	$\frac{(\widehat{\mathbf{Y}}_i - \widehat{\mathbf{Y}}_{(i)})}{(\hat{\sigma}_{(i)}\sqrt{h_i})}$
PRESS(predr$_i$)	$\frac{r_i}{1-h_i}$

Classification Variables and the SPLIT Option

PROC HPREG supports the ability to split classification variables when doing model selection. You use the SPLIT option in the CLASS statement to specify that the columns of the design matrix that correspond to effects that contain a split classification variable can enter or leave a model independently of the other design columns of that effect. The following statements illustrate the use of SPLIT option:

```
data splitExample;
   length c2 $6;
   drop i;
   do i=1 to 1000;
     c1 = 1 + mod(i,6);
     if      i < 250 then c2 = 'low';
     else if i < 500 then c2 = 'medium';
     else                 c2 = 'high';
```

```
         x1 = ranuni(1);
         x2 = ranuni(1);
         y = x1+3*(c2 ='low')  + 10*(c1=3) +5*(c1=5) + rannor(1);
         output;
      end;
   run;

   proc hpreg data=splitExample;
      class c1(split) c2(order=data);
      model y = c1 c2 x1 x2/orderselect;
      selection method=forward;
   run;
```

The “Class Levels” table shown in Figure 12.8 is produced by default whenever you specify a CLASS statement.

Figure 12.8 Class Levels

```
                              The HPREG Procedure

                            Class Level Information

                      Class     Levels      Values

                      c1             6 *    1 2 3 4 5 6
                      c2             3      low medium high

                          * Associated Parameters Split
```

The SPLIT option has been specified for the classification variable c1. This permits the parameters associated with the effect c1 to enter or leave the model individually. The “Parameter Estimates” table in Figure 12.9 shows that for this example the parameters that correspond to only levels 3 and 5 of c1 are in the selected model. Finally, note that the ORDERSELECT option in the MODEL statement specifies that the parameters be displayed in the order in which they first entered the model.

Figure 12.9 Parameter Estimates

```
                              Parameter Estimates

                                                    Standard
      Parameter     DF        Estimate                 Error     t Value     Pr > |t|

      Intercept      1       -0.308111              0.075387       -4.09       <.0001
      c1_3           1       10.161702              0.087601      116.00       <.0001
      c1_5           1        5.018407              0.087587       57.30       <.0001
      c2 low         1        3.139941              0.078495       40.00       <.0001
      c2 medium      1        0.221539              0.078364        2.83       0.0048
      c2 high        0               0                     .           .        .
      x1             1        1.317420              0.109510       12.03       <.0001
```

Using Validation and Test Data

When you have sufficient data, you can subdivide your data into three parts called the training, validation, and test data. During the selection process, models are fit on the training data, and the prediction error for the models so obtained is found by using the validation data. This prediction error on the validation data can be used to decide when to terminate the selection process or to decide what effects to include as the selection process proceeds. Finally, after a selected model has been obtained, the test set can be used to assess how the selected model generalizes on data that played no role in selecting the model.

In some cases you might want to use only training and test data. For example, you might decide to use an information criterion to decide what effects to include and when to terminate the selection process. In this case no validation data are required, but test data can still be useful in assessing the predictive performance of the selected model. In other cases you might decide to use validation data during the selection process but forgo assessing the selected model on test data. Hastie, Tibshirani, and Friedman (2001) note that it is difficult to give a general rule for how many observations you should assign to each role. They note that a typical split might be 50% for training and 25% each for validation and testing.

You use a PARTITION statement to logically subdivide the DATA= data set into separate roles. You can name the fractions of the data that you want to reserve as test data and validation data. For example, the following statements randomly subdivide the "inData" data set, reserving 50% for training and 25% each for validation and testing:

```
proc hpreg data=inData;
  partition fraction(test=0.25 validate=0.25);
  ...
run;
```

In some cases you might need to exercise more control over the partitioning of the input data set. You can do this by naming a both variable in the input data set and also a formatted value of that variable that correspond to each role. For example, the following statements assign roles to the observations in the "inData" data set based on the value of the variable group in that data set. Observations where the value of group is 'group 1' are assigned for testing, and those with value 'group 2' are assigned to training. All other observations are ignored.

```
proc hpreg data=inData;
  partition roleVar=group(test='group 1' train='group 2')
  ...
run;
```

When you have reserved observations for training, validation, and testing, a model fit on the training data is scored on the validation and test data, and the average squared error (ASE) is computed separately for each of these subsets. The ASE for each data role is the error sum of squares for observations in that role divided by the number of observations in that role.

Using the Validation ASE as the STOP= Criterion

If you have provided observations for validation, then you can specify STOP=VALIDATE as a suboption of the METHOD= option in the SELECTION statement. At step *k* of the selection process, the best candidate effect to enter or leave the current model is determined. Here "best candidate" means the effect that gives the best value of the SELECT= criterion; this criterion need not be based on the validation data. The validation ASE for the model with this candidate effect added or removed is computed. If this validation ASE is greater than the validation ASE for the model at step k, then the selection process terminates at step k.

Using the Validation ASE as the CHOOSE= Criterion

When you specify the CHOOSE=VALIDATE suboption of the METHOD= option in the SELECTION statement, the validation ASE is computed for the models at each step of the selection process. The smallest model at any step that yields the smallest validation ASE is selected.

Using the Validation ASE as the SELECT= Criterion

You request the validation ASE as the selection criterion by specifying the SELECT=VALIDATE suboption of the METHOD= option in the SELECTION statement. At step k of the selection process, the validation ASE is computed for each model in which a candidate for entry is added or candidate for removal is dropped. The selected candidate for entry or removal is the one that yields a model with the minimal validation ASE. This method is computationally very expensive because validation statistics need to be computed for every candidate at every step; it should be used only with small data sets or models with a small number of regressors.

Computational Method

Multithreading

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

The number of threads spawned by the HPREG procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statements, the HPREG procedure schedules threads as if it executes on a system with four CPUs, regardless of the actual CPU count.

  ```
  options cpucount=4;
  ```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This specification overrides the system option. Specify NTHREADS=1 to force single-threaded execution.

- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement, which overrides the CPUCOUNT system option and instructs the HPREG procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output. The HPREG procedure allocates one thread per CPU.

The tasks multithreaded by the HPREG procedures are primarily defined by dividing the data processed on a single machine among the threads—that is, the HPREG procedure implements multithreading through a data-parallel model. For example, if the input data set has 1,000 observations and you are running with four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded. This operations include the following:

- variable levelization
- effect levelization
- formation of the crossproducts matrix
- evaluation of predicted residual sums of squares on validation and test data
- scoring of observations

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

Output Data Set

Many procedures in SAS software add the variables from the input data set when an observationwise output data set is created. The assumption of High-Performance Analytics procedures is that the input data sets can be large and contain many variables. For performance reasons, the output data set contains the following:

- those variables explicitly created by the statement
- variables listed in the ID statement
- distribution keys or hash keys that are transferred from the input data set

This enables you to add output data set information that is necessary for subsequent SQL joins without copying the entire input data set to the output data set. For further details about output data sets when PROC HPREG is run in distributed (MPP) mode, see the section "Output Data Sets" on page 18 in Chapter 2, "Shared Concepts and Topics."

Displayed Output

The following sections describe the output produced by PROC HPREG. The output is organized into various tables, which are discussed in the order of appearance.

Performance Information

The “Performance Information” table is produced by default and displays information about the grid host for distributed execution and about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also displayed, depending on the environment.

Model Information

The “Model Information” table displays basic information about the model, such as the response variable, frequency variable, weight variable, and the type of parameterization used for classification variables named in the CLASS statement.

Selection Information

When you specify the SELECTION statement, the HPREG procedure produces by default a series of tables with information about the model selection. The “Selection Information” table informs you about the model selection method; select, stop, and choose criteria; and other parameters that govern the selection. You can suppress this table by specifying DETAILS=NONE in the SELECTION statement.

Number of Observations

The “Number of Observations” table displays the number of observations read from the input data set and the number of observations used in the analysis. If you specify a FREQ statement, the table also displays the sum of frequencies read and used. If you use a PARTITION statement, the table also displays the number of observations used for each data role.

Class Level Information

The “Class Level Information” table lists the levels of every variable specified in the CLASS statement. You should check this information to make sure that the data are correct. You can adjust the order of the CLASS variable levels with the ORDER= option in the CLASS statement. You can suppress the “Class Level Information” table completely or partially with the NOCLPRINT= option in the PROC HPREG statement.

If the classification variables are in the reference parameterization, the “Class Level Information” table also displays the reference value for each variable. The “Class Level Information” table also indicates which, if any, of the classification variables are split by using the SPLIT option in the CLASS statement.

Dimensions

The “Dimensions” table displays information about the number of effects and the number of parameters from which the selected model is chosen. If you use split classification variables, then this table also includes the number of effects after splitting is taken into account.

Entry and Removal Candidates

When you specify the DETAILS=ALL or DETAILS=STEPS option in the SELECTION statement, the HPREG procedure produces “Entry Candidates” and “Removal Candidates” tables that display the effect names and values of the criterion used to select entering or departing effects at each step of the selection process. The effects are displayed in sorted order from best to worst of the selection criterion.

Selection Summary

When you specify the SELECTION statement, the HPREG procedure produces the “Selection Summary” table with information about the sequence of steps of the selection process. For each step, the effect that was entered or dropped is displayed along with the statistics used to select the effect, stop the selection, and choose the selected model. For all criteria that you can use for model selection, the steps at which the optimal values of these criteria occur are also indicated.

The display of the “Selection Summary” table can be suppressed by specifying DETAILS=NONE in the SELECTION statement.

Stop Reason

The “Stop Reason” table displays the reason why the selection stopped. To facilitate programmatic use of this table, an integer code is assigned to each reason and is included if you output this table by using an ODS OUTPUT statement. The reasons and their associated codes follow:

Code	**Stop Reason**
1	All eligible effects are in the model.
2	All eligible effects have been removed.
3	Specified maximum number of steps done.
4	The model contains the specified maximum number of effects.
5	The model contains the specified minimum number of effects (for backward selection).
6	The stopping criterion is at a local optimum.
7	No suitable add or drop candidate could be found.
8	Adding or dropping any effect does not improve the selection criterion.
9	No candidate meets the appropriate SLE or SLS significance level.
10	Stepwise selection is cycling.
11	The model is an exact fit.
12	Dropping an effect would result in an empty model.

The display of the "Stop Reason" table can be suppressed by specifying DETAILS=NONE in the SELECTION statement.

Selection Reason

When you specify the SELECTION statement, the HPREG procedure produces a simple table that contains text informing you about the reason why the final model was selected.

The display of the "Selection Reason" table can be suppressed by specifying DETAILS=NONE in the SELECTION statement.

Selected Effects

When you specify the SELECTION statement, the HPREG procedure produces a simple table that contains text informing you about which effects were selected into the final model.

ANOVA

The "ANOVA" table displays an analysis of variance for the selected model. This table includes the following:

- the Source of the variation, Model for the fitted regression, Error for the residual error, and C Total for the total variation after correcting for the mean. The Uncorrected Total Variation is produced when the NOINT option is used.
- the degrees of freedom (DF) associated with the source
- the Sum of Squares for the term
- the Mean Square, the sum of squares divided by the degrees of freedom
- the *F* Value for testing the hypothesis that all parameters are 0 except for the intercept. This is formed by dividing the mean square for Model by the mean square for Error.
- the Prob>*F*, the probability of getting a greater *F* statistic than that observed if the hypothesis is true. When you do model selection, these *p*-values are generally liberal because they are not adjusted for the fact that the terms in the model have been selected.

You can request "ANOVA" tables for the model at each step of the selection process with the DETAILS= option in the SELECTION statement.

Fit Statistics

The "Fit Statistics" table displays fit statistics for the selected model. The statistics displayed include the following:

- Root MSE, an estimate of the standard deviation of the error term. It is calculated as the square root of the mean square error.
- R-square, a measure between 0 and 1 that indicates the portion of the (corrected) total variation attributed to the fit rather than left to residual error. It is calculated as SS(Model) divided by SS(Total). It is also called the *coefficient of determination*. It is the square of the multiple correlation—in other words, the square of the correlation between the dependent variable and the predicted values.
- Adj R-Sq, the adjusted R-square, a version of R-square that has been adjusted for degrees of freedom. It is calculated as

  $$\bar{R}^2 = 1 - \frac{(n-i)(1-R^2)}{n-p}$$

  where i is equal to 1 if there is an intercept and 0 otherwise, n is the number of observations used to fit the model, and p is the number of parameters in the model.
- fit criteria AIC, AICC, BIC, CP, and PRESS if they are used in the selection process. See Table 12.4 for the formulas for evaluating these criteria.
- the average square errors (ASE) on the training, validation, and test data.

You can request "Fit Statistics" tables for the model at each step of the selection process with the DETAILS= option in the SELECTION statement.

Parameter Estimates

The "Parameter Estimates" table displays the parameters in the selected model and their estimates. The information displayed for each parameter in the selected model includes the following:

- the parameter label that includes the effect name and level information for effects that contain classification variables
- the degrees of freedom (DF) for the parameter. There is one degree of freedom unless the model is not full rank.
- the parameter estimate
- the standard error, which is the estimate of the standard deviation of the parameter estimate
- t Value, the t test that the parameter is 0. This is computed as the parameter estimate divided by the standard error.
- the Pr > |t|, the probability that a t statistic would obtain a greater absolute value than that observed given that the true parameter is 0. This is the two-tailed significance probability.

  When you do model selection, these p-values are generally liberal because they are not adjusted for the fact that the terms in the model have been selected.

You can request "Parameter Estimates" tables for the model at each step of the selection process with the DETAILS= option in the SELECTION statement.

Timing Information

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a "Timing" table in which the elapsed time for each main task of the procedure is displayed.

ODS Table Names

Each table created by the HPREG procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 12.6.

Table 12.6 ODS Tables Produced by PROC HPREG

Table Name	Description	Required Statement / Option
ANOVA	Selected model ANOVA table	Default output
Candidates	Swap candidates at step	SELECTION DETAILS=ALL|STEPS
ClassLevels	Level information from the CLASS statement	CLASS
Dimensions	Model dimensions	Default output
EntryCandidates	Candidates for entry at step	SELECTION DETAILS=ALL|STEPS
FitStatistics	Fit statistics	Default output
ModelInfo	Information about the modeling environment	Default output
NObs	Number of observations read and used	Default output
ParameterEstimates	Solutions for the parameter estimates associated with effects in MODEL statement	Default output
PerformanceInfo	Information about high-performance computing environment	Default output
RemovalCandidates	Candidates for removal at step	SELECTION DETAILS=ALL|STEPS
SelectedEffects	List of selected effects	SELECTION
SelectionInfo	Information about selection settings	Default output
SelectionReason	Reason for selecting the final model	SELECTION
SelectionSummary	Summary information about the model selection steps	SELECTION
StopReason	Reason selection was terminated	SELECTION
Timing	Timing breakdown by task	SELECTION DETAILS

Examples: HPREG Procedure

Example 12.1: Model Selection with Validation

This example is based on the example "Using Validation and Cross Validation" in the documentation for the GLMSELECT procedure in the *SAS/STAT User's Guide*. This example shows how you can use validation data to monitor and control variable selection. It also demonstrates the use of split classification variables.

The following DATA step produces analysis data that contains a variable that you can use to assign observations to the training, validation, and testing roles. In this case, each role has 5,000 observations.

```
data analysisData;
   drop i j c3Num;
   length c3$ 7;

   array x{20} x1-x20;

   do i=1 to 15000;
      do j=1 to 20;
         x{j} = ranuni(1);
      end;

      c1 = 1 + mod(i,8);
      c2 = ranbin(1,3,.6);

      if      i < 50   then do; c3 = 'tiny';     c3Num=1;end;
      else if i < 250  then do; c3 = 'small';    c3Num=1;end;
      else if i < 600  then do; c3 = 'average';  c3Num=2;end;
      else if i < 1200 then do; c3 = 'big';      c3Num=3;end;
      else                  do; c3 = 'huge';     c3Num=5;end;

      yTrue = 10 + x1 + 2*x5 + 3*x10 + 4*x20  + 3*x1*x7 + 8*x6*x7
                 + 5*(c1=3)*c3Num + 8*(c1=7);

      error = 5*rannor(1);

      y = yTrue + error;

           if mod(i,3)=1 then Role = 'TRAIN';
      else if mod(i,3)=2 then Role = 'VAL';
      else                    Role = 'TEST';

      output;
  end;
run;
```

By construction, the true model consists of main effects x1, x5, x10, x20, and c1 and interaction effects x1*x7, x6*x7, and c1*c3. Furthermore, you can see that only levels 3 and 7 of the classification variable c1 are systematically related to the response.

Because the error term for each observation is five times a value drawn from a standard normal distribution, the expected error variance is 25. For the data in each role, you can compute an estimate of this error variance by forming the average square error (ASE) for the observations in the role. Output 12.1.1 shows the ASE for each role that you can compute with the following statements:

```
proc summary data=analysisData;
   class role;
   ways 1;
   var error;
   output out=ASE uss=uss n=n;
data ASE; set ASE;
   OracleASE = uss / n;
   label OracleASE = 'Oracle ASE';
   keep Role OracleASE;
proc print data=ASE label noobs;
run;

proc print data=ASE label noobs;
run;
```

Output 12.1.1 Oracle ASE Values by Role

```
                    Oracle
         Role        ASE

         TEST     25.5784
         TRAIN    25.4008
         VAL      25.8993
```

The ASE values shown Output 12.1.1 are labeled as “Oracle ASE” because you need to know the true underlying model if you want to compute these values from the response and underlying regressors. In a modeling context, a good predictive model produces values that are close to these oracle values. An overfit model produces a smaller ASE on the training data but higher values on the validation and test data. An underfit model exhibits higher values for all data roles.

Suppose you suspect that the dependent variable depends on both main effects and two-way interactions. You can use the following statements to select a model:

```
proc hpreg data=analysisData;
   partition roleVar=role(train='TRAIN' validate='VAL' test='TEST');
   class c1 c2 c3(order=data);
   model y =  c1|c2|c3|x1|x2|x3|x4|x5|x5|x6|x7|x8|x9|x10
            |x11|x12|x13|x14|x15|x16|x17|x18|x19|x20 @2 /stb;
   selection method = stepwise(select=sl sle=0.1 sls=0.15 choose=validate)
                      hierarchy=single details=steps;
run;
```

A PARTITION statement assigns observations to training, validation, and testing roles based on the values of the input variable named role. The SELECTION statement requests STEPWISE selection based on significance level where the SLE and SLS values are set to use the defaults of PROC REG. The CHOOSE=VALIDATE option selects the model that yields the smallest ASE value on the validation data.

The "Number Of Observation" table in Output 12.1.2 confirms that there are 5,000 observations for each data role. The "Dimensions" table shows that the selection is from 278 effects with a total of 661 parameters.

Output 12.1.2 Number of Observations, Class Levels, and Dimensions

```
                              The HPREG Procedure

            Number of Observations Read                             15000
            Number of Observations Used                             15000
            Number of Observations Used for Training                 5000
            Number of Observations Used for Validation               5000
            Number of Observations Used for Testing                  5000

                            Class Level Information

                 Class    Levels    Values

                 c1            8    1 2 3 4 5 6 7 8
                 c2            4    0 1 2 3
                 c3            5    tiny small average big huge

                                  Dimensions

                         Number of Effects            278
                         Number of Parameters         661
```

Output 12.1.3 shows the "Selection Summary" table. You see that 18 steps are done, at which point all effects in the model are significant at the SLS value of 0.15 and all the remaining effects if added individually would not be significant at the SLE significance level of 0.1. However, because you have specified the CHOOSE=VALIDATE option, the model at step 18 is not used as the selected model. Instead the model at step 10 (where the validation ASE achieves a local minimum value) is selected. The "Stop Reason," "Selection Reason," and "Selected Effects" in Output 12.1.4 provide this information.

Output 12.1.3 Selection Summary

```
                              The HPREG Procedure

                                Selection Summary

                       Effect          Number    Validation         p
               Step    Entered    Effects In           ASE      Value

                  0    Intercept            1       98.3895     1.0000
          -------------------------------------------------------------
                  1    c1                   2       34.8572     <.0001
                  2    x7                   3       32.5531     <.0001
                  3    x6                   4       31.0646     <.0001
                  4    x20                  5       29.7078     <.0001
                  5    x6*x7                6       29.2210     <.0001
                  6    x10                  7       28.6683     <.0001
                  7    x1                   8       28.3250     <.0001
                  8    x5                   9       27.9766     <.0001
                  9    c3                  10       27.8288     <.0001
                 10    c1*c3               11       25.9701*    <.0001
                 11    x10*c1              12       26.0696     0.0109
                 12    x4                  13       26.1594     0.0128
                 13    x4*x10              14       26.1814     0.0035
                 14    x20*c1              15       26.3294     0.0156
                 15    x1*c3               16       26.3945     0.0244
                 16    x1*x7               17       26.3632     0.0270
                 17    x7*x10              18       26.4120     0.0313
                 18    x1*x20              19       26.4330     0.0871

                           * Optimal Value of Criterion
```

Output 12.1.4 Stopping and Selection Reasons

```
Selection stopped because all candidates for removal are significant at the 0.15
level and no candidate for entry is significant at the 0.1 level.

        The model at step 10 where Validation ASE is 25.9701 is selected.

       Selected Effects: Intercept c1 c3 c1*c3 x1 x5 x6 x7 x6*x7 x10 x20
```

You can see that the selected effects include all the main effects in the true model and two of the three true interaction terms. Furthermore, the selected model does not include any variables that are not in the true model. Note that these statements are not true of the larger model at the final step of the selection process.

Output 12.1.5 shows the fit statistics of the selected model. You can see that the ASE values on the training, validation, and test data are all similar, which is indicative of a reasonable predictive model. In this case where the true model is known, you can see that all three ASE values are close to oracle values for the true model, as shown in Output 12.1.1.

Output 12.1.5 Fit Statistics for the Selected Model

```
Root MSE              5.03976
R-Square              0.74483
Adj R-Sq              0.74246
AIC                     21222
AICC                    21223
SBC                     16527
ASE (Train)          25.16041
ASE (Validate)       25.97010
ASE (Test)           25.83436
```

Because you specified the DETAILS=STEPS option in the SELECTION statement, you can see the "Fit Statistics" for the model at each step of the selection process. Output 12.1.6 shows these fit statistics for final model at step 18. You see that for this model, the ASE value on the training data is smaller than the ASE values on the validation and test data. This is indicative an overfit model that might not generalize well to new data. You see the ASE values on the validation and test data are now worse in comparison to the oracle values than the values for the selected model at step 10.

Output 12.1.6 Fit Statistics for the Model at Step 18

```
Root MSE              5.01386
R-Square              0.74862
Adj R-Sq              0.74510
AIC                     21194
AICC                    21196
SBC                     16648
ASE (Train)          24.78688
ASE (Validate)       26.43304
ASE (Test)           26.07078
```

Output 12.1.7 shows part of the "Parameter Estimates" table for the selected model at step 10 that includes the estimates for the main effect c1. Because the STB option is specified in the MODEL statement, this table includes standardized estimates.

Output 12.1.7 Part of the Parameter Estimates Table for the Selected Model

```
                                 Parameter Estimates

                                        Standardized      Standard
Parameter         DF      Estimate          Estimate         Error   t Value   Pr > |t|

Intercept          1      9.479114                 0      0.422843     22.42     <.0001
c1 1               1      0.279417          0.009306      0.297405      0.94     0.3475
c1 2               1      0.615589          0.020502      0.297332      2.07     0.0385
c1 3               1     25.678601          0.855233      0.297280     86.38     <.0001
c1 4               1      0.420360          0.014000      0.297283      1.41     0.1574
c1 5               1      0.473986          0.015786      0.297265      1.59     0.1109
c1 6               1      0.394044          0.013124      0.297299      1.33     0.1851
c1 7               1      8.469793          0.282089      0.297345     28.48     <.0001
c1 8               0             0                 0             .         .          .
```

The magnitudes of the standardized estimates and the *t* statistics of the parameters of the effect c1 reveal that only levels 3 and 7 of this effect contribute appreciably to the model. This suggests that a more parsimonious model with similar or better predictive power might be obtained if parameters that correspond to the levels of c1 can enter or leave the model independently. You request this with the SPLIT option in the CLASS statement as shown in the following statements:

```
proc hpreg data=analysisData;
   partition roleVar=role(train='TRAIN' validate='VAL' test='TEST');
   class c1(split) c2 c3(order=data);
   model y =  c1|c2|c3|x1|x2|x3|x4|x5|x5|x6|x7|x8|x9|x10
             |x11|x12|x13|x14|x15|x16|x17|x18|x19|x20 @2 /stb;
   selection method = stepwise(select=sl sle=0.1 sls=0.15 choose=validate)
                      hierarchy=single details=steps;
run;
```

Output 12.1.8 shows the “Dimensions” table. You can see that because the columns in the design matrix that correspond to levels of c1 are treated as separate effects, the selection is now from 439 effects, even though the number of parameters is unchanged.

Output 12.1.8 Dimensions with c1 Split

```
                          The HPREG Procedure

                               Dimensions

                  Number of Effects                      278
                  Number of Effects after Splits         439
                  Number of Parameters                   661
```

Output 12.1.9 shows the selected effects. You can see that as anticipated the selected model now depends on only levels 3 and 7 of c1.

Output 12.1.9 Selected Effects with c1 Split

```
  Selected Effects: Intercept c1_3 c1_7 c3 c1_3*c3 x1 x5 x6 x7 x6*x7 x10 x20
```

Finally, the fit statistics for the selected model are shown Output 12.1.10.

Output 12.1.10 Fit Statistics for the Selected Model with c1 Split

```
                     Root MSE              5.04060
                     R-Square              0.74325
                     Adj R-Sq              0.74238
                     AIC                     21195
                     AICC                    21195
                     SBC                     16311
                     ASE (Train)          25.31622
                     ASE (Validate)       25.98055
                     ASE (Test)           25.76059
```

If you compare the ASE values for this model in Output 12.1.10 with the oracle values in Output 12.1.1 and the values for the model without splitting c1 in Output 12.1.5, you see that this more parsimonious model produces the best predictive performance on the test data of all the models considered in this example.

Example 12.2: Backward Selection in SMP and MPP Modes

This example shows how you can run PROC HPREG in SMP and MPP modes. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details about the execution modes of SAS High-Performance Analytics procedures. The focus of this example is to simply show how you can switch the modes of execution of PROC HPREG, rather than on any statistical features of the procedure. The following DATA step generates the data for this example. The response y depends on 20 of the 1,000 regressors.

```
data ex2Data;
   array x{1000};

   do i=1 to 10000;
      y=1;
      sign=1;

      do j=1 to 1000;
         x{j} = ranuni(1);
         if j<=20  then do;
           y = y + sign*j*x{j};
           sign=-sign;
         end;
      end;
      y = y + 5*rannor(1);
      output;
  end;
run;
```

The following statements use PROC HPREG to select a model by using BACKWARD selection:

```
proc hpreg data=ex2Data;
    model y = x: ;
    selection method = backward;
    performance details;
run;
```

Output 12.2.1 shows the "Performance Information" table. This shows that the HPREG procedure executes in client (SMP) mode using four threads because the client machine has four CPUs. You can force a certain number of threads on any machine involved in the computations with the NTHREADS option in the PERFORMANCE statement.

Output 12.2.1 Performance Information

```
                              The HPREG Procedure

                            Performance Information

                         Execution Mode        On client
                         Number of Threads     4
```

Output 12.2.2 shows the parameter estimates for the selected model. You can see that the default BACKWARD selection with selection and stopping based on the SBC criterion retains all 20 of the true effects but also keeps two extraneous effects.

Output 12.2.2 Parameter Estimates for the Selected Model

```
                              Parameter Estimates

                                             Standard
     Parameter    DF         Estimate           Error     t Value    Pr > |t|

     Intercept     1         1.506615        0.419811        3.59      0.0003
     x1            1         1.054402        0.176930        5.96      <.0001
     x2            1        -1.996080        0.176967      -11.28      <.0001
     x3            1         3.293331        0.177032       18.60      <.0001
     x4            1        -3.741273        0.176349      -21.22      <.0001
     x5            1         4.908310        0.176047       27.88      <.0001
     x6            1        -5.772356        0.176642      -32.68      <.0001
     x7            1         7.398822        0.175792       42.09      <.0001
     x8            1        -7.958471        0.176281      -45.15      <.0001
     x9            1         8.899407        0.177624       50.10      <.0001
     x10           1        -9.687667        0.176431      -54.91      <.0001
     x11           1        11.083373        0.175195       63.26      <.0001
     x12           1       -12.046504        0.176324      -68.32      <.0001
     x13           1        13.009052        0.176967       73.51      <.0001
     x14           1       -14.456393        0.175968      -82.15      <.0001
     x15           1        14.928731        0.174868       85.37      <.0001
     x16           1       -15.762907        0.177651      -88.73      <.0001
     x17           1        16.842889        0.177037       95.14      <.0001
     x18           1       -18.468844        0.176502     -104.64      <.0001
     x19           1        18.810193        0.176616      106.50      <.0001
     x20           1       -20.212291        0.176325     -114.63      <.0001
     x87           1        -0.542384        0.176293       -3.08      0.0021
     x362          1        -0.560999        0.176594       -3.18      0.0015
```

Output 12.2.3 shows timing information for the PROC HPREG run. This table is produced when you specify the DETAILS option in the PERFORMANCE statement. You can see that, in this case, the majority of time is spent forming the crossproducts matrix for the model that contains all the regressors.

Output 12.2.3 Timing

```
                          Procedure Task Timing

                                                   Time
        Task                                      (sec.)

        Reading and Levelizing Data                 1.25     6.78%
        Loading Design Matrix                       0.78     4.24%
        Computing Moments                           0.09     0.51%
        Computing Cross Products Matrix            12.67     68.8%
        Performing Model Selection                  3.62     19.7%
```

You can switch to running PROC HPREG in MPP mode by specifying valid values for the NODES=, INSTALL=, and HOST= options in the PERFORMANCE statement. An alternative to specifying the INSTALL= and HOST= options in the PERFORMANCE statement is to set appropriate values for the GRIDHOST and GRIDINSTALLLOC environment variables by using OPTIONS SET commands. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details about setting these options or environment variables.

The following statements provide an example. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to macro variables with appropriate values.

```
proc hpreg data=ex2Data;
    model y = x: ;
    selection method = backward;
    performance details nodes = 10
                host="&GRIDHOST" install="&GRIDINSTALLLOC";
run;
```

The execution mode in the "Performance Information" table shown in Output 12.2.4 indicates that the calculations were performed in a distributed environment that uses 10 nodes, each of which uses eight threads.

Output 12.2.4 Performance Information in MPP Mode

```
                         Performance Information

     Host Node                     << your grid host >>
     Install Location              << your grid install location >>
     Execution Mode                Distributed
     Number of Compute Nodes       10
     Number of Threads per Node    8
```

Another indication of distributed (MPP) execution is the following message issued by all High-Performance Analytics procedures in the SAS Log:

```
NOTE: The HPREG procedure is executing in the distributed
      computing environment with 10 worker nodes.
```

Output 12.2.5 shows timing information for this distributed run of the HPREG procedure. In contrast to the SMP case (where forming the crossproducts matrix dominated the time spent), the majority of time in the MPP run is spent performing the model selection.

Output 12.2.5 Timing

```
                         Procedure Task Timing

                                                     Time
         Task                                       (sec.)

         Distributing Data                           1.22     27.1%
         Reading and Levelizing Data                 0.03     0.65%
         Loading Design Matrix                       0.01     0.29%
         Computing Moments                           0.02     0.38%
         Computing Cross Products Matrix             0.86     19.2%
         Performing Model Selection                  2.36     52.4%
```

Example 12.3: Forward-Swap Selection

This example highlights the use of the forward-swap selection method, which is a generalization of the maximum R-square improvement (MAXR) method that is available in the REG procedure in SAS/STAT software. This example also demonstrates the use of the INCLUDE and START options.

The following DATA step produces the simulated data in which the response y depends on six main effects and three 2-way interactions from a set of 20 regressors.

```
data ex3Data;
   array x{20};
   do i=1 to 10000;
      do j=1 to 20;
         x{j} = ranuni(1);
      end;
      y = 3*x1 + 7*x2 -5*x3 + 5*x1*x3 +
          4*x2*x13 + x7 + x11 -x13  + x1*x4 + rannor(1);
      output;
   end;
run;
```

Suppose you want to find the best model of each size in a range of sizes for predicting the response y. You can use the forward-swap selection method to produce good models of each size without the computational expense of examining all possible models of each size. In this example, the criterion used to evaluate the models of each size is the model R square. With this criterion, the forward-swap method coincides with the MAXR method that is available in the REG procedure in SAS/STAT software. The model of a given size for which no pairwise swap of an effect in the model with any candidate effect improves the R-square value is deemed to be the best model of that size.

Suppose that you have prior knowledge that the regressors x1, x2, and x3 are needed in modeling the response y. Suppose that you also believe that some of the two-way interactions of these variables are likely to be important in predicting y and that some other two-way interactions might also be needed. You can use this prior information by specifying the selection process shown in the following statements:

```
proc hpreg data=ex3Data;
   model y = x1|x2|x3|x4|x5|x6|x7|x8|x9|x10|X11|
             x12|x13|x14|x5|x16|x7|x18|x19|x20@2 /
                include=(x1 x2 x3) start=(x1*x2 x1*x3 x2*x3);
   selection method=forwardswap(select=rsquare maxef=15 choose=sbc) details=all;
run;
```

The MODEL statement specifies that all main effects and two-way interactions are candidates for selection. The INCLUDE= option specifies that the effects x1, x2, and x3 must appear in all models that are examined. The START= option specifies that all the two-way interactions of these variables should be used in the initial model that is considered but that these interactions are eligible for removal during the forward-swap selection.

The “Selection Summary” table is shown in Output 12.3.1.

Output 12.3.1 Selection Summary

```
                              The HPREG Procedure

                               Selection Summary

             Effect     Effect        Number                        Model
     Step    Entered    Removed    Effects In           SBC      R-Square

        0    Intercept                      1
             x1                             2
             x2                             3
             x1*x2                          4
             x3                             5
             x1*x3                          6
             x2*x3                          7     3307.6836        0.8837
--------------------------------------------------------------------------
        1    x2*x13                         8     1892.8403        0.8992
        2    x7*x11     x1*x2               8      618.9298        0.9112
        3    x1*x4      x2*x3               8      405.3751        0.9131
        4    x13                            9      213.6140        0.9148
        5    x7                            10      180.4457        0.9152
        6    x11        x7*x11             10        1.4039*       0.9167
        7    x10*x11                       11        2.3393        0.9168
        8    x3*x7                         12        4.5000        0.9168
        9    x6*x7                         13       10.0589        0.9169
       10    x3*x6                         14       13.1113        0.9169
       11    x5*x20                        15       19.4612        0.9169
       12    x13*x20    x3*x6              15       18.3678        0.9169
       13    x5*x5      x6*x7              15       12.1398        0.9170*

                         * Optimal Value of Criterion
```

You see that starting from the model with an intercept and the effects specified in the INCLUDE= and START= options at step 0, the forward-swap selection method adds the effect x2*x13 at step one, because this yields the maximumum improvement in R square that can be obtained by adding a single effect. The forward-swap selection method now evaluates whether any effect swap yields a better eight-effect model (one with a higher R-square value). Because you specified the DETAILS=ALL option in the SELECTION statement, at each step where a swap is made you obtain a “Candidates” table that shows the R-square values for the evaluated swaps. Output 12.3.2 shows the “Candidates” for step 2. By default, only the best 10 swaps are displayed.

Output 12.3.2 Swap Candidates at Step 2

```
                              Best 10 Candidates

                              Effect      Effect
                      Rank    Dropped     Added       R-Square

                         1    x1*x2       x7*x11        0.9112
                         2    x2*x3       x7*x11        0.9112
                         3    x1*x2       x7            0.9065
                         4    x2*x3       x7            0.9065
                         5    x1*x2       x7*x7         0.9060
                         6    x2*x3       x7*x7         0.9060
                         7    x1*x2       x4*x7         0.9060
                         8    x2*x3       x4*x7         0.9060
                         9    x1*x2       x11           0.9058
                        10    x2*x3       x11           0.9058
```

You see that the best swap adds x7*x11 and drops x1*x2. This yields an eight-effect model whose R-square value (0.9112) is larger than the R-square value (0.8992) of the eight-effect model at step 1. Hence this swap is made at step 2. At step 3, an even better eight-effect model than the model at step 2 is obtained by dropping x2*x3 and adding x1*x4. No additional swap improves the R-square value, and so the model at step 3 is deemed to be the best eight-effect model. Although this is the best eight-effect model that can be found by this method given the starting model, it is not guaranteed that this model that has the highest R-square value among all possible models that consist of seven effects and an intercept.

Because the DETAILS=ALL option is specified in the SELECTION statement, details for the model at each step of the selection process are displayed. Output 12.3.3 provides details of the model at step 3.

Output 12.3.3 Model Details at Step 3

```
                             Analysis of Variance

                                    Sum of            Mean
Source                   DF        Squares          Square    F Value    Pr > F

Model                     7         108630           15519    15000.3    <.0001
Error                  9992          10337         1.03455
Corrected Total        9999         118967

                           Root MSE         1.01713
                           R-Square         0.91311
                           Adj R-Sq         0.91305
                           AIC                10350
                           AICC               10350
                           SBC            405.37511
                           ASE              1.03373
```

Output 12.3.3 *continued*

```
                         Parameter Estimates

                                          Standard
Parameter      DF        Estimate           Error     t Value    Pr > |t|

Intercept       1        0.012095        0.045712        0.26      0.7913
x1              1        3.087078        0.076390       40.41      <.0001
x2              1        7.775180        0.046815      166.08      <.0001
x3              1       -4.957140        0.070995      -69.82      <.0001
x1*x3           1        4.910115        0.122503       40.08      <.0001
x1*x4           1        0.890436        0.060523       14.71      <.0001
x7*x11          1        1.708469        0.045939       37.19      <.0001
x2*x13          1        2.584078        0.061506       42.01      <.0001
```

The forward-swap method continues to find the best nine-effect model, best 10-effect model, and so on until it obtains the best 15-effect model. At this point the selection terminates because you specified the MAXEF=15 option in the SELECTION statement. The R-square value increases at each step of the selection process. However, because you specified the CHOOSE=SBC criterion in the SELECTION statement, the final model selected is the model at step 6.

References

Akaike, H. (1969), "Fitting Autoregressive Models for Prediction," *Annals of the Institute of Statistical Mathematics*, 21, 243–247.

Burnham, K. P. and Anderson, D. R. (2002), *Model Selection and Multimodel Inference*, Second Edition, New York: Springer-Verlag.

Collier Books (1987), *The 1987 Baseball Encyclopedia Update*, Macmillan.

Darlington, R. B. (1968), "Multiple Regression in Psychological Research and Practice," *Psychological Bulletin*, 69, 161–182.

Draper, N. R., Guttman, I., and Kanemasu, H. (1971), "The Distribution of Certain Regression Statistics," *Biometrika*, 58, 295–298.

Harrell, F. E. (2001), *Regression Modeling Strategies*, New York: Springer-Verlag.

Hastie, T., Tibshirani, R., and Friedman, J. (2001), *The Elements of Statistical Learning*, New York: Springer-Verlag.

Hocking, R. R. (1976), "The Analysis and Selection of Variables in a Linear Regression," *Biometrics*, 32, 1–50.

Hurvich, C. M. and Tsai, C.-L. (1989), "Regression and Time Series Model Selection in Small Samples," *Biometrika*, 76, 297–307.

Judge, G. G., Griffiths, W. E., Hill, R. C., Lutkepohl, H., and Lee, T. C. (1985), *The Theory and Practice of Econometrics*, Second Edition, New York: John Wiley & Sons.

Mallows, C. L. (1973), "Some Comments on C_p," *Technometrics*, 15, 661–675.

Schwarz, G. (1978), "Estimating the Dimension of a Model," *Annals of Statistics*, 6, 461–464.

Time Inc. (1987), "What They Make," *Sports Illustrated*, 54–81.

Chapter 13

The HPSAMPLE Procedure

Contents

Overview: HPSAMPLE Procedure

The HPSAMPLE procedure is a high-performance procedure that performs either simple random sampling or stratified sampling. The HPSAMPLE procedure creates the following:

- one output data set, which contains the sample data set
- one performance table, which contains performance information
- one frequency table, which contains the frequency information for the population and sample

The HPSAMPLE procedure is high-performance in that it takes advantage of distributed computing environments (currently using only a single thread) when the input data are stored on the SAS appliance.

Getting Started: HPSAMPLE Procedure

The following example shows a 10% stratified sampling with the target variable BAD used by the HPSAMPLE procedure as a stratum:

```
proc hpsample data=Sampsio.Hmeq  out=Smp samppct=10 seed=1234 partition;
    var loan derog mortdue value yoj delinq
        clage ninq clno debtinc;
    class bad reason job;
    target bad;
run;
proc print data=Smp;run;
```

The input data set Sampsio.Hmeq includes information about 5,960 fictitious mortgages. Each observation represents an applicant for a home equity loan, and all applicants have an existing mortgage. The SAMPPCT=10 option specifies that 10% of the input data be sampled. The SEED option specifies the random seed used in the sampling process is 1234. The PARTITION option specifies that the output data set, Smp, include an indicator that shows whether each observation is selected to the sample (1) or not (0). The VAR statement specifies ten numeric input variables, and the CLASS statement specifies three classification input variables. All these variables are included in the output sample. The binary TARGET variable BAD indicates whether an applicant eventually defaulted or was ever seriously delinquent. The displayed output contains a performance table (Figure 13.1) which shows the performance environment information, and a frequency table (Figure 13.2), which shows the frequency of observations in each level of BAD.

Figure 13.1 Performance Information

```
                    Performance Information

                 Execution Mode       On client
                 Number of Threads    1
```

Figure 13.2 Frequency Table

```
              Stratified Sampling Frequency Table

       Target Level      Number of Obs    Number of Sample

       0                          4771                 478
       1                          1189                 118
```

Syntax: HPSAMPLE Procedure

The following statements are available in the HPSAMPLE procedure:

PROC HPSAMPLE < *options* > **;**
  VAR *variable* < *variable ... variable* > **;**
  CLASS *variable* < *variable ... variable* > **;**
  TARGET *variable* < *variable ... variable* > **;**
  PERFORMANCE *performance-options* **;**

Either a VAR or a CLASS statement is required for simple random sampling; both the CLASS and TARGET statements are required for stratified sampling.

PROC HPSAMPLE Statement

PROC HPSAMPLE < *options* > **;**

The PROC HPSAMPLE statement invokes the procedure.

You can specify the following *options*:

DATA=< *libref.* **>***table*
names the table (SAS data set or database table) that you want to sample from. The default is the most recently opened or created data set. If the data are already distributed, the procedure reads the data alongside the distributed database. See the section "SMP and MPP Modes" on page 10 for the various execution modes and the section "Alongside-the-Database Execution" on page 15 for the alongside-the-database model.

OUT=< *libref.* **>***SAS-data-set*
names the SAS data set that you want to output the sample to. If you run alongside database, you need to specify a data set with the same database *libref* as the input data and make sure it does not already exist in the database. This option is required.

SAMPPCT=*sample-percentage*
names sample percentage to be used by PROC HPSAMPLE. The value of *sample-percentage* should be a positive number less than 100. For example, SAMPPCT=50.5 specifies that you want to sample 50.5 percent of data.

SAMPOBS=*number*
names the minimal number of observations you want to sample from the input data. The value of *number* must be a positive integer. If it exceeds the total number of observations in the input data, the output sample has the same number of observations as the input data set.

NOTE: You must specify either the SAMPPCT or the SAMPOBS option. If both are specified, only the SAMPPCT option is honored.

SEED=*random-seed*

specifies the seed for the random number generator. If *random-seed* is not specified or it is specified as a negative number, the seed is set to be the default 12345. The SEED option enables you to reproduce the same sample output.

PARTITION

produces an output data set with the same number of rows as the input data set but with an additional partition indicator (_PARTIND_), which indicates whether an observation is selected to the sample (1) or not (0).

NONORM

distinguishes target values that share the same normalized value when you do stratified sampling. For example, if a target has three distinct values, "A", "B", and "b", and you want to treat "B" and "b" as different levels, you need to use NONORM. By default, "B" and "b" are treated as the same level. PROC HPSAMPLE normalizes a value as follows:

1. Leading blanks are removed.
2. The value is truncated to 32 characters.
3. Letters are changed from lowercase to uppercase.

CLASS Statement

CLASS *variable* < *variable* ... *variable* > **;**

The CLASS statement specifies one or more classification variables to be included in the sample. At least one *variable* is required. A *variable* can be character or numeric. The CLASS and VAR statements are mutually exclusive.

NOTE: Each *variable* in the TARGET statement must be specified in the CLASS statement.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS High-Performance Analytics procedure.

With the PERFORMANCE statement you can also control whether a SAS High-Performance Analytics procedure executes in symmetric multiprocessor (SMP) or massively parallel processor (MPP) mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

NOTE: PROC HPSAMPLE does not support multithreading in this release.

TARGET Statement

TARGET *variable* < *variable* ... *variable* > **;**

The TARGET statement specifies one or two classification variables to be used for stratificaiton. Each *variable* must be specified in the CLASS statement. Currently up to two target variables are supported. The maximum number of levels (that is, distinct values) in any target variable is 256.

VAR Statement

VAR *variable* < *variable* ... *variable* > **;**

The VAR statement specifies one or more numeric variables to be included in the sample. At least one *variable* is required; all *variables* must be numeric. You can use this statement to include only the variables of interests in your sample. The CLASS and VAR statements are mutually exclusive.

Details: HPSAMPLE Procedure

Class Level

For classification variables, a *level* is an observed value that is distinct after formatting, removal of beginning and ending white space, and capitalization. For example, the values **`MyLevel`** and **`MYLEVEL`** are treated as a single level in the data set. Class variables can be numeric, and the same levelization rules apply. For example, `3.000002` and `3.0000001` are treated as the same level if they are formatted using **`BEST3`**.

Displayed Output

The following sections describe the output that PROC HPSAMPLE produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Performance Information

The “Performance Information” table is produced by default. It displays information about the grid host for distributed execution and information about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The numbers of compute nodes and threads are also displayed, depending on the environment.

Timing Table

The “Timing Table” lists the timing information for various computational stages of the procedure.

Frequency Information Table

For simple random sampling, the “Frequency Information Table” lists the number of observations in the input data set and in the sample output data set.

For stratified sampling, the “Frequency Information Table” table lists the respective frequency in each stratum for the input data and the sample. If one target variable is specified, each level of the target variable represents a stratum; if two target variables are specified, a combination of the levels of two target variables represents a stratum.

ODS Table Names

Each table created by the HPSAMPLE procedure has a name associated with it, and you must use this name to refer to the table when you use ODS statements. These names are listed in Table 13.1.

Table 13.1 ODS Tables Produced by PROC HPSAMPLE

Table Name	**Description**	**Required Statement / Option**
PerformanceInfo	Information about the high-performance computing environment	Default output
Timing	Timing information for various computational stages of the procedure	DETAILS (PERFORMANCE statement)
FreqTable	Frequency table of input data set and output sample (when target variables are used, this table contains stratification information for population and sample)	Default output

Examples: HPSAMPLE Procedure

Example 13.1: Running PROC HPSAMPLE on the Client

This example demonstrates how to use PROC HPSAMPLE to perform a simple random sampling on the Sampsio.Hmeq data set, which resides on the client.

When the input data set resides on the client and no PERFORMANCE statement is specified, as in the following statements, the client performs all computations:

```
/*sampsio is a libname for a data source on the client.*/
proc hpsample data=sampsio.hmeq out=out1 sampobs=20 seed=13579;
    class job reason;
    var loan value delinq derog;
run;
proc print data=out1;
run;
```

Output 13.1.1 shows the performance environment information.

Output 13.1.1 Performance Information

```
                    Performance Information

                 Execution Mode       On client
                 Number of Threads    1
```

Output 13.1.2 shows the number of observations in the data set sampsio.hmeq and the number of samples.

Output 13.1.2 Frequency Table

```
                     Simple Random Sampling
                        Frequency Table

              Number of Obs    Number of Sample

                       5960                  22
```

Output 13.1.3 shows the sample data.

Output 13.1.3 Sample Data

```
Obs    JOB        REASON      LOAN     VALUE    DELINQ    DEROG

  1    Other      HomeImp     4900     65774       2        1
  2    Other      HomeImp     5700     82923       0        0
  3    Other      HomeImp     7000    124827       .        .
  4               DebtCon     8300     75081       0        0
  5    Office     DebtCon    10000    125500       0        0
  6    Other                 11100     61406       0        0
  7    Other      HomeImp    11500     64037       0        0
  8    ProfExe    HomeImp    11900    105454       0        0
  9    Mgr        HomeImp    15000    122400       2        0
 10    Other      HomeImp    15000    107207       0        0
 11    ProfExe    HomeImp    15600    106824       0        0
 12    Office     DebtCon    17000     69000       0        0
 13    ProfExe    DebtCon    17300     49100       0        0
 14    Other      DebtCon    18000     60000       0        2
 15    Office     DebtCon    21100     98000       2        0
 16    Other      HomeImp    21400    103427       0        0
 17    Mgr        HomeImp    22400    121601       0        0
 18    Other      HomeImp    25000    202500       0        0
 19    Other      DebtCon    25500     43031       .        .
 20    Mgr        DebtCon    27500    149877       0        0
 21    ProfExe    DebtCon    36500    195729       0        0
 22    Self       DebtCon    70300    294169       0        0
```

Example 13.2: Running with Client Data on the SAS appliance

This example uses the same data set as is used in Example 13.1.

When the input data set resides on the client and a PERFORMANCE statement with a NODES= option is specified, as in the following statements, PROC HPSAMPLE copies the data set to the SAS appliance, where the sampling is performed:

```
/*Perform the computation on the SAS appliance using 2 nodes*/
option set=GRIDHOST="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";
proc hpsample data=sampsio.hmeq out=out2 samppct=10 seed=13579 partition;
    var loan value delinq derog;
    class job reason;
    target job;
    performance nodes=2;
run;
proc print data=out2(obs=15);
run;
```

Output 13.2.1 shows the performance environment information.

Output 13.2.1 Performance Information

```
                        Performance Information

              Host Node                      rdgrd0001.unx.sas.com
              Execution Mode                 Distributed
              Number of Compute Nodes        2
              Number of Threads per Node     1
```

Output 13.2.2 shows the frequency information for each level of target variable JOB in the data set Sampsio.Hmeq and in the sample.

Output 13.2.2 Frequency Table

```
                  Stratified Sampling Frequency Table

    Target Level                         Number of Obs    Number of Sample

                                                   279                  28
    Mgr                                            767                  77
    Office                                         948                  95
    Other                                         2388                 239
    ProfExe                                       1276                 127
    Sales                                          109                  11
    Self                                           193                  19
```

Output 13.2.3 shows first 15 output sample observations which contain an indicator "_PARTIND_" to indicate whether the observation is selected to the sample (1) or not (0).

Output 13.2.3 Sample Output with Partition Indicator

```
                                                                      _
                                                                    Part
     Obs    JOB        REASON     LOAN     VALUE    DELINQ    DEROG    Ind_

       1    Other      HomeImp    1100     39025       0        0        0
       2    Other      HomeImp    1300     68400       2        0        0
       3    Other      HomeImp    1500     16700       0        0        0
       4                          1500         .       .        .        0
       5    Office     HomeImp    1700    112000       0        0        0
       6    Other      HomeImp    1700     40320       0        0        0
       7    Other      HomeImp    1800     57037       2        3        0
       8    Other      HomeImp    1800     43034       0        0        0
       9    Other      HomeImp    2000     46740       2        0        0
      10    Sales      HomeImp    2000     62250       0        0        0
      11                          2000         .       .        .        0
      12    Office     HomeImp    2000     29800       1        0        0
      13    Other      HomeImp    2000     55000       0        0        0
      14    Mgr                   2000     87400       0        0        0
      15    Other      HomeImp    2100     83850       1        0        0
```

Example 13.3: Running with Data on the SAS appliance

This example uses the same data set as is used in Example 13.1.

When the input data set resides on the SAS appliance, the SAS appliance performs all samplings, writes out a sample on the SAS appliance, and reports the frequency results back to the client. In the following statements, the input data resides in the MyLib library (which is a distributed data source), and the output data set is a distributed data set in the MyLib library. You can use PROC DATASETS to delete the output table if it already exists on the SAS appliance. The **ods output FreqTable=Freqtab;** statement saves the frequency table to a SAS data set called Freqtab on the client.

```
/*MyLib is a libname for a distributed data source
  In this case, the computation is automatically done
  on the SAS appliance.*/
option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";
libname MyLib &LIBTYPE
        server  ="&GRIDDATASERVER"
        user    =&USER
        password=&PASSWORD
        database=&DATABASE;

proc datasets lib=Mylib nolist; delete out1;
run;
proc hpsample data=MyLib.hmeq out=MyLib.out1 samppct=10 seed=1;
    var loan value delinq derog;
    class job reason;
    target  job reason;
    ods output FreqTable=Freqtab;
run;
```

Output 13.3.1 shows the performance environment information.

Output 13.3.1 Performance Information

```
                         Performance Information

         Host Node                     green1.unx.sas.com
         Execution Mode                Distributed, alongside Greenplum
         Number of Compute Nodes       32
         Number of Threads per Node    1
```

Output 13.3.2 shows the number of observations in each stratum (constructed by a combination of the levels of each of the two targets) in the data set MyLib.Hmeq and in the sample.

Output 13.3.2 Frequency Table

```
                      Stratified Sampling1 Frequency Table

      Target1 Level                          Target2 Level


                                             DebtCon
                                             HomeImp
      Mgr
      Mgr                                    DebtCon
      Mgr                                    HomeImp
      Office
      Office                                 DebtCon
      Office                                 HomeImp
      Other
      Other                                  DebtCon
      Other                                  HomeImp
      ProfExe
      ProfExe                                DebtCon
      ProfExe                                HomeImp
      Sales                                  DebtCon
      Sales                                  HomeImp
      Self
      Self                                   DebtCon
      Self                                   HomeImp

                             Stratified Sampling1
                                Frequency Table

                       Number of Obs    Number of Sample

                                 107                   6
                                 115                  10
                                  57                   1
                                  21                   0
                                 572                  58
                                 174                  20
                                  27                   0
                                 620                  64
                                 301                  33
                                  68                   1
                                1604                 164
                                 716                  73
                                  24                   0
                                 847                  87
                                 405                  43
                                  97                   5
                                  12                   0
                                   5                   0
                                  73                   2
                                 115                  11
```

Chapter 14

The HPSEVERITY Procedure

Contents

Overview: HPSEVERITY Procedure

The HPSEVERITY procedure is a high-performance version of the SEVERITY procedure in SAS/ETS software. PROC HPSEVERITY estimates parameters of any arbitrary continuous probability distribution that is used to model the magnitude (severity) of a continuous-valued event of interest. Some examples of such events are loss amounts paid by an insurance company and demand of a product as depicted by its sales. PROC HPSEVERITY is especially useful when the severity of an event does not follow typical distributions (such as the normal distribution) that are often assumed by standard statistical methods.

PROC HPSEVERITY provides a default set of probability distribution models that includes the Burr, exponential, gamma, generalized Pareto, inverse Gaussian (Wald), lognormal, Pareto, Tweedie, and Weibull distributions. In the simplest form, you can estimate the parameters of any of these distributions by using a list of severity values that are recorded in a SAS data set. PROC HPSEVERITY computes the estimates of the model parameters, their standard errors, and their covariance structure by using the maximum likelihood method.

PROC HPSEVERITY can fit multiple distributions at the same time and choose the best distribution according to a specified selection criterion. Four different statistics of fit can be used as selection criteria. They are log likelihood, Akaike's information criterion (AIC), corrected Akaike's information criterion (AICC), and Schwarz Bayesian information criterion (BIC).

You can request the procedure to output the status of the estimation process, the parameter estimates and their standard errors, the estimated covariance structure of the parameters, and the statistics of fit.

The following key features make PROC HPSEVERITY and PROC SEVERITY unique among SAS procedures that can estimate continuous probability distributions:

- Both procedures enable you to fit a distribution model when the severity values are truncated or censored or both. You can specify any combination of the following types of censoring and truncation effects: left-censoring, right-censoring, left-truncation, or right-truncation. This is especially useful in applications with an insurance-type model where a severity (loss) is reported and recorded only if it is greater than the deductible amount (left-truncation) and where a severity value greater than or equal to the policy limit is recorded at the limit (right-censoring). Another useful application is that of interval-censored data, where you know both the lower limit (right-censoring) and upper limit (left-censoring) on the severity, but you do not know the exact value.

- Both procedures enable you to define any arbitrary continuous parametric distribution model and to estimate its parameters. You just need to define the key components of the distribution, such as its probability density function (PDF) and cumulative distribution function (CDF), as a set of functions and subroutines written with the FCMP procedure, which is part of Base SAS software. As long as the functions and subroutines follow certain rules, PROC HPSEVERITY can fit the distribution model defined by them.

- Both procedures can model the effect of exogenous or regressor variables on a probability distribution, as long as the distribution has a scale parameter. A linear combination of the regressor variables is assumed to affect the scale parameter via an exponential link function.

  If a distribution does not have a scale parameter, then either it needs to have another parameter that can be derived from a scale parameter by using a supported transformation or it needs to be reparameterized to have a scale parameter. If neither of these is possible, then regression effects cannot be modeled.

- PROC HPSEVERITY employs multithreading and distributed processing to significantly reduce the time it takes to fit the distribution model. The PERFORMANCE statement enables you to control the threading and distributed processing parameters.

These features and the core functionality are described in detail in the following sections.

Getting Started: HPSEVERITY Procedure

This section outlines the use of the HPSEVERITY procedure to fit continuous probability distribution models. Three examples illustrate different features of the procedure.

A Simple Example of Fitting Predefined Distributions

The simplest way to use PROC HPSEVERITY is to fit all the predefined distributions to a set of values and let the procedure identify the best fitting distribution.

Consider a lognormal distribution, whose probability density function (PDF) f and cumulative distribution function (CDF) F are as follows, respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2} \quad \text{and} \quad F(x;\mu,\sigma) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$$

The following DATA step statements simulate a sample from a lognormal distribution with population parameters $\mu = 1.5$ and $\sigma = 0.25$, and store the sample in the variable Y of a data set Work.Test_sev1:

```
/*------------- Simple Lognormal Example -------------*/
data test_sev1(keep=y label='Simple Lognormal Sample');
   call streaminit(45678);
   label y='Response Variable';
   Mu = 1.5;
   Sigma = 0.25;
   do n = 1 to 100;
      y = exp(Mu) * rand('LOGNORMAL')**Sigma;
      output;
   end;
run;
```

The following statements fit all the predefined distribution models to the values of Y and identify the best distribution according to the corrected Akaike's information criterion (AICC):

```
proc hpseverity data=test_sev1 crit=aicc;
   loss y;
   dist _predefined_;
run;
```

The PROC HPSEVERITY statement specifies the input data set along with the model selection criterion, the LOSS statement specifies the variable to be modeled, and the DIST statement with the _PREDEFINED_ keyword specifies that all the predefined distribution models be fitted.

Some of the default output displayed by this step is shown in Figure 14.1 through Figure 14.3. First, information about the input data set is displayed followed by the "Model Selection Table", as shown in Figure 14.1. The model selection table displays the convergence status, the value of the selection criterion, and the selection status for each of the candidate models. The Converged column indicates whether the estimation process for a given distribution model has converged, might have converged, or failed. The Selected column indicates whether a given distribution has the best fit for the data according to the selection criterion. For this example, the lognormal distribution model is selected, because it has the lowest value for the selection criterion.

Figure 14.1 Data Set Information and Model Selection Table

```
                         The HPSEVERITY Procedure

                              Input Data Set

                     Name                WORK.TEST_SEV1
                     Label    Simple Lognormal Sample

                           Model Selection Table

                                            Corrected
                                             Akaike's
                                          Information
          Distribution    Converged         Criterion    Selected

          Burr            Yes               322.50845    No
          Exp             Yes               508.12287    No
          Gamma           Yes               320.50264    No
          Igauss          Yes               319.61652    No
          Logn            Yes               319.56579    Yes
          Pareto          Yes               510.28172    No
          Gpd             Yes               510.20576    No
          Weibull         Yes               334.82373    No
```

Next, the estimation information for each of the candidate models is displayed. The information for the lognormal model, which is the best fitting model, is shown in Figure 14.2. The first table displays a summary of the distribution. The second table displays the convergence status. This is followed by a summary of the optimization process which indicates the technique used, the number of iterations, the number of times the objective function was evaluated, and the log likelihood attained at the end of the optimization. Since the model with lognormal distribution has converged, PROC HPSEVERITY displays its statistics of fit and parameter estimates. The estimates of *Mu*=1.49605 and *Sigma*=0.26243 are quite close to the population parameters of *Mu*=1.5 and *Sigma*=0.25 from which the sample was generated. The p-value for each estimate indicates the rejection of the null hypothesis that the estimate is 0, implying that both the estimates are significantly different from 0.

Figure 14.2 Estimation Details for the Lognormal Model

```
                        The HPSEVERITY Procedure

                        Distribution Information

           Name                                                   Logn
           Description                           Lognormal Distribution
           Number of Distribution Parameters                         2

                  Convergence Status for Logn Distribution

               Convergence criterion (GCONV=1E-8) satisfied.

                  Optimization Summary for Logn Distribution

               Optimization Technique              Trust Region
               Number of Iterations                           2
               Number of Function Evaluations                 8
               Log Likelihood                        -157.72104

                     Fit Statistics for Logn Distribution

             -2 Log Likelihood                             315.44208
             Akaike's Information Criterion                319.44208
             Corrected Akaike's Information Criterion      319.56579
             Schwarz's Bayesian Information Criterion      324.65242

                  Parameter Estimates for Logn Distribution

                                         Standard                  Approx
           Parameter       Estimate         Error     t Value    Pr > |t|

           Mu               1.49605       0.02651       56.43      <.0001
           Sigma            0.26243       0.01874       14.00      <.0001
```

The parameter estimates of the Burr distribution are shown in Figure 14.3. These estimates are used in the next example.

Figure 14.3 Parameter Estimates for the Burr Model

```
                  Parameter Estimates for Burr Distribution

                                         Standard                  Approx
           Parameter       Estimate         Error     t Value    Pr > |t|

           Theta            4.62348       0.46181       10.01      <.0001
           Alpha            1.15706       0.47493        2.44      0.0167
           Gamma            6.41227       0.99039        6.47      <.0001
```

An Example with Left-Truncation and Right-Censoring

PROC HPSEVERITY enables you to specify that the response variable values are left-truncated or right-censored. The following DATA step expands the data set of the previous example to simulate a scenario that is typically encountered by an automobile insurance company. The values of the variable Y represent the loss values on claims that are reported to an auto insurance company. The variable THRESHOLD records the deductible on the insurance policy. If the actual value of Y is less than or equal to the deductible, then it is unobservable and does not get recorded. In other words, THRESHOLD specifies the left-truncation of Y. LIMIT records the policy limit. If the value of Y is equal to or greater than the recorded value, then the observation is right-censored.

```
/*----- Lognormal Model with left-truncation and censoring -----*/
data test_sev2(keep=y threshold limit
        label='A Lognormal Sample With Censoring and Truncation');
   set test_sev1;
   label y='Censored & Truncated Response';
   if _n_ = 1 then call streaminit(45679);

   /* make about 20% of the observations left-truncated */
   if (rand('UNIFORM') < 0.2) then
      threshold = y * (1 - rand('UNIFORM'));
   else
      threshold = .;
   /* make about 15% of the observations right-censored */
   iscens = (rand('UNIFORM') < 0.15);
   if (iscens) then
      limit = y;
   else
      limit = .;
run;
```

The following statements use the AICC criterion to analyze which of the four predefined distributions (lognormal, Burr, gamma, and Weibull) has the best fit for the data:

```
proc hpseverity data=test_sev2 crit=aicc print=all ;
   loss y / lt=threshold rc=limit;

   dist logn burr gamma weibull;
   performance nthreads=2;
run;
```

The LOSS statement specifies the left-truncation and right-censoring variables. Each candidate distribution needs to be specified by using a separate DIST statement. The PRINT= option in the PROC HPSEVERITY statement requests that all the displayed output be prepared. The NTHREADS option in the PERFORMANCE statement specifies that two threads of computation be used. The option is shown here just for illustration. You should use it only when you want to restrict the procedure to use a different number of threads than the value of the CPUCOUNT= system option, which typically defaults to the number of physical CPU cores available on your machine, thereby allowing the procedure to fully utilize the computational power of your machine.

Some of the key results prepared by PROC HPSEVERITY are shown in Figure 14.4 through Figure 14.7. In addition to the estimates of the range, mean, and standard deviation of Y, the "Descriptive Statistics for Variable y" table shown in Figure 14.4 also indicates the number of observations that are left-truncated or right-censored. The "Model Selection Table" in Figure 14.4 shows that models with all the candidate distributions have converged and that the Logn (lognormal) model has the best fit for the data according to the AICC criterion.

Figure 14.4 Summary Results for the Truncated and Censored Data

```
                         The HPSEVERITY Procedure

                             Input Data Set

            Name                                        WORK.TEST_SEV2
            Label    A Lognormal Sample With Censoring and Truncation

                   Descriptive Statistics for Variable y

             Number of Observations                                100
             Number of Observations Used for Estimation            100
             Minimum                                           2.30264
             Maximum                                           8.34116
             Mean                                              4.62007
             Standard Deviation                                1.23627
             Number of Left Truncated Observations                  23
             Number of Right Censored Observations                  14

                          Model Selection Table

                                          Corrected
                                           Akaike's
                                         Information
               Distribution    Converged      Criterion    Selected

               Logn            Yes            298.92672    Yes
               Burr            Yes            302.66229    No
               Gamma           Yes            299.45293    No
               Weibull         Yes            309.26779    No
```

PROC HPSEVERITY also prepares a table that shows all the fit statistics for all the candidate models. It is useful to see which model would be the best fit according to each of the criteria. The "All Fit Statistics Table" prepared for this example is shown in Figure 14.5. It indicates that the lognormal model is chosen by all the criteria.

Figure 14.5 Comparing All Statistics of Fit for the Truncated and Censored Data

```
                         All Fit Statistics Table

                    -2 Log
  Distribution    Likelihood        AIC          AICC          BIC

  Logn            294.80301*     298.80301*    298.92672*    304.01335*
  Burr            296.41229      302.41229     302.66229     310.22780
  Gamma           295.32921      299.32921     299.45293     304.53955
  Weibull         305.14408      309.14408     309.26779     314.35442
```

Specifying Initial Values for Parameters

All the predefined distributions have parameter initialization functions built into them. For the current example, Figure 14.6 shows the initial values that are obtained by the predefined method for the Burr distribution. It also shows the summary of the optimization process and the final parameter estimates.

Figure 14.6 Burr Model Summary for the Truncated and Censored Data

```
                       Initial Parameter Values and
                       Bounds for Burr Distribution

                            Initial          Lower          Upper
             Parameter        Value          Bound          Bound

             Theta          4.78102    1.05367E-8          Infty
             Alpha          2.00000    1.05367E-8          Infty
             Gamma          2.00000    1.05367E-8          Infty

                    Optimization Summary for Burr Distribution

                   Optimization Technique            Trust Region
                   Number of Iterations                         8
                   Number of Function Evaluations              23
                   Log Likelihood                      -148.20614

                    Parameter Estimates for Burr Distribution

                                         Standard                  Approx
         Parameter       Estimate           Error     t Value    Pr > |t|

         Theta            4.76980         0.62492        7.63      <.0001
         Alpha            1.16363         0.58859        1.98      0.0509
         Gamma            5.94081         1.05004        5.66      <.0001
```

You can specify a different set of initial values if estimates are available from fitting the distribution to similar data. For this example, the parameters of the Burr distribution can be initialized with the final parameter estimates of the Burr distribution that were obtained in the first example (shown in Figure 14.3). One of the ways in which you can specify the initial values is as follows:

```
/*------ Specifying initial values using INIT= option -------*/
proc hpseverity data=test_sev2 crit=aicc print=all;
   loss y / lt=threshold rc=limit;

   dist burr(init=(theta=4.62348 alpha=1.15706 gamma=6.41227));
   performance nthreads=2;
run;
```

The names of the parameters specified in the INIT option must match the names used in the definition of the distribution. The results obtained with these initial values are shown in Figure 14.7. These results indicate that new set of initial values causes the optimizer to reach the same solution with fewer iterations and function evaluations as compared to the default initialization.

Figure 14.7 Burr Model Optimization Summary for the Truncated and Censored Data

```
                         The HPSEVERITY Procedure

                  Optimization Summary for Burr Distribution

                Optimization Technique              Trust Region
                Number of Iterations                           5
                Number of Function Evaluations                16
                Log Likelihood                        -148.20614

                  Parameter Estimates for Burr Distribution

                                         Standard                 Approx
         Parameter       Estimate           Error    t Value    Pr > |t|

         Theta            4.76980         0.62492       7.63      <.0001
         Alpha            1.16363         0.58859       1.98      0.0509
         Gamma            5.94081         1.05004       5.66      <.0001
```

An Example of Modeling Regression Effects

Consider a scenario in which the magnitude of the response variable might be affected by some regressor (exogenous or independent) variables. The HPSEVERITY procedure enables you to model the effect of such variables on the distribution of the response variable via an exponential link function. In particular, if you have k random regressor variables denoted by x_j ($j = 1, \ldots, k$), then the distribution of the response variable Y is assumed to have the form

$$Y \sim \exp(\sum_{j=1}^{k} \beta_j x_j) \cdot \mathcal{F}(\Theta)$$

where $\mathcal{F}$ denotes the distribution of Y with parameters Θ and $\beta_j (j = 1, \ldots, k)$ denote the regression parameters (coefficients).

For the effective distribution of Y to be a valid distribution from the same parametric family as $\mathcal{F}$, it is necessary for $\mathcal{F}$ to have a scale parameter. The effective distribution of Y can be written as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

where θ denotes the scale parameter and Ω denotes the set of nonscale parameters. The scale θ is affected by the regressors as

$$\theta = \theta_0 \cdot \exp(\sum_{j=1}^{k} \beta_j x_j)$$

where θ_0 denotes a *base* value of the scale parameter.

Given this form of the model, PROC HPSEVERITY allows a distribution to be a candidate for modeling regression effects only if it has an untransformed or a log-transformed scale parameter.

All the predefined distributions, except the lognormal distribution, have a direct scale parameter (that is, a parameter that is a scale parameter without any transformation). For the lognormal distribution, the parameter μ is a log-transformed scale parameter. This can be verified by replacing μ with a parameter $\theta = e^{\mu}$, which results in the following expressions for the PDF f and the CDF F in terms of θ and σ, respectively, where Φ denotes the CDF of the standard normal distribution:

$$f(x;\theta,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\log(\theta)}{\sigma}\right)^2} \quad \text{and} \quad F(x;\theta,\sigma) = \Phi\left(\frac{\log(x)-\log(\theta)}{\sigma}\right)$$

With this parameterization, the PDF satisfies the $f(x;\theta,\sigma) = \frac{1}{\theta} f(\frac{x}{\theta};1,\sigma)$ condition and the CDF satisfies the $F(x;\theta,\sigma) = F(\frac{x}{\theta};1,\sigma)$ condition. This makes θ a scale parameter. Hence, $\mu = \log(\theta)$ is a log-transformed scale parameter and the lognormal distribution is eligible for modeling regression effects.

The following DATA step simulates a lognormal sample whose scale is decided by the values of the three regressors X1, X2, and X3 as follows:

$$\mu = \log(\theta) = 1 + 0.75\ \mathrm{X1} - \mathrm{X2} + 0.25\ \mathrm{X3}$$

```
/*----------- Lognormal Model with Regressors ------------*/
data test_sev3(keep=y x1-x3
               label='A Lognormal Sample Affected by Regressors');
   array x{*} x1-x3;
   array b{4} _TEMPORARY_ (1 0.75 -1 0.25);
   call streaminit(45678);
   label y='Response Influenced by Regressors';
   Sigma = 0.25;
   do n = 1 to 100;
      Mu = b(1); /* log of base value of scale */
      do i = 1 to dim(x);
         x(i) = rand('UNIFORM');
         Mu = Mu + b(i+1) * x(i);
      end;
      y = exp(Mu) * rand('LOGNORMAL')**Sigma;
      output;
   end;
run;
```

The following PROC HPSEVERITY step fits the lognormal, Burr, and gamma distribution models to this data. The regressors are specified in the SCALEMODEL statement.

```
proc hpseverity data=test_sev3 crit=aicc print=all;
   loss y;
   scalemodel x1-x3;

   dist logn burr gamma;
run;
```

Some of the key results prepared by PROC HPSEVERITY are shown in Figure 14.8 through Figure 14.12. The descriptive statistics of all the variables are shown in Figure 14.8.

Figure 14.8 Summary Results for the Regression Example

```
                              The HPSEVERITY Procedure

                                   Input Data Set

                  Name                                WORK.TEST_SEV3
                  Label    A Lognormal Sample Affected by Regressors

                        Descriptive Statistics for Variable y

               Number of Observations                                 100
               Number of Observations Used for Estimation             100
               Minimum                                            1.17863
               Maximum                                            6.65269
               Mean                                               2.99859
               Standard Deviation                                 1.12845

                  Descriptive Statistics for the Regressor Variables

                                                                         Standard
 Variable              N       Minimum        Maximum          Mean     Deviation

 x1                  100     0.0005115        0.97971       0.51689       0.28206
 x2                  100       0.01883        0.99937       0.47345       0.28885
 x3                  100       0.00255        0.97558       0.48301       0.29709
```

The comparison of the fit statistics of all the models is shown in Figure 14.9. It indicates that the lognormal model is the best model according to each of the likelihood-based statistics.

Figure 14.9 Comparison of Statistics of Fit for the Regression Example

```
                              All Fit Statistics Table

                        -2 Log
    Distribution      Likelihood          AIC            AICC             BIC

    Logn              187.49609*     197.49609*     198.13439*     210.52194*
    Burr              190.69154      202.69154      203.59476      218.32256
    Gamma             188.91483      198.91483      199.55313      211.94069
```

The distribution information and the convergence results of the lognormal model are shown in Figure 14.10. The iteration history gives you a summary of how the optimizer is traversing the surface of the log-likelihood function in its attempt to reach the optimum. Both the change in the log likelihood and the maximum gradient of the objective function with respect to any of the parameters typically approach 0 if the optimizer converges.

Figure 14.10 Convergence Results for the Lognormal Model with Regressors

```
                            The HPSEVERITY Procedure

                            Distribution Information

           Name                                                     Logn
           Description                              Lognormal Distribution
           Number of Distribution Parameters                           2
           Number of Regression Parameters                             3

                      Convergence Status for Logn Distribution

                   Convergence criterion (GCONV=1E-8) satisfied.

                 Optimization Iteration History for Logn Distribution

                        Number of                  Change in
                         Function        Log         Log        Maximum
            Iter      Evaluations  Likelihood  Likelihood       Gradient

               0                2   -93.75285           .        6.16002
               1                4   -93.74805   0.0048055        0.11031
               2                6   -93.74805  1.50188E-6     0.00003376
               3               10   -93.74805  1.4211E-13     3.2401E-12

                     Optimization Summary for Logn Distribution

                   Optimization Technique             Trust Region
                   Number of Iterations                          3
                   Number of Function Evaluations               10
                   Log Likelihood                        -93.74805
```

The final parameter estimates of the lognormal model are shown in Figure 14.11. All the estimates are significantly different from 0. The estimate that is reported for the parameter *Mu* is the base value for the log-transformed scale parameter μ. Let $x_i (1 \leq i \leq 3)$ denote the observed value for regressor Xi. If the lognormal distribution is chosen to model Y, then the effective value of the parameter μ varies with the observed values of regressors as

$$\mu = 1.04047 + 0.65221\, x_1 - 0.91116\, x_2 + 0.16243\, x_3$$

These estimated coefficients are reasonably close to the population parameters (that is, within one or two standard errors).

Figure 14.11 Parameter Estimates for the Lognormal Model with Regressors

```
                    Parameter Estimates for Logn Distribution

                                         Standard                     Approx
           Parameter       Estimate         Error     t Value      Pr > |t|

           Mu               1.04047       0.07614       13.66        <.0001
           Sigma            0.22177       0.01609       13.78        <.0001
           x1               0.65221       0.08167        7.99        <.0001
           x2              -0.91116       0.07946      -11.47        <.0001
           x3               0.16243       0.07782        2.09        0.0395
```

The estimates of the gamma distribution model, which is the next best model according to the fit statistics, are shown in Figure 14.12. The estimate that is reported for the parameter *Theta* is the base value for the scale parameter θ. If the gamma distribution is chosen to model Y, then the effective value of the scale parameter is $\theta = 0.14293 \exp(0.64562\, x_1 - 0.89831\, x_2 + 0.14901\, x_3)$.

Figure 14.12 Parameter Estimates for the Gamma Model with Regressors

```
                    Parameter Estimates for Gamma Distribution

                                         Standard                     Approx
           Parameter       Estimate         Error     t Value      Pr > |t|

           Theta            0.14293       0.02329        6.14        <.0001
           Alpha           20.37726       2.93277        6.95        <.0001
           x1               0.64562       0.08224        7.85        <.0001
           x2              -0.89831       0.07962      -11.28        <.0001
           x3               0.14901       0.07870        1.89        0.0613
```

Syntax: HPSEVERITY Procedure

The following statements are available in the HPSEVERITY procedure:

PROC HPSEVERITY *options* ;
  LOSS < *response-variable* > < / *censoring-truncation-options* > ;
  WEIGHT *weight-variable* ;
  SCALEMODEL *regressor-variable-list* ;
  DIST *distribution-name-or-keyword< (distribution-option)< distribution-name-or-keyword< (distribution-option)> > ... >* ;
  NLOPTIONS *options* ;
  PERFORMANCE *options* ;

Functional Summary

Table 14.1 summarizes the statements and options that control the HPSEVERITY procedure.

Table 14.1 HPSEVERITY Functional Summary

Description	Statement	Option
Statements		
Specifies the response variable to model along with censoring and truncation effects	LOSS	
Specifies the weight variable	WEIGHT	
Specifies the regression variables to model	SCALEMODEL	
Specifies distributions to fit	DIST	
Specifies optimization options	NLOPTIONS	
Specifies performance options	PERFORMANCE	
Data Set Options		
Specifies that the OUTEST= data set contain covariance estimates	PROC HPSEVERITY	COVOUT
Specifies the input data set	PROC HPSEVERITY	DATA=
Specifies the input data set for parameter estimates	PROC HPSEVERITY	INEST=
Specifies the output data set for parameter estimates	PROC HPSEVERITY	OUTEST=
Specifies the output data set for model information	PROC HPSEVERITY	OUTMODELINFO=
Specifies the output data set for statistics of fit	PROC HPSEVERITY	OUTSTAT=
Data Interpretation Options		
Specifies left-censoring	LOSS	LEFTCENSORED=
Specifies left-truncation	LOSS	LEFTTRUNCATED=
Specifies right-censoring	LOSS	RIGHTCENSORED=
Specifies right-truncation	LOSS	RIGHTTRUNCATED=
Model Estimation Options		
Specifies the model selection criterion	PROC HPSEVERITY	CRITERION=
Specifies initial values for model parameters	DIST	INIT=
Specifies the sample to be used for parameter initialization	PROC HPSEVERITY	INITSAMPLE
Specifies the denominator for computing covariance estimates	PROC HPSEVERITY	VARDEF=
Optimization technique	NLOPTIONS	TECHNIQUE=
Optimization Termination Criteria		
Absolute function value criterion	NLOPTIONS	ABSCONV=

Table 14.1 *continued*

Description	Statement	Option
Absolute function difference convergence criterion	NLOPTIONS	ABSFCONV=
Absolute gradient convergence criterion	NLOPTIONS	ABSGCONV=
Absolute parameter convergence criterion	NLOPTIONS	ABSXCONV=
Relative function convergence criterion	NLOPTIONS	FCONV=
Another relative function convergence criterion	NLOPTIONS	FCONV2=
Used in FCONV, GCONV criterion	NLOPTIONS	FSIZE=
Relative gradient convergence criterion	NLOPTIONS	GCONV=
Maximum number of function calls	NLOPTIONS	MAXFUNC=
Maximum number of iterations	NLOPTIONS	MAXITER=
Upper limit on CPU time in seconds	NLOPTIONS	MAXTIME=
Minimum number of iterations	NLOPTIONS	MINITER=
Relative parameter convergence criterion	NLOPTIONS	XCONV=
Used in XCONV criterion	NLOPTIONS	XSIZE=
Displayed Output Options		
Specifies that no output be displayed	PROC HPSEVERITY	NOPRINT
Specifies which output to display	PROC HPSEVERITY	PRINT=
Specifies the verbosity of messages printed to the log	PROC HPSEVERITY	VERBOSE=

PROC HPSEVERITY Statement

PROC HPSEVERITY *options* **;**

The PROC HPSEVERITY statement invokes the procedure. You can specify two types of *options* in the PROC HPSEVERITY statement. One set of *options* controls input and output. The other set of *options* controls the model estimation and selection process.

The following *options* control the input data sets used by PROC HPSEVERITY and various forms of output generated by PROC HPSEVERITY. The *options* are listed in alphabetical order:

COVOUT

specifies that the OUTEST= data set contain the estimate of the covariance structure of the parameters. This option has no effect if the OUTEST= option is not specified. For more information about how the covariance is reported in the OUTEST= data set, see the section “OUTEST= Data Set” on page 442.

DATA=*SAS-data-set*

names the input data set. If the DATA= option is not specified, then the most recently created SAS data set is used.

INEST=*SAS-data-set*

names the input data set that contains the initial values of the parameter estimates to start the optimization process. The initial values specified in the INIT= option in the DIST statement take precedence over any initial values specified in this data set. For more information about the variables in this data set, see the section "INEST= Data Set" on page 441.

INITSAMPLE (*initsample-option***)**

INITSAMPLE (*initsample-option . . . initsample-option***)**

specifies that a sample of the input data be used for initializing the distribution parameters. If you specify more than one *initsample-option*, then separate them with spaces.

When you do not specify initial values for the distribution parameters, PROC HPSEVERITY needs to compute the empirical distribution function (EDF) estimates as part of the default method for parameter initialization. The EDF estimation process can be expensive, especially when you specify censoring or truncation effects for the loss variable. Furthermore, it is not amenable to parallelism due to the sequential nature of the algorithm for truncation effects. You can use the INITSAMPLE option to specify that only a fraction of the input data be used in order to reduce the time taken to compute the EDF estimates. PROC HPSEVERITY uses the uniform random sampling method to select the sample, the size and randomness of which is controlled by the following *initsample-options*:

FRACTION=*number*

specifies the fraction, between 0 and 1, of the input data to be used for sampling.

SEED=*number*

specifies the seed to be used for the uniform random number generator. This option enables you to select the same sample from the same input data across different runs of PROC HPSEVERITY, which can be useful for replicating the results across different runs. If you do not specify the seed value, PROC HPSEVERITY generates a seed that is based on the system clock.

SIZE=*number*

specifies the size of the sample. If the data are distributed across different nodes, then this size applies to the sample that is prepared at each node. For example, let the input data set of size 100,000 observations be distributed across 10 nodes such that each node has 10,000 observations. If you specify SIZE=1000, then each node computes a local EDF estimate by using a sample of size 1,000 selected randomly from its 10,000 observations. If you specify both of the SIZE= and FRACTION= options, then the value specified in the SIZE= option is used and the FRACTION= option is ignored.

NOPRINT

turns off all displayed output. If specified, any value specified for the PRINT= option is ignored.

OUTEST=*SAS-data-set*

names the output data set to contain estimates of the parameter values and their standard errors for each model whose parameter estimation process converges. For more information about the variables in this data set, see the section "OUTEST= Data Set" on page 442.

OUTMODELINFO=*SAS-data-set*

names the output data set to contain the status of each fitted model. The status information includes the convergence status of the optimization process that is used to estimate the parameters, the status of estimating the covariance matrix, and whether a model is the best according to the specified selection criterion. For more information about the variables in this data set, see the section "OUTMODELINFO= Data Set" on page 443.

OUTSTAT=*SAS-data-set*
names the output data set to contain the values of statistics of fit for each model whose parameter estimation process converges. For more information about the variables in this data set, see the section "OUTSTAT= Data Set" on page 443.

PRINT < **(***global-display-option***)** > < **=***display-option* >
PRINT < **(***global-display-option***)** > < **= (***display-option . . . display-option***)** >
specifies the desired displayed output. If you specify more than one *display-option*, then separate them with spaces and enclose them in parentheses.

The following *global-display-option* is available:

ONLY
turns off the default displayed output and displays only the requested output.

The following *display-options* are available:

ALL
displays all the output.

ALLFITSTATS
displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge.

CONVSTATUS
displays the convergence status of the parameter estimation process.

DESCSTATS
displays the descriptive statistics for the response variable and the regressor variables, if they are specified.

ESTIMATES
PARMEST
displays the final estimates of parameters. The estimates are not displayed for models whose parameter estimation process does not converge.

ESTIMATIONDETAILS
displays the details of the estimation process for all the models in one table.

INITIALVALUES
displays the initial values and bounds used for estimating each model.

NLOHISTORY
displays the iteration history of the nonlinear optimization process used for estimating the parameters.

NLOSUMMARY
displays the summary of the nonlinear optimization process used for estimating the parameters.

NONE

displays none of the output. If specified, this option overrides all the other display options. The default displayed output is also suppressed.

SELECTION

SELECT

displays the model selection table.

STATISTICS

FITSTATS

displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

If the PRINT= option is not specified or the ONLY *global-display-option* is not specified, then the default displayed output is equivalent to specifying PRINT=(SELECTION CONVSTATUS NLO-SUMMARY STATISTICS ESTIMATES).

VARDEF=*option*

specifies the denominator to use for computing the covariance estimates. You can specify one of the following values for *option*:

DF

specifies that the number of nonmissing observations minus the model degrees of freedom (number of parameters) be used.

N

specifies that the number of nonmissing observations be used.

For more information about the covariance estimation, see the section "Estimating Covariance and Standard Errors" on page 412.

VERBOSE=*verbosity-level*

specifies the amount of messages printed to the SAS log by PROC HPSEVERITY. A higher number prints messages with the same or more detail.

The following *options* control the model estimation and selection process:

CRITERION=*criterion-option*

CRITERIA=*criterion-option*

CRIT=*criterion-option*

specifies the model selection criterion.

If two or more models are specified for estimation, then the one with the best value for the selection criterion is chosen as the best model. If the OUTMODELINFO= data set is specified, then the best model's observation has a value of 1 for the _SELECTED_ variable. You can specify one of the following *criterion-options*:

AIC

specifies Akaike's information criterion (AIC) as the selection criterion. A lower value is deemed better.

AICC

specifies the finite-sample corrected Akaike's information criterion (AICC) as the selection criterion. A lower value is deemed better.

BIC

specifies the Schwarz Bayesian information criterion (BIC) as the selection criterion. A lower value is deemed better.

LOGLIKELIHOOD

LL

specifies $-2 * \log(L)$ as the selection criterion, where L is the likelihood of the data. A lower value is deemed better. This is the default.

For more information about these *criterion-options*, see the section "Statistics of Fit" on page 421.

DIST Statement

DIST *distribution-name-or-keyword* < *(distribution-option)* > < *distribution-name-or-keyword* < *(distribution-option)* > > ... **;**

The DIST statement specifies candidate distributions to be estimated by the HPSEVERITY procedure. You can specify multiple DIST statements, and each statement can contain one or more distribution specifications.

For your convenience, PROC HPSEVERITY provides the following 10 different predefined distributions (the name in the parentheses is the name to use in the DIST statement): Burr (BURR), exponential (EXP), gamma (GAMMA), generalized Pareto (GPD), inverse Gaussian or Wald (IGAUSS), lognormal (LOGN), Pareto (PARETO), Tweedie (TWEEDIE), scaled Tweedie (STWEEDIE), and Weibull (WEIBULL). These are described in detail in the section "Predefined Distributions" on page 400.

You can specify any of the predefined distributions or any distribution that you have defined. If the specified distribution is not a predefined distribution, then you must submit the CMPLIB= system option with appropriate libraries before you submit the PROC HPSEVERITY step to enable the procedure to find the functions associated with your distribution. The predefined distributions are defined in the Sashelp.Svrtdist library. However, you are not required to specify this library in the CMPLIB= system option.

As a convenience, you can also use a shortcut keyword _PREDEFINED_ to indicate that you want to fit all the predefined distributions. **NOTE:** The TWEEDIE and STWEEDIE distributions are not included by this keyword. If you want to fit these distributions, then you must specify them explicitly.

The following *distribution-option* values can be used in the DIST statement for a distribution name that is not a shortcut keyword:

INIT=*(name=value . . . name=value)*

specifies the initial values to be used for the distribution parameters to start the parameter estimation process. The values must be specified by parameter names. The parameter names must match the names used in the model definition.

For example, let a model M's definition contain a M_PDF function with following signature:

```
function M_PDF(x, alpha, beta);
```

For this model, the names **`alpha`** and **`beta`** must be used for the INIT option. The names are case-insensitive. If you do not specify initial values for some parameters in the INIT statement, then a default value of 0.001 is assumed for those parameters. If you specify an incorrect parameter, PROC HPSEVERITY prints a warning to the SAS log and does not fit the model. All specified values must be nonmissing.

If you are modeling regression effects, then the initial value of the first distribution parameter (**`alpha`** in the preceding example) should be the initial *base* value of the scale parameter or log-transformed scale parameter. For more information, see the section "Estimating Regression Effects" on page 416.

The use of INIT= option is one of the three methods available for initializing the parameters. For more information, see the section "Parameter Initialization" on page 415. If none of the initialization methods is used, then PROC HPSEVERITY initializes all parameters to 0.001.

LOSS Statement

LOSS < *response-variable-name* > < / *censoring-truncation-options* > **;**

The LOSS statement specifies the name of the response or loss variable whose distribution needs to be modeled. You can also specify additional options to indicate any truncation or censoring of the response. The specification of response variable is optional if at least one type of censoring is specified. You must specify a response variable if no censoring is specified. If you specify more than one LOSS statement, then the first statement is used.

All the analysis variables specified in this statement must be present in the input data set that is specified by using the DATA= option in the PROC HPSEVERITY statement. The response variable is expected to have nonmissing values. If the variable has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

The following *censoring-truncation-options* can be used in the LOSS statement:

LEFTCENSORED=*variable-name*
LC=*variable-name*
LEFTCENSORED=*number*
LC=*number*

specifies the left-censoring variable or a global left-censoring limit.

You can use the *variable-name* argument to specify a data set variable that contains the left-censoring limit. If the value of this variable is missing, then PROC HPSEVERITY assumes that such observations are not left-censored.

Alternatively, you can use the *number* argument to specify a left-censoring limit value that applies to all the observations in the data set. This limit must be a nonzero positive number.

By the definition of left-censoring, an exact value of the response is not known when it is less than or equal to the left-censoring limit. If the response variable is specified and the value of that variable is less than or equal to the value of the left-censoring limit for some observations, then PROC HPSEVERITY treats such observations as left-censored and the value of the response variable is ignored. If the response variable is specified and the value of that variable is greater than the value of the left-censoring limit for some observations, then PROC HPSEVERITY assumes that such observations are not left-censored.

If you specify left-censoring, then you must also specify right-censoring. PROC HPSEVERITY ignores observations that are purely left-censored. In other words, left-censoring specification is allowed only in the context of interval-censoring.

If both right-censoring and left-censoring limits are specified, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about left-censoring, see the section "Censoring and Truncation" on page 410.

LEFTTRUNCATED | LT=*variable-name*
LT=*variable-name*
LEFTTRUNCATED=*number*
LT=*number*

specifies the left-truncation variable or a global left-truncation threshold.

You can use the *variable-name* argument to specify a data set variable that contains the left-truncation threshold. If the value of this variable is missing or 0 for some observations, then PROC HPSEVERITY assumes that such observations are not left-truncated.

Alternatively, you can use the *number* argument to specify a left-truncation threshold that applies to all the observations in the data set. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of left-truncation, you can observe only a value that is greater than the left-truncation threshold. If a response variable value is less than or equal to the left-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about left-truncation, see the section "Censoring and Truncation" on page 410.

RIGHTCENSORED | RC=*variable-name*
RC=*variable-name*
RIGHTCENSORED=*number*
RC=*number*

specifies the right-censoring variable or a global right-censoring limit.

You can use the *variable-name* argument to specify a data set variable that contains the right-censoring limit. If the value of this variable is missing, then PROC HPSEVERITY assumes that such observations are not right-censored.

Alternatively, you can use the *number* argument to specify a right-censoring limit value that applies to all the observations in the data set. This limit must be a nonzero positive number.

By the definition of right-censoring, an exact value of the response is not known when it is greater than or equal to the right-censoring limit. If the response variable is specified and the value of that variable is greater than or equal to the value of the right-censoring limit for some observations, then PROC HPSEVERITY treats such observations as right-censored and the value of the response variable is ignored. If the response variable is specified and the value of that variable is less than the value of the right-censoring limit for some observations, then PROC HPSEVERITY assumes that such observations are not right-censored and the value of the right-censoring limit is ignored.

If both right-censoring and left-censoring limits are specified, then the left-censoring limit must be greater than or equal to the right-censoring limit. If both limits are identical, then the observation is assumed to be uncensored.

For more information about right-censoring, see the section "Censoring and Truncation" on page 410.

RIGHTTRUNCATED | RT=*variable-name*
RT=*variable-name*
RIGHTTRUNCATED=*number*
RT=*number*

specifies the right-truncation variable or a global right-truncation threshold.

You can use the *variable-name* argument to specify a data set variable that contains the right-truncation threshold. If the value of this variable is missing for some observations, then PROC HPSEVERITY assumes that such observations are not right-truncated.

Alternatively, you can use the *number* argument to specify a right-truncation threshold that applies to all the observations in the data set. This threshold must be a nonzero positive number.

It is assumed that the response variable contains the observed values. By the definition of right-truncation, you can observe only a value that is less than or equal to the right-truncation threshold. If a response variable value is greater than the right-truncation threshold, a warning is printed to the SAS log, and the observation is ignored. For more information about right-truncation, see the section "Censoring and Truncation" on page 410.

NLOPTIONS Statement

NLOPTIONS *options* **;**

The HPSEVERITY procedure uses the nonlinear optimization (NLO) subsystem to perform nonlinear optimization of the likelihood function to obtain the estimates of distribution and regression parameters. You can use the NLOPTIONS statement to control different aspects of this optimization process. If you specify more than one NLOPTIONS statement, then the first statement is used.

For most problems, the default settings of the optimization process are adequate. However, in some cases it might be useful to change the optimization technique or to change the maximum number of iterations. The following statement uses the MAXITER= option to set the maximum number of iterations to 200 and uses the TECH= option to change the optimization technique to the double-dogleg optimization (DBLDOG) rather than the default technique, the trust region optimization (TRUREG), used in the HPSEVERITY procedure:

```
nloptions tech=dbldog maxiter=200;
```

The following *options* values can be used in the NLOPTIONS statement:

ABSCONV=*r*

ABSTOL=*r*

specifies an absolute function value convergence criterion. For minimization, termination requires $f(\theta^{(k)}) \leq r$. The default value of *r* is the negative square root of the largest double-precision value, which serves only as a protection against overflows.

ABSFCONV=*r*

ABSFTOL=*r*

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

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

The same formula is used for the NMSIMP technique, but $\theta^{(k)}$ is defined as the vertex with the lowest function value, and $\theta^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is $r = 0$.

ABSGCONV=*r*

ABSGTOL=*r*

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

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

This criterion is not used by the NMSIMP technique. The default value is *r*=1E–5.

ABSXCONV=*r*

ABSXTOL=*r*

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

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

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

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

or a small simplex size,

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

where the simplex size $\delta^{(k)}$ is defined as the L1 distance from the simplex vertex $\xi^{(k)}$ with the smallest function value to the other simplex points $\theta_l^{(k)} \neq \xi^{(k)}$:

$$\delta^{(k)} = \sum \| \theta_l^{(k)} - \xi^{(k)} \|_1$$

The default is *r*=1E–8 for the NMSIMP technique and $r = 0$ otherwise.

FCONV=*r*

FTOL=*r*

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

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

where FSIZE is defined by the FSIZE= option. The same formula is used for the NMSIMP technique, but $\theta^{(k)}$ is defined as the vertex with the lowest function value, and $\theta^{(k-1)}$ is defined as the vertex with the highest function value in the simplex.

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

FCONV2=*r*

FTOL2=*r*

specifies another function convergence criterion.

For all techniques except NMSIMP, termination requires a small predicted reduction of the objective function:

$$df^{(k)} \approx f(\theta^{(k)}) - f(\theta^{(k)} + s^{(k)})$$

The predicted reduction

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

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

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

For the NMSIMP technique, termination requires a small standard deviation of the function values of the $p + 1$ simplex vertices $\theta_l^{(k)}$, $l = 0, \ldots, p$, $\sqrt{\frac{1}{p+1} \sum_l \left[f(\theta_l^{(k)}) - \overline{f}(\theta^{(k)}) \right]^2} \leq r$ where $\overline{f}(\theta^{(k)}) = \frac{1}{p+1} \sum_l f(\theta_l^{(k)})$. If there are p_{act} boundary constraints active at $\theta^{(k)}$, the mean and standard deviation are computed only for the $p + 1 - p_{act}$ unconstrained vertices.

The default value is r=1E–6 for the NMSIMP technique and $r = 0$ otherwise.

FSIZE=*r*

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

GCONV=*r*

GTOL=*r*

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

$$\frac{g(\theta^{(k)})^T [H^{(k)}]^{-1} g(\theta^{(k)})}{\max(|f(\theta^{(k)})|, \mathrm{FSIZE})} \leq r$$

where FSIZE is defined by the FSIZE= option. For the CONGRA technique (where a reliable Hessian estimate H is not available), the following criterion is used:

$$\frac{\| g(\theta^{(k)}) \|_2^2 \quad \| s(\theta^{(k)}) \|_2}{\| g(\theta^{(k)}) - g(\theta^{(k-1)}) \|_2 \max(|f(\theta^{(k)})|, \mathrm{FSIZE})} \leq r$$

This criterion is not used by the NMSIMP technique. The default value is $r = 1E - 8$.

MAXFUNC=*i*

MAXFU=*i*

specifies the maximum number *i* of function calls in the optimization process. The default values are

- TRUREG, NRRIDG, NEWRAP: 125
- QUANEW, DBLDOG: 500
- CONGRA: 1000
- NMSIMP: 3000

Note that the optimization can terminate only after completing a full iteration. Therefore, the number of function calls that is actually performed can exceed the number that is specified by the MAX-FUNC= option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number *i* of iterations in the optimization process. The default values are

- TRUREG, NRRIDG, NEWRAP: 50
- QUANEW, DBLDOG: 200
- CONGRA: 400
- NMSIMP: 1000

These default values are also valid when *i* is specified as a missing value.

MAXTIME=*r*

specifies an upper limit of *r* seconds of CPU time for the optimization process. The default value is the largest floating-point double representation of your computer. The time specified by the MAXTIME= option is checked only once at the end of each iteration. Therefore, the actual running time can be much longer than that specified by the MAXTIME= option. The actual running time includes the rest of the time needed to finish the iteration and the time needed to generate the output of the results.

MINITER=*i*

MINIT=*i*

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

TECHNIQUE=*name*

TECH=*name*

specifies the optimization technique. Valid values for *name* are as follows:

CONGRA	performs a conjugate-gradient optimization.
DBLDOG	performs a version of double-dogleg optimization.
NMSIMP	performs a Nelder-Mead simplex optimization.
NONE	does not perform any optimization. This option can be used as follows: • to perform a grid search without optimization • to compute estimates and predictions that cannot be obtained efficiently with any of the optimization techniques
NEWRAP	performs a Newton-Raphson optimization that combines a line-search algorithm with ridging.
NRRIDG	performs a Newton-Raphson optimization with ridging.
QUANEW	performs a quasi-Newton optimization
TRUREG	performs a trust region optimization. This is the default estimation method.

For more information about optimization algorithms, see the section "Details of Optimization Algorithms" on page 413.

XCONV=*r*

XTOL=*r*

specifies the relative parameter convergence criterion. For all techniques except NMSIMP, termination requires a small relative parameter change in subsequent iterations.

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

For the NMSIMP technique, the same formula is used, but $\theta_j^{(k)}$ is defined as the vertex with the lowest function value and $\theta_j^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is r=1E–8 for the NMSIMP technique and $r = 0$ otherwise.

XSIZE=*r*

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

PERFORMANCE Statement

PERFORMANCE *options* **;**

The PERFORMANCE statement defines performance parameters for distributed and multithreaded computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a High-Performance Analytics procedure.

With the PERFORMANCE statement, you can also control whether a High-Performance Analytics procedure executes in SMP or MPP mode.

The PERFORMANCE statement for SAS High-Performance Analytics procedures is documented in the section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

SCALEMODEL Statement

SCALEMODEL *regressor-variable-list* **;**

The SCALEMODEL statement specifies regression variables. All the variables specified in this statement must be present in the input data set that is specified by the DATA= option in the PROC HPSEVERITY statement. The scale parameter of each candidate distribution is linked to a linear combination of these regression variables along with an intercept. If a distribution does not have a scale parameter, then a model based on that distribution is not estimated. If you specify more than one SCALEMODEL statement, then the first statement is used.

The regressor variables are expected to have nonmissing values. If any of the variables has a missing value in an observation, then a warning is written to the SAS log and that observation is ignored.

For more information about modeling regression effects, see the section "Estimating Regression Effects" on page 416.

WEIGHT Statement

WEIGHT *variable-name* **;**

The WEIGHT statement specifies the name of a variable whose values represent the weight of each observation. PROC HPSEVERITY associates a weight of w to each observation, where w is the value of the WEIGHT variable for the observation. If the weight value is missing or less than or equal to 0, then the observation is ignored and a warning is written to the SAS log. When the WEIGHT statement is not specified, each observation is assigned a weight of 1. If you specify more than one WEIGHT statement, then the last statement is used.

The weights are normalized so that they add up to the actual sample size. In particular, the weight of each observation is multiplied by $\frac{N}{\sum_{i=1}^{N} w_i}$, where N is the sample size.

Details: HPSEVERITY Procedure

Predefined Distributions

PROC HPSEVERITY assumes the following model for the response variable Y

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is a continuous probability distribution with parameters Θ. The model hypothesizes that the observed response is generated from a stochastic process that is governed by the distribution $\mathcal{F}$. This model is typically referred to as the error model. Given a representative input sample of response variable values, PROC HPSEVERITY estimates the model parameters for any distribution $\mathcal{F}$ and computes the statistics of fit for each model. This enables you to find the distribution that is most likely to generate the observed sample.

A set of predefined distributions is provided with the HPSEVERITY procedure. A summary of the distributions is provided in Table 14.2. For each distribution, the table lists the name of the distribution that should be used in the DIST statement, the parameters of the distribution along with their bounds, and the mathematical expressions for the probability density function (PDF) and cumulative distribution function (CDF) of the distribution.

All the predefined distributions, except LOGN and TWEEDIE, are parameterized such that their first parameter is the scale parameter. For LOGN, the first parameter μ is a log-transformed scale parameter. TWEEDIE does not have a scale parameter. The presence of scale parameter or a log-transformed scale parameter enables you to use all of the predefined distributions, except TWEEDIE, as a candidate for estimating regression effects.

A distribution model is associated with each predefined distribution. You can also define your own distribution model, which is a set of functions and subroutines that you define by using the FCMP procedure. For more information, see the section "Defining a Severity Distribution Model with the FCMP Procedure" on page 424.

Table 14.2 Predefined HPSEVERITY Distributions

Name	Distribution	Parameters	PDF (f) and CDF (F)
BURR	Burr	$\theta > 0, \alpha > 0,$ $\gamma > 0$	$f(x) = \frac{\alpha \gamma z^{\gamma}}{x(1+z^{\gamma})^{(\alpha+1)}}$ $F(x) = 1 - \left(\frac{1}{1+z^{\gamma}}\right)^{\alpha}$
EXP	Exponential	$\theta > 0$	$f(x) = \frac{1}{\theta} e^{-z}$ $F(x) = 1 - e^{-z}$
GAMMA	Gamma	$\theta > 0, \alpha > 0$	$f(x) = \frac{z^{\alpha} e^{-z}}{x \Gamma(\alpha)}$ $F(x) = \frac{\gamma(\alpha, z)}{\Gamma(\alpha)}$
GPD	Generalized Pareto	$\theta > 0, \xi > 0$	$f(x) = \frac{1}{\theta} (1 + \xi z)^{-1-1/\xi}$ $F(x) = 1 - (1 + \xi z)^{-1/\xi}$
IGAUSS	Inverse Gaussian (Wald)	$\theta > 0, \alpha > 0$	$f(x) = \frac{1}{\theta} \sqrt{\frac{\alpha}{2\pi z^3}}\, e^{\frac{-\alpha(z-1)^2}{2z}}$ $F(x) = \Phi\left((z-1)\sqrt{\frac{\alpha}{z}}\right) +$ $\Phi\left(-(z+1)\sqrt{\frac{\alpha}{z}}\right) e^{2\alpha}$
LOGN	Lognormal	μ (no bounds), $\sigma > 0$	$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$ $F(x) = \Phi\left(\frac{\log(x)-\mu}{\sigma}\right)$
PARETO	Pareto	$\theta > 0, \alpha > 0$	$f(x) = \frac{\alpha \theta^{\alpha}}{(x+\theta)^{\alpha+1}}$ $F(x) = 1 - \left(\frac{\theta}{x+\theta}\right)^{\alpha}$
TWEEDIE	Tweedie[6]	$\mu > 0, \phi > 0,$ $p > 1$	$f(x) = a(x, \phi) \exp\left[\frac{1}{\phi}\left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$ $F(x) = \int_0^x f(t)dt$
STWEEDIE	Scaled Tweedie[6]	$\theta > 0, \lambda > 0,$ $1 < p < 2$	$f(x) = a(x, \theta, \lambda, p) \exp\left(-\frac{x}{\theta} - \lambda\right)$ $F(x) = \int_0^x f(t)dt$
WEIBULL	Weibull	$\theta > 0, \tau > 0$	$f(x) = \frac{1}{x} \tau z^{\tau} e^{-z^{\tau}}$ $F(x) = 1 - e^{-z^{\tau}}$

Notes:
1. $z = x/\theta$, wherever z is used.
2. θ denotes the scale parameter for all the distributions. For LOGN, $\log(\theta) = \mu$.
3. Parameters are listed in the order in which they are defined in the distribution model.
4. $\gamma(a, b) = \int_0^b t^{a-1} e^{-t} dt$ is the lower incomplete gamma function.
5. $\Phi(y) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{y}{\sqrt{2}}\right)\right)$ is the standard normal CDF.
6. For more information, see the section “Tweedie Distributions” on page 402.

Tweedie Distributions

Tweedie distributions are a special case of the exponential dispersion family (Jørgensen, 1987) with a property that the variance of the distribution is equal to $\phi\mu^p$, where μ is the mean of the distribution, ϕ is a dispersion parameter, and p is an index parameter as discovered by Tweedie (1984). The distribution is defined for all values of p except for values of p in the open interval $(0, 1)$. Many important known distributions are a special case of Tweedie distributions including normal (p=0), Poisson (p=1), gamma (p=2), and the inverse Gaussian (p=3). Apart from these special cases, the probability density function (PDF) of the Tweedie distribution does not have an analytic expression. For $p > 1$, it has the form (Dunn and Smyth 2005),

$$f(x; \mu, \phi, p) = a(x, \phi) \exp\left[\frac{1}{\phi}\left(\frac{x\mu^{1-p}}{1-p} - \kappa(\mu, p)\right)\right]$$

where $\kappa(\mu, p) = \mu^{2-p}/(2-p)$ for $p \neq 2$ and $\kappa(\mu, p) = \log(\mu)$ for $p = 2$. The function $a(x, \phi)$ does not have an analytical expression. It is typically evaluated using series expansion methods described in Dunn and Smyth (2005).

For $1 < p < 2$, the Tweedie distribution is a compound Poisson-gamma mixture distribution, which is the distribution of S defined as

$$S = \sum_{i=1}^{N} X_i$$

where $N \sim \text{Poisson}(\lambda)$ and $X_i \sim \text{gamma}(\alpha, \theta)$ are independent and identically distributed gamma random variables with shape parameter α and scale parameter θ. At $X = 0$, the density is a probability mass that is governed by the Poisson distribution, and for values of $X > 0$, it is a mixture of gamma variates with Poisson mixing probability. The parameters λ, α, and θ are related to the natural parameters μ, ϕ, and p of the Tweedie distribution as

$$\begin{aligned}\lambda &= \frac{\mu^{2-p}}{\phi(2-p)}\\ \alpha &= \frac{2-p}{p-1}\\ \theta &= \phi(p-1)\mu^{p-1}\end{aligned}$$

The mean of a Tweedie distribution is positive for $p > 1$.

Two predefined versions of the Tweedie distribution are provided with the HPSEVERITY procedure. The first version, named TWEEDIE and defined for $p > 1$, has the natural parameterization with parameters μ, ϕ, and p. The second version, named STWEEDIE and defined for $1 < p < 2$, is the version with a scale parameter. It corresponds to the compound Poisson-gamma distribution with gamma scale parameter θ, Poisson mean parameter λ, and the index parameter p. The index parameter decides the shape parameter α of the gamma distribution as

$$\alpha = \frac{2-p}{p-1}$$

The parameters θ and λ of the STWEEDIE distribution are related to the parameters μ and ϕ of the TWEEDIE distribution as

$$\mu = \lambda\theta\alpha$$
$$\phi = \frac{(\lambda\theta\alpha)^{2-p}}{\lambda(2-p)} = \frac{\theta}{(p-1)(\lambda\theta\alpha)^{p-1}}$$

You can fit either version when there are no regression variables. Each version has its own merits. If you fit the TWEEDIE version, you have the direct estimate of the overall mean of the distribution. If you are interested in the most practical range of the index parameter $1 < p < 2$, then you can fit the STWEEDIE version, which provides you direct estimates of the Poisson and gamma components that comprise the distribution (an estimate of the gamma shape parameter α is easily obtained from the estimate of p).

If you want to estimate the effect of exogenous (regression) variables on the distribution, then you must use the STWEEDIE version, because PROC HPSEVERITY requires a distribution to have a scale parameter in order to estimate regression effects. For more information, see the section "Estimating Regression Effects" on page 416. The gamma scale parameter θ is the scale parameter of the STWEEDIE distribution. If you are interested in determining the effect of regression variables on the mean of the distribution, you can do so by first fitting the STWEEDIE distribution to determine the effect of the regression variables on the scale parameter θ. Then, you can easily estimate how the mean of the distribution μ is affected by the regression variables using the relationship $\mu = c\theta$, where $c = \lambda\alpha = \lambda(2-p)/(p-1)$. The estimates of the regression parameters remain the same, whereas the estimate of the intercept parameter is adjusted by the estimates of the λ and p parameters.

Parameter Initialization for Predefined Distributions

The parameters are initialized by using the method of moments for all the distributions, except for the gamma and the Weibull distributions. For the gamma distribution, approximate maximum likelihood estimates are used. For the Weibull distribution, the method of percentile matching is used.

Given n observations of the severity value y_i $(1 \leq i \leq n)$, the estimate of kth raw moment is denoted by m_k and computed as

$$m_k = \frac{1}{n}\sum_{i=1}^{n} y_i^k$$

The $100p$th percentile is denoted by π_p $(0 \leq p \leq 1)$. By definition, π_p satisfies

$$F(\pi_p-) \leq p \leq F(\pi_p)$$

where $F(\pi_p-) = \lim_{h\downarrow 0} F(\pi_p - h)$. PROC HPSEVERITY uses the following practical method of computing π_p. Let $\hat{F}_n(y)$ denote the empirical distribution function (EDF) estimate at a severity value y. Let y_p^- and y_p^+ denote two consecutive values in the ascending sequence of y values such that $\hat{F}_n(y_p^-) < p$ and $\hat{F}_n(y_p^+) \geq p$. Then, the estimate $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = y_p^- + \frac{p - \hat{F}_n(y_p^-)}{\hat{F}_n(y_p^+) - \hat{F}_n(y_p^-)}(y_p^+ - y_p^-)$$

Let ϵ denote the smallest double-precision floating-point number such that $1 + \epsilon > 1$. This machine precision constant can be obtained by using the CONSTANT function in Base SAS software.

The details of how parameters are initialized for each predefined distribution are as follows:

BURR The parameters are initialized by using the method of moments. The kth raw moment of the Burr distribution is:

$$E[X^k] = \frac{\theta^k \Gamma(1 + k/\gamma)\Gamma(\alpha - k/\gamma)}{\Gamma(\alpha)}, \quad -\gamma < k < \alpha\gamma$$

Three moment equations $E[X^k] = m_k$ $(k = 1, 2, 3)$ need to be solved for initializing the three parameters of the distribution. In order to get an approximate closed form solution, the second shape parameter $\hat{\gamma}$ is initialized to a value of 2. If $2m_3 - 3m_1m_2 > 0$, then simplifying and solving the moment equations yields the following feasible set of initial values:

$$\hat{\theta} = \sqrt{\frac{m_2 m_3}{2m_3 - 3m_1m_2}}, \quad \hat{\alpha} = 1 + \frac{m_3}{2m_3 - 3m_1m_2}, \quad \hat{\gamma} = 2$$

If $2m_3 - 3m_1m_2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \sqrt{m_2}, \quad \hat{\alpha} = 2, \quad \hat{\gamma} = 2$$

EXP The parameters are initialized by using the method of moments. The kth raw moment of the exponential distribution is:

$$E[X^k] = \theta^k \Gamma(k + 1), \quad k > -1$$

Solving $E[X] = m_1$ yields the initial value of $\hat{\theta} = m_1$.

GAMMA The parameter α is initialized by using its *approximate* maximum likelihood (ML) estimate. For a set of n independent and identically distributed observations y_i $(1 \le i \le n)$ drawn from a gamma distribution, the log likelihood l is defined as follows:

$$\begin{aligned} l &= \sum_{i=1}^{n} \log\left(y_i^{\alpha-1} \frac{e^{-y_i/\theta}}{\theta^\alpha \Gamma(\alpha)} \right) \\ &= (\alpha - 1)\sum_{i=1}^{n} \log(y_i) - \frac{1}{\theta}\sum_{i=1}^{n} y_i - n\alpha\log(\theta) - n\log(\Gamma(\alpha)) \end{aligned}$$

Using a shorter notation of $\sum$ to denote $\sum_{i=1}^{n}$ and solving the equation $\partial l/\partial\theta = 0$ yields the following ML estimate of θ:

$$\hat{\theta} = \frac{\sum y_i}{n\alpha} = \frac{m_1}{\alpha}$$

Substituting this estimate in the expression of l and simplifying gives

$$l = (\alpha - 1)\sum \log(y_i) - n\alpha - n\alpha\log(m_1) + n\alpha\log(\alpha) - n\log(\Gamma(\alpha))$$

Let d be defined as follows:

$$d = \log(m_1) - \frac{1}{n}\sum \log(y_i)$$

Solving the equation $\partial l / \partial \alpha = 0$ yields the following expression in terms of the digamma function, $\psi(\alpha)$:

$$\log(\alpha) - \psi(\alpha) = d$$

The digamma function can be approximated as follows:

$$\hat{\psi}(\alpha) \approx \log(\alpha) - \frac{1}{\alpha}\left(0.5 + \frac{1}{12\alpha + 2}\right)$$

This approximation is within 1.4% of the true value for all the values of $\alpha > 0$ except when α is arbitrarily close to the positive root of the digamma function (which is approximately 1.461632). Even for the values of α that are close to the positive root, the absolute error between true and approximate values is still acceptable ($|\hat{\psi}(\alpha) - \psi(\alpha)| < 0.005$ for $\alpha > 1.07$). Solving the equation that arises from this approximation yields the following estimate of α:

$$\hat{\alpha} = \frac{3 - d + \sqrt{(d-3)^2 + 24d}}{12d}$$

If this approximate ML estimate is infeasible, then the method of moments is used. The kth raw moment of the gamma distribution is:

$$E[X^k] = \theta^k \frac{\Gamma(\alpha + k)}{\Gamma(\alpha)}, \quad k > -\alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial value for α:

$$\hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then α is initialized as follows:

$$\hat{\alpha} = 1$$

After computing the estimate of α, the estimate of θ is computed as follows:

$$\hat{\theta} = \frac{m_1}{\hat{\alpha}}$$

Both the maximum likelihood method and the method of moments arrive at the same relationship between $\hat{\alpha}$ and $\hat{\theta}$.

GPD The parameters are initialized by using the method of moments. Notice that for $\xi > 0$, the CDF of the generalized Pareto distribution (GPD) is:

$$\begin{aligned} F(x) &= 1 - \left(1 + \frac{\xi x}{\theta}\right)^{-1/\xi} \\ &= 1 - \left(\frac{\theta/\xi}{x + \theta/\xi}\right)^{1/\xi} \end{aligned}$$

This is equivalent to a Pareto distribution with scale parameter $\theta_1 = \theta/\xi$ and shape parameter $\alpha = 1/\xi$. Using this relationship, the parameter initialization method used for the

PARETO distribution is used to get the following initial values for the parameters of the GPD distribution:

$$\hat{\theta} = \frac{m_1 m_2}{2(m_2 - m_1^2)}, \quad \hat{\xi} = \frac{m_2 - 2m_1^2}{2(m_2 - m_1^2)}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = \frac{m_1}{2}, \quad \hat{\xi} = \frac{1}{2}$$

IGAUSS The parameters are initialized by using the method of moments. The standard parameterization of the inverse Gaussian distribution (also known as the Wald distribution), in terms of the location parameter μ and shape parameter λ, is as follows (Klugman, Panjer, and Willmot 1998, p. 583):

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2\mu^2 x}\right)$$
$$F(x) = \Phi\left(\left(\frac{x}{\mu} - 1\right)\sqrt{\frac{\lambda}{x}}\right) + \Phi\left(-\left(\frac{x}{\mu} + 1\right)\sqrt{\frac{\lambda}{x}}\right) \exp\left(\frac{2\lambda}{\mu}\right)$$

For this parameterization, it is known that the mean is $E[X] = \mu$ and the variance is $Var[X] = \mu^3/\lambda$, which yields the second raw moment as $E[X^2] = \mu^2(1+\mu/\lambda)$ (computed by using $E[X^2] = Var[X] + (E[X])^2$).

The predefined IGAUSS distribution in PROC HPSEVERITY uses the following alternate parameterization to allow the distribution to have a scale parameter, θ:

$$f(x) = \sqrt{\frac{\alpha\theta}{2\pi x^3}} \exp\left(\frac{-\alpha(x-\theta)^2}{2x\theta}\right)$$
$$F(x) = \Phi\left(\left(\frac{x}{\theta} - 1\right)\sqrt{\frac{\alpha\theta}{x}}\right) + \Phi\left(-\left(\frac{x}{\theta} + 1\right)\sqrt{\frac{\alpha\theta}{x}}\right) \exp(2\alpha)$$

The parameters θ (scale) and α (shape) of this alternate form are related to the parameters μ and λ of the preceding form such that $\theta = \mu$ and $\alpha = \lambda/\mu$. Using this relationship, the first and second raw moments of the IGAUSS distribution are:

$$E[X] = \theta$$
$$E[X^2] = \theta^2\left(1 + \frac{1}{\alpha}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = \frac{m_1^2}{m_2 - m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance), then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 1$$

LOGN The parameters are initialized by using the method of moments. The kth raw moment of the lognormal distribution is:

$$E[X^k] = \exp\left(k\mu + \frac{k^2\sigma^2}{2}\right)$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\mu} = 2\log(m1) - \frac{\log(m2)}{2}, \quad \hat{\sigma} = \sqrt{\log(m2) - 2\log(m1)}$$

PARETO The parameters are initialized by using the method of moments. The kth raw moment of the Pareto distribution is:

$$E[X^k] = \frac{\theta^k \Gamma(k+1)\Gamma(\alpha-k)}{\Gamma(\alpha)}, -1 < k < \alpha$$

Solving $E[X] = m_1$ and $E[X^2] = m_2$ yields the following initial values:

$$\hat{\theta} = \frac{m_1 m_2}{m_2 - 2m_1^2}, \quad \hat{\alpha} = \frac{2(m_2 - m_1^2)}{m_2 - 2m_1^2}$$

If $m_2 - m_1^2 < \epsilon$ (almost zero sample variance) or $m_2 - 2m_1^2 < \epsilon$, then the parameters are initialized as follows:

$$\hat{\theta} = m_1, \quad \hat{\alpha} = 2$$

TWEEDIE The parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. The parameter μ is the mean of the distribution. Hence, it is initialized to the sample mean as

$$\hat{\mu} = m_1$$

Variance of a Tweedie distribution is equal to $\phi\mu^p$. Thus, the sample variance is used to initialize the value of ϕ as

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

STWEEDIE STWEEDIE is a compound Poisson-gamma mixture distribution with mean $\mu = \lambda\theta\alpha$, where α is the shape parameter of the gamma random variables in the mixture and the parameter p is determined solely by α. First, the parameter p is initialized by assuming that the sample is generated from a gamma distribution with shape parameter α and by computing $\hat{p} = \frac{\hat{\alpha}+2}{\hat{\alpha}+1}$. The initial value $\hat{\alpha}$ is obtained from using the method previously described for the GAMMA distribution. As done for initializing the parameters of the TWEEDIE distribution, the sample mean and variance are used to compute the values $\hat{\mu}$ and $\hat{\phi}$ as

$$\hat{\mu} = m_1$$

$$\hat{\phi} = \frac{m_2 - m_1^2}{\hat{\mu}^{\hat{p}}}$$

Based on the relationship between the parameters of TWEEDIE and STWEEDIE distributions described in the section "Tweedie Distributions" on page 402, values of θ and λ are initialized as

$$\hat{\theta} = \hat{\phi}(\hat{p} - 1)\hat{\mu}^{p-1}$$
$$\hat{\lambda} = \frac{\hat{\mu}}{\hat{\theta}\hat{\alpha}}$$

WEIBULL The parameters are initialized by using the percentile matching method. Let $q1$ and $q3$ denote the estimates of the 25th and 75th percentiles, respectively. Using the formula for the CDF of Weibull distribution, they can be written as

$$1 - \exp(-(q1/\theta)^{\tau}) = 0.25$$
$$1 - \exp(-(q3/\theta)^{\tau}) = 0.75$$

Simplifying and solving these two equations yields the following initial values:

$$\hat{\theta} = \exp\left(\frac{r\log(q1) - \log(q3)}{r - 1}\right), \quad \hat{\tau} = \frac{\log(\log(4))}{\log(q3) - \log(\hat{\theta})}$$

where $r = \log(\log(4))/\log(\log(4/3))$. These initial values agree with those suggested in Klugman, Panjer, and Willmot (1998).

A summary of the initial values of all the parameters for all the predefined distributions is given in Table 14.3. The table also provides the names of the parameters to use in the INIT= option in the DIST statement if you want to provide a different initial value.

Table 14.3 Parameter Initialization for Predefined Distributions

Distribution	Parameter	Name for INIT option	Default Initial Value
BURR	θ	theta	$\sqrt{\frac{m_2 m_3}{2m_3 - 3m_1 m_2}}$
	α	alpha	$1 + \frac{m_3}{2m_3 - 3m_1 m_2}$
	γ	gamma	2
EXP	θ	theta	m_1
GAMMA	θ	theta	m_1/α
	α	alpha	$\frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
GPD	θ	theta	$m_1 m_2/(2(m_2 - m_1^2))$
	ξ	xi	$(m_2 - 2m_1^2)/(2(m_2 - m_1^2))$
IGAUSS	θ	theta	m_1
	α	alpha	$m_1^2/(m_2 - m_1^2)$
LOGN	μ	mu	$2\log(m1) - \log(m2)/2$
	σ	sigma	$\sqrt{\log(m2) - 2\log(m1)}$
PARETO	θ	theta	$m_1 m_2/(m_2 - 2m_1^2)$
	α	alpha	$2(m_2 - m_1^2)/(m_2 - 2m_1^2)$
TWEEDIE	μ	mu	m_1
	ϕ	phi	$(m_2 - m_1^2)/m_1^p$
	p	p	$(\alpha + 2)/(\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
STWEEDIE	θ	theta	$(m_2 - m_1^2)(p-1)/m_1$
	λ	lambda	$m_1^2/(\alpha(m_2 - m_1^2)(p-1))$
	p	p	$(\alpha + 2)/(\alpha + 1)$
			where $\alpha = \frac{3-d+\sqrt{(d-3)^2+24d}}{12d}$
WEIBULL	θ	theta	$\exp\left(\frac{r\log(q1) - \log(q3)}{r-1}\right)$
	τ	tau	$\log(\log(4))/(\log(q3) - \log(\hat{\theta}))$

Notes:

- m_k denotes the kth raw moment
- $d = \log(m_1) - (\sum \log(y_i))/n$
- $q1$ and $q3$ denote the 25th and 75th percentiles, respectively
- $r = \log(\log(4))/\log(\log(4/3))$

Censoring and Truncation

One of the key features of PROC HPSEVERITY is that it enables you to specify whether the severity event's magnitude is observable and if it is observable, then whether the exact value of the magnitude is known. If an event is unobservable when the magnitude is in certain intervals, then it is referred to as a truncation effect. If the exact magnitude of the event is not known, but it is known to have a value in a certain interval, then it is referred to as a censoring effect.

PROC HPSEVERITY allows a severity event to be subject to any combination of the following four censoring and truncation effects:

- Left-truncation: An event is said to be left-truncated if it is observed only when $Y > T^l$, where Y denotes the random variable for the magnitude and T^l denotes a random variable for the truncation threshold. You can specify left-truncation using the LEFTTRUNCATED= option in the LOSS statement.
- Right-truncation: An event is said to be right-truncated if it is observed only when $Y \leq T^r$, where Y denotes the random variable for the magnitude and T^r denotes a random variable for the truncation threshold. You can specify right-truncation using the RIGHTTRUNCATED= option in the LOSS statement.
- Left-censoring: An event is said to be left-censored if it is known that the magnitude is $Y \leq C^l$, but the exact value of Y is not known. C^l is a random variable for the censoring limit. You can specify left-censoring using the LEFTCENSORED= option in the LOSS statement.
- Right-censoring: An event is said to be right-censored if it is known that the magnitude is $Y > C^r$, but the exact value of Y is not known. C^r is a random variable for the censoring limit. You can specify right-censoring using the RIGHTCENSORED= option in the LOSS statement.

For each effect, you can specify a different threshold or limit for each observation or specify a single threshold or limit that applies to all the observations.

If all the four types of effects are present on an event, then the following relationship holds: $T^l < C^r \leq C^l \leq T^r$. PROC HPSEVERITY checks these relationships and write a warning to the SAS log if any is violated.

If the response variable is specified in the LOSS statement, then PROC HPSEVERITY also checks whether each observation satisfies the definitions of the specified censoring and truncation effects. If left-truncation is specified, then PROC HPSEVERITY ignores observations where $Y \leq T^l$, because such observations are not observable by definition. Similarly, if right-truncation is specified, then PROC HPSEVERITY ignores observations where $Y > T^r$. If left-censoring is specified, then PROC HPSEVERITY treats an observation with $Y > C^l$ as uncensored and ignores the value of C^l. The observations with $Y \leq C^l$ are considered as left-censored, and the value of Y is ignored. If right-censoring is specified, then PROC HPSEVERITY treats an observation with $Y \leq C^r$ as uncensored and ignores the value of C^r. The observations with $Y > C^r$ are considered as right-censored, and the value of Y is ignored. The specification of both left-censoring and right-censoring is referred to as interval-censoring. If $C^r < C^l$ is satisfied for an observation, then it is considered as interval-censored and the value of the response variable is ignored. If $C^r = C^l$ for an observation, then PROC HPSEVERITY assumes that observation to be uncensored. If all the observations in a data set are censored in some form, then the specification of the response variable in the LOSS statement is optional, because the actual value of the response variable is not required for the purposes of estimating a model.

Specification of censoring and truncation affects the likelihood of the data (see the section "Likelihood Function" on page 411) and how the empirical distribution function (EDF) is estimated (see the section "Empirical Distribution Function Estimation Methods" on page 418).

Parameter Estimation Method

PROC HPSEVERITY uses the maximum likelihood (ML) method to estimate the parameters of each model. A nonlinear optimization process is used to maximize the log of the likelihood function.

Likelihood Function

Let $f_\Theta(x)$ and $F_\Theta(x)$ denote the PDF and CDF, respectively, evaluated at x for a set of parameter values Θ. Let Y denote the random response variable, and let y denote its value recorded in an observation in the input data set. Let T^l and T^r denote the random variables for the left-truncation and right-truncation threshold, respectively, and let t^l and t^r denote their values for an observation, respectively. If there is no left-truncation, then $t^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t^l) = 0$. If there is no right-truncation, then $t^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t^r) = 1$. Let C^l and C^r denote the random variables for the left-censoring and right-censoring limit, respectively, and let c^l and c^r denote their values for an observation, respectively. If there is no left-censoring, then $c^l = \tau_h$; so $F(c^l) = 1$. If there is no right-censoring, then $c^r = \tau^l$; so $F(c^r) = 0$.

The set of input observations can be categorized into the following four subsets:

- E is the set of uncensored and untruncated observations. The likelihood of an observation in E is

$$l_E = \Pr(Y = y) = f_\Theta(y)$$

- E_t is the set of uncensored observations that are truncated. The likelihood of an observation in E_t is

$$l_{E_t} = \Pr(Y = y | t^l < Y \leq t^r) = \frac{f_\Theta(y)}{F_\Theta(t^r) - F_\Theta(t^l)}$$

- C is the set of censored observations that are not truncated. The likelihood of an observation C is

$$l_C = \Pr(c^r < Y \leq c^l) = F_\Theta(c^l) - F_\Theta(c^r)$$

- C_t is the set of censored observations that are truncated. The likelihood of an observation C_t is

$$l_{C_t} = \Pr(c^r < Y \leq c^l | t^l < Y \leq t^r) = \frac{F_\Theta(c^l) - F_\Theta(c^r)}{F_\Theta(t^r) - F_\Theta(t^l)}$$

Note that $(E \cup E_t) \cap (C \cup C_t) = \emptyset$. Also, the sets E_t and C_t are empty when no truncation is specified, and the sets C and C_t are empty when no censoring is specified.

Given this, the likelihood of the data L is as follows:

$$L = \prod_{E} f_\Theta(y) \prod_{E_t} \frac{f_\Theta(y)}{F_\Theta(t^r) - F_\Theta(t^l)} \prod_{C} F_\Theta(c^l) - F_\Theta(c^r) \prod_{C_t} \frac{F_\Theta(c^l) - F_\Theta(c^r)}{F_\Theta(t^r) - F_\Theta(t^l)}$$

The maximum likelihood procedure used by PROC HPSEVERITY finds an optimal set of parameter values $\hat{\Theta}$ that maximizes $\log(L)$ subject to the boundary constraints on parameter values. For a distribution *dist*, such boundary constraints can be specified by using the *dist*_LOWERBOUNDS and *dist*_UPPERBOUNDS subroutines. For more information, see the section "Defining a Severity Distribution Model with the FCMP Procedure" on page 424. Some aspects of the optimization process can be controlled by using the NLOPTIONS statement.

Estimating Covariance and Standard Errors

PROC HPSEVERITY computes an estimate of the covariance matrix of the parameters by using the asymptotic theory of the maximum likelihood estimators (MLE). If N denotes the number of observations used for estimating a parameter vector $\boldsymbol{\theta}$, then the theory states that as $N \rightarrow \infty$, the distribution of $\hat{\boldsymbol{\theta}}$, the estimate of $\boldsymbol{\theta}$, converges to a normal distribution with mean $\boldsymbol{\theta}$ and covariance $\hat{\mathbf{C}}$ such that $\mathbf{I}(\boldsymbol{\theta}) \cdot \hat{\mathbf{C}} \rightarrow 1$, where $\mathbf{I}(\boldsymbol{\theta}) = -E\left[\nabla^2 \log(L(\boldsymbol{\theta}))\right]$ is the information matrix for the likelihood of the data, $L(\boldsymbol{\theta})$. The covariance estimate is obtained by using the inverse of the information matrix.

In particular, if $\mathbf{G} = \nabla^2 \log(L(\boldsymbol{\theta}))$ denotes the Hessian matrix of the log likelihood, then the covariance estimate is computed as

$$\hat{\mathbf{C}} = \frac{N}{d}\mathbf{G}^{-1}$$

where d is a denominator determined by the VARDEF= option. If VARDEF=N, then $d = N$, which yields the asymptotic covariance estimate. If VARDEF=DF, then $d = N - k$, where k is number of parameters (the model's degrees of freedom). The VARDEF=DF option is the default, because it attempts to correct the potential bias introduced by the finite sample.

The standard error s_i of the parameter θ_i is computed as the square root of the ith diagonal element of the estimated covariance matrix; that is, $s_i = \sqrt{\hat{C}_{ii}}$.

Covariance and standard error estimates might not be available if the Hessian matrix is found to be singular at the end of the optimization process. This can especially happen if the optimization process stops without converging.

Details of Optimization Algorithms

Overview

There are several optimization techniques available. You can choose a particular optimizer with the TECH=*name* option in the PROC statement or NLOPTIONS statement.

Table 14.4 Optimization Techniques

Algorithm	**TECH=**
Trust region Method	TRUREG
Newton-Raphson method with line search	NEWRAP
Newton-Raphson method with ridging	NRRIDG
Quasi-Newton methods (DBFGS, DDFP, BFGS, DFP)	QUANEW
Double-dogleg method (DBFGS, DDFP)	DBLDOG
Conjugate gradient methods (PB, FR, PR, CD)	CONGRA
Nelder-Mead simplex method	NMSIMP

No algorithm for optimizing general nonlinear functions exists that always finds the global optimum for a general nonlinear minimization problem in a reasonable amount of time. Since no single optimization technique is invariably superior to others, NLO provides a variety of optimization techniques that work well in various circumstances. However, you can devise problems for which none of the techniques in NLO can find the correct solution. Moreover, nonlinear optimization can be computationally expensive in terms of time and memory, so you must be careful when matching an algorithm to a problem.

All optimization techniques in NLO use $O(n^2)$ memory except the conjugate gradient methods, which use only $O(n)$ of memory and are designed to optimize problems with many parameters. These iterative techniques require repeated computation of the following:

- the function value (optimization criterion)
- the gradient vector (first-order partial derivatives)
- for some techniques, the (approximate) Hessian matrix (second-order partial derivatives)

However, since each of the optimizers requires different derivatives, some computational efficiencies can be gained. Table 14.5 shows which derivatives are required for each optimization technique. (FOD means that first-order derivatives or the gradient is computed; SOD means that second-order derivatives or the Hessian is computed.)

Table 14.5 Optimization Computations

Algorithm	FOD	SOD
TRUREG	x	x
NEWRAP	x	x
NRRIDG	x	x
QUANEW	x	-
DBLDOG	x	-
CONGRA	x	-
NMSIMP	-	-

Each optimization method uses one or more convergence criteria that determine when it has converged. The various termination criteria are listed and described in the previous section. An algorithm is considered to have converged when any one of the convergence criteria is satisfied. For example, under the default settings, the QUANEW algorithm converges if ABSGCONV $< 1\text{E–}5$, FCONV $< 10^{-\text{FDIGITS}}$, or GCONV $<$ 1E–8.

Choosing an Optimization Algorithm

The factors for choosing a particular optimization technique for a particular problem are complex and might involve trial and error.

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

A few general remarks about the various optimization techniques follow:

- The second-derivative methods TRUREG, NEWRAP, and NRRIDG are best for small problems where the Hessian matrix is not expensive to compute. Sometimes the NRRIDG algorithm can be faster than the TRUREG algorithm, but TRUREG can be more stable. The NRRIDG algorithm requires only one matrix with $n(n+1)/2$ double words; TRUREG and NEWRAP require two such matrices.
- The first-derivative methods QUANEW and DBLDOG are best for medium-sized problems where the objective function and the gradient are much faster to evaluate than the Hessian. The QUANEW and DBLDOG algorithms generally require more iterations than TRUREG, NRRIDG, and NEWRAP, but each iteration can be much faster. The QUANEW and DBLDOG algorithms require only the gradient to update an approximate Hessian, and they require slightly less memory than TRUREG or NEWRAP (essentially one matrix with $n(n+1)/2$ double words). QUANEW is the default optimization method.
- The first-derivative method CONGRA is best for large problems where the objective function and the gradient can be computed much faster than the Hessian and where too much memory is required to

store the (approximate) Hessian. The CONGRA algorithm generally requires more iterations than QUANEW or DBLDOG, but each iteration can be much faster. Since CONGRA requires only a factor of n double-word memory, many large applications can be solved only by CONGRA.

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

Parameter Initialization

PROC HPSEVERITY enables you to initialize parameters of a model in different ways. A model can have two kinds of parameters: distribution parameters and regression parameters.

The distribution parameters can be initialized by using one of the following three methods:

INIT= option	You can use the INIT= option in the DIST statement.
INEST= data set	You can use the INEST= data set.
PARMINIT subroutine	You can define a *dist*_PARMINIT subroutine in the distribution model. For more information, see the section "Defining a Severity Distribution Model with the FCMP Procedure" on page 424.

Note that only one of the initialization methods is used. You cannot combine them. They are used in the following order:

- The method that uses the INIT= option takes the highest precedence. If you use the INIT= option to provide an initial value for at least one parameter, then other initialization methods (INEST= and PARMINIT) are not used. If you specify initial values for some but not all the parameters by using the INIT= option, then the uninitialized parameters are initialized to the default value of 0.001.

  If this option is used when regression effects are specified, then the value of the first distribution parameter must be related to the initial value for the *base* value of the scale or log-transformed scale parameter. For more information, see the section "Estimating Regression Effects" on page 416.

- The method that uses the INEST= data set takes the second precedence. If there is a nonmissing value specified for even one distribution parameter, then the PARMINIT method is not used and any uninitialized parameters are initialized to the default value of 0.001.

- If none of the distribution parameters are initialized by using the INIT= option or the INEST= data set, but the distribution model defines a PARMINIT subroutine, then PROC HPSEVERITY invokes that subroutine with appropriate inputs to initialize the parameters. If the PARMINIT subroutine returns missing values for some parameters, then those parameters are initialized to the default value of 0.001.

- If none of the initialization methods are used, each distribution parameter is initialized to the default value of 0.001.

For more information about regression models and initialization of regression parameters, see the section "Estimating Regression Effects" on page 416.

PARMINIT-Based Parameter Initialization Method and Distributed Data

If you have specified a MPP model of execution for the procedure, then the input data is distributed across the computational nodes. For more information about the distributed computing model, see the section "Distributed and Multithreaded Computation" on page 421. If the PARMINIT subroutine is used for initializing the distribution parameters, then PROC HPSEVERITY invokes that subroutine on each computational node with the data that are local to that node. The EDF estimates that are supplied to the PARMINIT subroutine are also computed using the local data. The initial values of the parameters that are supplied to the optimizer are the average of the local estimates computed on each node. This approach works well if the data are distributed randomly across nodes. If you distribute the data on the appliance before you run the procedure (alongside-the-database model), then you should try to make the distribution as random as possible in order to increase the chances of computing good initial values. If you specify a data set that is not distributed before you run the procedure, then PROC HPSEVERITY distributes the data for you in a round-robin manner. That is, it sends the first observation to the first node, second observation to the second node, and so forth. If the order of observations is random, then this method ensures random distribution of data across the computational nodes.

Estimating Regression Effects

The HPSEVERITY procedure enables you to estimate the effects of regressor (exogenous) variables while fitting a distribution if the distribution has a scale parameter or a log-transformed scale parameter.

Let x_j $(j = 1, \dots, k)$ denote the k regressor variables. Let β_j denote the regression parameter that corresponds to the regressor x_j. If regression effects are not specified, then the model for the response variable Y is of the form

$$Y \sim \mathcal{F}(\Theta)$$

where $\mathcal{F}$ is the distribution of Y with parameters Θ. This model is typically referred to as the error model. The regression effects are modeled by extending the error model to the following form:

$$Y \sim \exp(\sum_{j=1}^{k} \beta_j x_j) \cdot \mathcal{F}(\Theta)$$

Under this model, the distribution of Y is valid and belongs to the same parametric family as $\mathcal{F}$ if and only if $\mathcal{F}$ has a scale parameter. Let θ denote the scale parameter and Ω denote the set of nonscale distribution parameters of $\mathcal{F}$. Then the model can be rewritten as

$$Y \sim \mathcal{F}(\theta, \Omega)$$

such that θ is affected by the regressors as

$$\theta = \theta_0 \cdot \exp(\sum_{j=1}^{k} \beta_j x_j)$$

where θ_0 is the *base* value of the scale parameter. Thus, the regression model consists of the following parameters: θ_0, Ω, and $\beta_j (j = 1, \dots, k)$.

Given this form of the model, distributions without a scale parameter cannot be considered when regression effects are to be modeled. If a distribution does not have a direct scale parameter, then PROC HPSEVERITY accepts it only if it has a log-transformed scale parameter — that is, if it has a parameter $p = \log(\theta)$.

Parameter Initialization for Regression Models

The regression parameters are initialized either by using the values you specify or by the default method.

- If you provide initial values for the regression parameters, then you must provide valid, nonmissing initial values for θ_0 and β_j parameters for all j.

  You can specify the initial value for θ_0 using either the INEST= data set or the INIT= option in the DIST statement. If the distribution has a direct scale parameter (no transformation), then the initial value for the first parameter of the distribution is used as an initial value for θ_0. If the distribution has a log-transformed scale parameter, then the initial value for the first parameter of the distribution is used as an initial value for $\log(\theta_0)$.

  You can use only the INEST= data set to specify the initial values for β_j. The INEST= data set must contain nonmissing initial values for all the regressors specified in the SCALEMODEL statement. The only missing value allowed is the special missing value .R, which indicates that the regressor is linearly dependent on other regressors. If you specify .R for a regressor for one distribution, you must specify it so for all the distributions.

- If you do not specify valid initial values for θ_0 or β_j parameters for all j, then PROC HPSEVERITY initializes those parameters using the following method:

  Let a random variable Y be distributed as $\mathcal{F}(\theta, \Omega)$, where θ is the scale parameter. By the definition of the scale parameter, a random variable $W = Y/\theta$ is distributed as $\mathcal{G}(\Omega)$ such that $\mathcal{G}(\Omega) = \mathcal{F}(1, \Omega)$. Given a random error term e that is generated from a distribution $\mathcal{G}(\Omega)$, a value y from the distribution of Y can be generated as

  $$y = \theta \cdot e$$

  Taking the logarithm of both sides and using the relationship of θ with the regressors yields:

  $$\log(y) = \log(\theta_0) + \sum_{j=1}^{k} \beta_j x_j + \log(e)$$

  PROC HPSEVERITY makes use of the preceding relationship to initialize parameters of a regression model with distribution *dist* as follows:

  1. The following linear regression problem is solved to obtain initial estimates of β_0 and β_j:

     $$\log(y) = \beta_0 + \sum_{j=1}^{k} \beta_j x_j$$

     The estimates of $\beta_j (j = 1, \ldots, k)$ in the solution of this regression problem are used to initialize the respective regression parameters of the model. The estimate of β_0 is later used to initialize the value of θ_0.

The results of this regression are also used to detect whether any regressors are linearly dependent on the other regressors. If any such regressors are found, then a warning is written to the SAS log and the corresponding regressor is eliminated from further analysis. The estimates for linearly dependent regressors are denoted by a special missing value of .R in the OUTEST= data set and in any displayed output.

2. Let s_0 denote the initial value of the scale parameter.

   If the distribution model of *dist* does not contain the *dist*_PARMINIT subroutine, then s_0 and all the nonscale distribution parameters are initialized to the default value of 0.001.

   However, it is strongly recommended that each distribution's model contain the *dist*_PARMINIT subroutine. For more information, see the section "Defining a Severity Distribution Model with the FCMP Procedure" on page 424. If that subroutine is defined, then s_0 is initialized as follows: Each input value y_i of the response variable is transformed to its scale-normalized version w_i as

$$w_i = \frac{y_i}{\exp(\beta_0 + \sum_{j=1}^{k} \beta_j x_{ij})}$$

   where x_{ij} denotes the value of jth regressor in the ith input observation. These w_i values are used to compute the input arguments for the *dist*_PARMINIT subroutine. The values that are computed by the subroutine for nonscale parameters are used as their respective initial values. If the distribution has an untransformed scale parameter, then s_0 is set to the value of the scale parameter that is computed by the subroutine. If the distribution has a log-transformed scale parameter P, then s_0 is computed as $s_0 = \exp(l_0)$, where l_0 is the value of P computed by the subroutine.

3. The value of θ_0 is initialized as

$$\theta_0 = s_0 \cdot \exp(\beta_0)$$

Reporting Estimates of Regression Parameters

When you request estimates to be written to the output (either ODS displayed output or in the OUTEST= data set), the estimate of the base value of the first distribution parameter is reported. If the first parameter is the log-transformed scale parameter, then the estimate of $\log(\theta_0)$ is reported; otherwise, the estimate of θ_0 is reported. The transform of the first parameter of a distribution *dist* is controlled by the *dist*_SCALETRANSFORM function that is defined for it.

Empirical Distribution Function Estimation Methods

The empirical distribution function (EDF) is a nonparametric estimate of the cumulative distribution function (CDF) of the distribution. PROC HPSEVERITY computes EDF estimates when it needs to invoke a distribution's PARMINIT subroutine in order to initialize distribution parameters.

This section describes the methods that are used for computing EDF estimates.

Notation

Let there be a set of N observations, each containing a quintuplet of values $(y_i, t_i^l, t_i^r, c_i^l, c_i^r), i = 1, \ldots, N$, where y_i is the value of the response variable, t_i^l is the value of the left-truncation threshold, t_i^r is the value of the right-truncation threshold, c_i^l is the value of the left-censoring limit, and c_i^r is the value of the right-censoring limit.

If an observation is not left-truncated, then $t_i^l = \tau^l$, where τ^l is the smallest value in the support of the distribution; so $F(t_i^l) = 0$. If an observation is not right-truncated, then $t_i^r = \tau_h$, where τ_h is the largest value in the support of the distribution; so $F(t_i^r) = 1$. If an observation is not left-censored, then $c_i^l = \tau_h$; so $F(c_i^l) = 1$. If an observation is not right-censored, then $c_i^r = \tau^l$; so $F(c_i^r) = 0$.

An indicator function $I[e]$ takes a value of 1 or 0 if the expression e is true or false, respectively.

Estimation Methods

The methods are described assuming that all observations either uncensored or right-censored; that is, each observation is of the form $(y_i, t_i^l, t_i^r, \tau_h, \tau^l)$ or $(y_i, t_i^l, t_i^r, \tau_h, c_i^r)$.

If an observation is left-censored, then by the requirement imposed by PROC HPSEVERITY, it must be interval-censored. Each such observation is converted to an uncensored observation $(y_i^u, t_i^l, t_i^r, \tau_h, \tau^l)$, where $y_i^u = (c_i^l + c_i^r)/2$. With this transformation, each observation is either uncensored or right-censored and the methods described here are applicable.

Further, a set of uncensored or right-censored observations can be converted to a set of observations of the form $(y_i, t_i^l, t_i^r, \delta_i)$, where δ_i is the indicator of right-censoring. $\delta_i = 0$ indicates a right-censored observation, in which case y_i is assumed to record the right-censoring limit c_i^r. $\delta_i = 1$ indicates an uncensored observation, and y_i records the exact observed value. In other words, $\delta_i = I[Y \leq C^r]$ and $y_i = \min(y_i, c_i^r)$.

Given this notation, the EDF is estimated as

$$F_n(y) = \begin{cases} 0 & \text{if } y < y^{(1)} \\ \hat{F}_n(y^{(k)}) & \text{if } y^{(k)} \leq y < y^{(k+1)}, k = 1, \ldots, N-1 \\ \hat{F}_n(y^{(N)}) & \text{if } y^{(N)} \leq y \end{cases}$$

where $y^{(k)}$ denotes the kth order statistic of the set $\{y_i\}$ and $\hat{F}_n(y^{(k)})$ is the estimate computed at that value. The definition of $\hat{F}_n$ depends on the estimation method as follows:

STANDARD This method is chosen when no censoring or truncation information is specified. It is the standard way of computing EDF. The EDF estimate at observation i is computed as follows:

$$\hat{F}_n(y_i) = \frac{1}{N} \sum_{j=1}^{N} I[y_j \leq y_i]$$

KAPLANMEIER This method is chosen when at least one form of censoring or truncation is specified. The Kaplan-Meier (KM) estimator, also known as the product-limit estimator, was first introduced by Kaplan and Meier (1958) for censored data. Lynden-Bell (1971) derived a similar estimator for left-truncated data. PROC HPSEVERITY uses the definition that combines both censoring and truncation information (Klein and Moeschberger 1997, Lai and Ying 1991).

The EDF estimate at observation i is computed as

$$\hat{F}_n(y_i) = 1 - \prod_{\tau \leq y_i} \left(1 - \frac{n_\tau}{R_n(\tau)}\right)$$

where n_τ and $R_n(\tau)$ are defined as follows:

- $n_\tau = \sum_{k=1}^{N} I[y_k = \tau \text{ and } \tau \leq t_k^r \text{ and } \delta_k = 1]$, which is the number of uncensored observations ($\delta_k = 1$) for which the response variable value is equal to τ and τ is observable according to the right-truncation threshold of that observation ($\tau \leq t_k^r$).
- $R_n(\tau) = \sum_{k=1}^{N} I[y_k \geq \tau > t_k^l]$, which is the size (cardinality) of the risk set at τ. The term *risk set* has its origins in survival analysis; it contains the events that are at risk of failure at a given time, τ. In other words, it contains the events that have survived up to time τ and might fail at or after τ. For PROC HPSEVERITY, *time* is equivalent to the magnitude of the event and *failure* is equivalent to an uncensored and observable event, where observable means it satisfies the truncation thresholds.

Supplying EDF Estimates to Functions and Subroutines

The parameter initialization subroutines in distribution models and some predefined utility functions require EDF estimates. For more information, see the sections "Defining a Severity Distribution Model with the FCMP Procedure" on page 424 and "Predefined Utility Functions" on page 436.

PROC HPSEVERITY supplies the EDF estimates to these subroutines and functions by using two arrays, x and F, the dimension of each array, and a type of the EDF estimates. The type identifies how the EDF estimates are computed and stored. They are as follows:

Type 1	specifies that EDF estimates are computed using the STANDARD method; that is, the data used for estimation are neither censored nor truncated.
Type 2	specifies that EDF estimates are computed using the KAPLANMEIER method; that is, the data used for estimation are subject to at least one form of truncation or censoring.

For each type, the EDF estimates are stored in arrays x and F of dimension N such that the following holds:

$$F_n(y) = \begin{cases} 0 & \text{if } y < x[1] \\ F[k] & \text{if } x[k] \leq y < x[k+1], k = 1, \ldots, N-1 \\ F[N] & \text{if } x[N] \leq y \end{cases}$$

where $[k]$ denotes kth element of the array ([1] denotes the first element of the array).

Statistics of Fit

PROC HPSEVERITY computes and reports four likelihood-based statistics: Neg2LogLike, AIC, AICC, and BIC. The following subsections provide definitions of each.

Likelihood-Based Statistics of Fit

Let $y_i, i = 1, \ldots, N$ denote the response variable values. Let L be the likelihood as defined in the section "Likelihood Function" on page 411. Let p denote the number of model parameters estimated. Note that $p = p_d + (k - k_r)$, where p_d is the number of distribution parameters, k is the number of regressors specified in the SCALEMODEL statement, and k_r is the number of regressors found to be linearly dependent (redundant) on other regressors. Given this notation, the likelihood-based statistics are defined as follows:

Neg2LogLike
: The log likelihood is reported as

$$\text{Neg2LogLike} = -2\log(L)$$

The multiplying factor -2 makes it easy to compare it to the other likelihood-based statistics. A model with a smaller value of Neg2LogLike is deemed better.

AIC
: The Akaike's information criterion (AIC) is defined as

$$\text{AIC} = -2\log(L) + 2p$$

A model with a smaller value of AIC is deemed better.

AICC
: The corrected Akaike's information criterion (AICC) is defined as

$$\text{AICC} = -2\log(L) + \frac{2Np}{N - p - 1}$$

A model with a smaller value of AICC is deemed better. It corrects the finite-sample bias that AIC has when N is small compared to p. AICC is related to AIC as

$$\text{AICC} = \text{AIC} + \frac{2p(p+1)}{N - p - 1}$$

As N becomes large compared to p, AICC converges to AIC. AICC is usually recommended over AIC as a model selection criterion.

BIC
: The Schwarz Bayesian information criterion (BIC) is defined as

$$\text{BIC} = -2\log(L) + p\log(N)$$

A model with a smaller value of BIC is deemed better.

Distributed and Multithreaded Computation

PROC HPSEVERITY makes an attempt to use all the computational resources that you specify in the PERFORMANCE statement in order to complete the assigned tasks as fast as possible. This section describes the distributed and multithreading computing methods that PROC HPSEVERITY uses.

Distributed Computing

Distributed computing refers to the organization of computation work into multiple tasks that are processed on different nodes, where a node refers to one of the machines that constitute the grid. The number of nodes used by PROC HPSEVERITY is determined by the execution model for massively parallel processing (MPP). If you specify the client-data (or local-data) model of execution, then the number of nodes is determined by the NODES= option in the PERFORMANCE statement. If you are using the alongside-database model of execution, then the number of nodes is determined internally by PROC HPSEVERITY using the information associated with the DATA= data set and the grid information that you specify either in the PERFORMANCE statement or in the grid environment variables. See the section "SMP and MPP Modes" on page 10 for a detailed description of MPP execution models.

In the client-data model, PROC HPSEVERITY distributes the input data across the number of nodes that you specify. It uses the round-robin method; that is, it sends the first observation to the first node, second observation to the second node, and so forth.

In the alongside-the-database model, PROC HPSEVERITY uses the existing distributed organization of the data, and knows about the number of nodes. So, you do not need to specify the NODES= option.

The number of nodes that are used for distributed computing is displayed in the "Performance Information" table, which is part of the default output.

Multithreading

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

The number of threads spawned by the HPSEVERITY procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statements, the HPSEVERITY procedure schedules threads as if it executes on a system with four CPUs, regardless of the actual CPU count.

  ```
  options cpucount=4;
  ```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This specification overrides the system option. Specify NTHREADS=1 to force single-threaded execution.

- You can specify the CPUCOUNT=*n* option in the PERFORMANCE statement, which overrides the CPUCOUNT= system option and instructs the HPSEVERITY procedure to operate as if it executed on a machine with *n* CPUs.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output.

Combining the Power of Distributed and Multithreading Computing

The HPSEVERITY procedure combines the powers of distributed and multithreading paradigms by using a data-parallel model. In particular, the distributed tasks are defined by dividing the data among multiple nodes, and within one node, the multithreading tasks are defined by further dividing the local data among the threads. For example, if the input data set has 10,000 observations and you are running on a grid with 5 nodes, then each node processes 2,000 observations (this assumes that if you have specified an alongside-the-database model, then you have equally and randomly divided the input data among the nodes). Further, if each node has eight CPUs, then 250 observations are associated with each thread within the node. All computations that require access to the data are then distributed and multithreaded.

Note that in the SMP mode (see the section "SMP and MPP Modes" on page 10), only multithreading is available.

When you specify more than one candidate distribution model, for some tasks PROC HPSEVERITY exploits the independence among models by processing multiple models in parallel on a single node such that each model is assigned to one of the threads executing in parallel. When a thread finishes processing the assigned model, it starts processing the next unprocessed model, if one exists.

The computations that take advantage of the distributed and multithreaded model include the following:

- Validation and preparation of data: In this stage, the observations in the input data set are validated and transformed, if necessary. The summary statistics of the data are prepared. Since each observation is independent, the computations can be distributed among nodes and threads-within-nodes without significant communication overhead.

- Initialization of distribution parameters: In this stage, the parallelism is achieved by initializing multiple models in parallel. The only computational step that is not fully parallelized in this release is the step of computing empirical distribution function (EDF) estimates, which are required when PROC HPSEVERITY needs to invoke a distribution's PARMINIT subroutine to initialize distribution parameters. The EDF estimation step is not amenable to full-fledged parallelism because it requires sequential access to sorted data, especially when the loss variable is modified by truncation effects. When the data are distributed across nodes, the EDF computations take place on local data and the PARMINIT function is invoked on the local data using the local EDF estimates. The initial values that are supplied to the nonlinear optimizer are computed by averaging the local estimates of the distribution parameters returned by the PARMINIT functions on each node.

- Initialization of regression parameters (if SCALEMODEL statement is specified): In this stage, if you have not specified initial values for the regression parameters by using the INEST= data set, then PROC HPSEVERITY initializes those parameters by solving a linear regression problem $\log(y) = \beta_0 + \sum_{i=1}^{k} \beta_j x_j$. For more information, see the section "Parameter Initialization for Regression Models" on page 417. The most computationally intensive step is the formation of the crossproducts matrix. PROC HPSEVERITY exploits the parallelism by observing the fact that the contribution to the crossproducts matrix due to one observation is independent from the contribution due to another observation. Each node computes the contribution of its local data to each entry of the crossproducts matrix. Within each node, each thread computes the contribution of its chunk of data to each entry of the crossproducts matrix. On each node, the contributions from all the threads are added up to form the contribution due to all of the local data. The partial crossproducts matrices are then gathered from all nodes on a central node, which sums them up to form the final crossproducts matrix.

- Optimization: In this stage, the nonlinear optimizer iterates over the parameter space in search of the optimal set of parameters. In each iteration, it evaluates the objective function along with the gradient and Hessian of the objective function, if needed by the optimization method. Within one iteration, for the current estimates of the parameters, each observation's contribution to the objective function, gradient, and Hessian is independent of another observation. This enables PROC HPSEVERITY to fully exploit the distributed and multithreaded paradigms to efficiently parallelize each iteration of the algorithm.

Defining a Severity Distribution Model with the FCMP Procedure

A *severity distribution model* consists of a set of functions and subroutines that are defined using the FCMP procedure. The FCMP procedure is part of Base SAS software. Each function or subroutine must be named as <*distribution-name*>_<*keyword*>, where *distribution-name* is the identifying short name of the distribution and *keyword* identifies one of the functions or subroutines. The total length of the name should not exceed 32. Each function or subroutine must have a specific signature, which consists of the number of arguments, sequence and types of arguments, and return value type. The summary of all the recognized function and subroutine names and their expected behavior is given in Table 14.6.

Consider the following points when you define a distribution model:

- When you define a function or subroutine requiring parameter arguments, the names and order of those arguments must be the same. Arguments other than the parameter arguments can have any name, but they must satisfy the requirements on their type and order.

- When the HPSEVERITY procedure invokes any function or subroutine, it provides the necessary input values according to the specified signature, and expects the function or subroutine to prepare the output and return it according to the specification of the return values in the signature.

- You can typically use most of the SAS programming statements and SAS functions that you can use in a DATA step for defining the FCMP functions and subroutines. However, there are a few differences in the capabilities of the DATA step and the FCMP procedure. To learn more, see the documentation of the FCMP procedure in the *Base SAS Procedures Guide*.

- You must specify either the PDF or the LOGPDF function. Similarly, you must specify either the CDF or the LOGCDF function. All other functions are optional, except when necessary for correct definition of the distribution. It is strongly recommended that you define the PARMINIT subroutine to provide a good set of initial values for the parameters. The information provided by PROC HPSEVERITY to the PARMINIT subroutine enables you to use popular initialization approaches based on the method of moments and the method of percentile matching, but you can implement any algorithm to initialize the parameters by using the values of the response variable and the estimate of its empirical distribution function.

- The LOWERBOUNDS subroutines should be defined if the lower bound on at least one distribution parameter is different from the default lower bound of 0. If you define a LOWERBOUNDS subroutine but do not set a lower bound for some parameter inside the subroutine, then that parameter is assumed to have no lower bound (or a lower bound of $-\infty$). Hence, it is recommended that you explicitly return the lower bound for each parameter when you define the LOWERBOUNDS subroutine.

- The UPPERBOUNDS subroutines should be defined if the upper bound on at least one distribution parameter is different from the default upper bound of ∞. If you define an UPPERBOUNDS subroutine but do not set an upper bound for some parameter inside the subroutine, then that parameter is assumed to have no upper bound (or a upper bound of ∞). Hence, it is recommended that you explicitly return the upper bound for each parameter when you define the UPPERBOUNDS subroutine.

- If you want to use the distribution in a model with regression effects, then make sure that the first parameter of the distribution is the scale parameter itself or a log-transformed scale parameter. If the first parameter is a log-transformed scale parameter, then you must define the SCALETRANSFORM function.

- In general, it is not necessary to define the gradient and Hessian functions, because the HPSEVERITY procedure uses an internal system to evaluate the required derivatives. The internal system typically computes the derivatives analytically. But it might not be able to do so if your function definitions use other functions that it cannot differentiate analytically. In such cases, derivatives are approximated using a finite difference method and a note is written to the SAS log to indicate the components that are differentiated using such approximations. PROC HPSEVERITY does reasonably well with these finite difference approximations. But, if you know of a way to compute the derivatives of such components analytically, then you should define the gradient and Hessian functions.

In order to use your distribution with PROC HPSEVERITY, you need to record the FCMP library that contains the functions and subroutines for your distribution and other FCMP libraries that contain FCMP functions or subroutines used within your distribution's functions and subroutines. Specify all those libraries in the CMPLIB= system option by using the OPTIONS global statement. For more information about the OPTIONS statement, see *SAS Statements: Reference*. For more information about the CMPLIB= system option, see *SAS System Options: Reference*.

Each predefined distribution mentioned in the section "Predefined Distributions" on page 400 has a distribution model associated with it. The functions and subroutines of all those models are available in the Sashelp.Svrtdist library. The order of the parameters in the signatures of the functions and subroutines is the same as listed in Table 14.2. You do not need to use the CMPLIB= option in order to use the predefined distributions with PROC HPSEVERITY. However, if you need to use the functions or subroutines of the predefined distributions in SAS statements other than the PROC HPSEVERITY step (such as in a DATA step), then specify the Sashelp.Svrtdist library in the CMPLIB= system option by using the OPTIONS global statement prior to using them.

Table 14.6 shows functions and subroutines that define a distribution model, and subsections after the table provide more detail. The functions are listed in alphabetical order of the keyword suffix.

Table 14.6 List of Functions and Subroutines That Define a Distribution Model

Name	Type	Required	Expected to Return
*dist*_CDF	Function	YES[1]	Cumulative distribution function value
*dist*_CDFGRADIENT	Subroutine	NO	Gradient of the CDF
*dist*_CDFHESSIAN	Subroutine	NO	Hessian of the CDF
*dist*_CONSTANTPARM	Subroutine	NO	Constant parameters
*dist*_DESCRIPTION	Function	NO	Description of the distribution
*dist*_LOGCDF	Function	YES[1]	Log of cumulative distribution function value
*dist*_LOGCDFGRADIENT	Subroutine	NO	Gradient of the LOGCDF
*dist*_LOGCDFHESSIAN	Subroutine	NO	Hessian of the LOGCDF
*dist*_LOGPDF	Function	YES[2]	Log of probability density function value
*dist*_LOGPDFGRADIENT	Subroutine	NO	Gradient of the LOGPDF
*dist*_LOGPDFHESSIAN	Subroutine	NO	Hessian of the LOGPDF
*dist*_LOGSDF	Function	NO	Log of survival function value
*dist*_LOGSDFGRADIENT	Subroutine	NO	Gradient of the LOGSDF
*dist*_LOGSDFHESSIAN	Subroutine	NO	Hessian of the LOGSDF
*dist*_LOWERBOUNDS	Subroutine	NO	Lower bounds on parameters
*dist*_PARMINIT	Subroutine	NO	Initial values for parameters
*dist*_PDF	Function	YES[2]	Probability density function value
*dist*_PDFGRADIENT	Subroutine	NO	Gradient of the PDF
*dist*_PDFHESSIAN	Subroutine	NO	Hessian of the PDF
*dist*_SCALETRANSFORM	Function	NO	Type of relationship between the first distribution parameter and the scale parameter
*dist*_SDF	Function	NO	Survival function value
*dist*_SDFGRADIENT	Subroutine	NO	Gradient of the SDF
*dist*_SDFHESSIAN	Subroutine	NO	Hessian of the SDF
*dist*_UPPERBOUNDS	Subroutine	NO	Upper bounds on parameters

Notes:
1. Either the *dist*_CDF or the *dist*_LOGCDF function must be defined.
2. Either the *dist*_PDF or the *dist*_LOGPDF function must be defined.

The signature syntax and semantics of each function or subroutine are as follows:

*dist*_**CDF**

defines a function that returns the value of the cumulative distribution function (CDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type*: Function
- *Required*: YES
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the CDF value should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
	…..
pm	Numeric value of the mth parameter

- *Return value*: Numeric value that contains the CDF value $F(x; p_1, p_2, \ldots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \ldots, p_m) = F(\frac{x}{p_1}; 1, p_2, \ldots, p_m)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$F(x; p_1, p_2, \ldots, p_m) = F(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m)$$

Here is a sample structure of the function for a distribution named ‘FOO’:

```
function FOO_CDF(x, P1, P2);
    /* Code to compute CDF by using x, P1, and P2 */

    F = <computed CDF>;
    return (F);
endsub;
```

*dist*_CONSTANTPARM

defines a subroutine that specifies constant parameters. A parameter is *constant* if it is required for defining a distribution but is not subject to optimization in PROC HPSEVERITY. Constant parameters are required to be part of the model in order to compute the PDF or the CDF of the distribution. Typically, values of these parameters are known a priori or estimated using some means other than the maximum likelihood method used by PROC HPSEVERITY. You can estimate them inside the *dist*_PARMINIT subroutine. Once initialized, the parameters remain constant in the context of PROC HPSEVERITY; that is, they retain their initial value. PROC HPSEVERITY estimates only the nonconstant parameters.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: k, where k is the number of constant parameters

- *Sequence and type of arguments*:

constant parameter 1	Name of the first constant parameter
…..	
constant parameter k	Name of the kth constant parameter

- *Return value*: None

Here is a sample structure of the subroutine for a distribution named 'FOO' that has P3 and P5 as its constant parameters, assuming that distribution has at least three parameters:

```
subroutine FOO_CONSTANTPARM(p5, p3);
endsub;
```

Note the following points when you specify the constant parameters:

- At least one distribution parameter must be free to be optimized; that is, if a distribution has total m parameters, then k must be strictly less than m.
- If you want to use this distribution for modeling regression effects, then the first parameter must not be a constant parameter.
- The order of arguments in the signature of this subroutine does not matter as long as each argument's name matches the name of one of the parameters that are defined in the signature of the *dist*_PDF function.
- The constant parameters must be specified in signatures of all the functions and subroutines that accept distribution parameters as their arguments.
- You must provide a nonmissing initial value for each constant parameter by using one of the supported parameter initialization methods.

***dist*_DESCRIPTION**

defines a function that returns a description of the distribution.

- *Type*: Function
- *Required*: NO
- *Number of arguments*: None
- *Sequence and type of arguments*: Not applicable
- *Return value*: Character value containing a description of the distribution

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_DESCRIPTION() $48;
    length desc $48;
    desc = "A model for a continuous distribution named foo";
    return (desc);
endsub;
```

There is no restriction on the length of the description (the length of 48 used in the previous example is for illustration purposes only). However, if the length is greater than 256, then only the first 256 characters appear in the displayed output and in the _DESCRIPTION_ variable of the OUTMODELINFO= data set. Hence, the recommended length of the description is less than or equal to 256.

*dist*_**LOG***core*

defines a function that returns the natural logarithm of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, or SDF.

- *Type*: Function
- *Required*: YES only if *core* is PDF or CDF and you have not defined that *core* function; otherwise, NO
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

  x   Numeric value of the random variable at which the natural logarithm of the *core* function should be evaluated

  p1   Numeric value of the first parameter

  p2   Numeric value of the second parameter

  . . . ..

  pm Numeric value of the mth parameter

- *Return value*: Numeric value that contains the natural logarithm of the *core* function

Here is a sample structure of the function for the core function PDF of a distribution named 'FOO':

```
function FOO_LOGPDF(x, P1, P2);
    /* Code to compute LOGPDF by using x, P1, and P2 */

    l = <computed LOGPDF>;
    return (l);
endsub;
```

*dist*_**LOWERBOUNDS**

defines a subroutine that returns lower bounds for the parameters of the distribution. If this subroutine is not defined for a given distribution, then the HPSEVERITY procedure assumes a lower bound of 0 for each parameter. If a lower bound of l_i is returned for a parameter p_i, then the HPSEVERITY procedure assumes that $l_i < p_i$ (strict inequality). If a missing value is returned for some parameter, then the HPSEVERITY procedure assumes that there is no lower bound for that parameter (equivalent to a lower bound of $-\infty$).

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: m, where m is the number of distribution parameters

- *Sequence and type of arguments*:

p1	Output argument that returns the lower bound on the first parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.
p2	Output argument that returns the lower bound on the second parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.
	. . . ..
p*m*	Output argument that returns the lower bound on the *m*th parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.

- *Return value*: The results, lower bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_LOWERBOUNDS(p1, p2);
    outargs p1, p2;

    p1 = <lower bound for P1>;
    p2 = <lower bound for P2>;
endsub;
```

***dist*_PARMINIT**

defines a subroutine that returns the initial values for the distribution's parameters given an empirical distribution function (EDF) estimate.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 4$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

dim	Input numeric value that contains the dimension of the x, nx, and F array arguments.
x{*}	Input numeric array of dimension *dim* that contains values of the random variables at which the EDF estimate is available. It can be assumed that x contains values in an increasing order. In other words, if $i < j$, then x[i] < x[j].
nx{*}	Input numeric array of dimension *dim*. Each nx[i] contains the number of observations in the original data that have the value x[i].
F{*}	Input numeric array of dimension *dim*. Each F[i] contains the EDF estimate for x[i]. This estimate is computed by the HPSEVERITY procedure based on the options specified in the LOSS statement.
Ftype	Input numeric value that contains the type of the EDF estimate that is stored in x and F. See the section "Supplying EDF Estimates to Functions and Subroutines" on page 420 for definition of types.
p1	Output argument that returns the initial value of the first parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.

p2	Output argument that returns the initial value of the second parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.
	…..
pm	Output argument that returns the initial value of the mth parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.

- *Return value*: The results, initial values of the parameters, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_PARMINIT(dim, x{*}, nx{*}, F{*}, Ftype, p1, p2);
    outargs p1, p2;

    /* Code to initialize values of P1 and P2 by using
       dim, x, nx, and F */

    p1 = <initial value for p1>;
    p2 = <initial value for p2>;
endsub;
```

*dist*_PDF

defines a function that returns the value of the probability density function (PDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type*: Function
- *Required*: YES
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the PDF value should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
	…..
pm	Numeric value of the mth parameter

- *Return value*: Numeric value that contains the PDF value $f(x; p_1, p_2, \ldots, p_m)$

If you want to consider this distribution as a candidate distribution when you estimate a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \ldots, p_m) = \frac{1}{p_1} f(\frac{x}{p_1}; 1, p_2, \ldots, p_m)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$f(x; p_1, p_2, \ldots, p_m) = \frac{1}{\exp(p_1)} f(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_PDF(x, P1, P2);
    /* Code to compute PDF by using x, P1, and P2 */

    f = <computed PDF>;
    return (f);
endsub;
```

*dist*_**SCALETRANSFORM**

defines a function that returns a keyword to identify the transform that needs to be applied to the scale parameter to convert it to the first parameter of the distribution.

If you want to use this distribution for modeling regression effects, then the first parameter of this distribution must be a scale parameter. However, for some distributions, a typical or convenient parameterization might not have a scale parameter, but one of the parameters can be a simple transform of the scale parameter. As an example, consider a typical parameterization of the lognormal distribution with two parameters, location μ and shape σ, for which the PDF is defined as follows:

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\log(x)-\mu}{\sigma}\right)^2}$$

You can reparameterize this distribution to contain a parameter θ instead of the parameter μ such that $\mu = \log(\theta)$. The parameter θ would then be a scale parameter. However, if you want to specify the distribution in terms of μ and σ (which is a more recognized form of the lognormal distribution) and still allow it as a candidate distribution for estimating regression effects, then instead of writing another distribution with parameters θ and σ, you can simply define the distribution with μ as the first parameter and specify that it is the logarithm of the scale parameter.

- *Type*: Function
- *Required*: NO
- *Number of arguments*: None
- *Sequence and type of arguments*: Not applicable
- *Return value*: Character value that contains one of the following keywords:

LOG	specifies that the first parameter is the logarithm of the scale parameter.
IDENTITY	specifies that the first parameter is a scale parameter without any transformation.

If this function is not specified, then the IDENTITY transform is assumed.

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SCALETRANSFORM() $8;
    length xform $8;
    xform = "IDENTITY";
    return (xform);
endsub;
```

*dist*_**SDF**

defines a function that returns the value of the survival distribution function (SDF) of the distribution at the specified values of the random variable and distribution parameters.

- *Type*: Function
- *Required*: NO
- *Number of arguments*: $m + 1$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

  x Numeric value of the random variable at which the SDF value should be evaluated

  p1 Numeric value of the first parameter

  p2 Numeric value of the second parameter

  .....

  pm Numeric value of the mth parameter
- *Return value*: Numeric value that contains the SDF value $S(x; p_1, p_2, \ldots, p_m)$

If you want to consider this distribution as a candidate distribution when estimating a response variable model with regression effects, then the first parameter of this distribution must be a scale parameter or log-transformed scale parameter. In other words, if the distribution has a scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \ldots, p_m) = S(\frac{x}{p_1}; 1, p_2, \ldots, p_m)$$

If the distribution has a log-transformed scale parameter, then the following equation must be satisfied:

$$S(x; p_1, p_2, \ldots, p_m) = S(\frac{x}{\exp(p_1)}; 0, p_2, \ldots, p_m)$$

Here is a sample structure of the function for a distribution named 'FOO':

```
function FOO_SDF(x, P1, P2);
    /* Code to compute SDF by using x, P1, and P2 */

    S = <computed SDF>;
    return (S);
endsub;
```

*dist*_**UPPERBOUNDS**

defines a subroutine that returns upper bounds for the parameters of the distribution. If this subroutine is not defined for a given distribution, then the HPSEVERITY procedure assumes that there is no upper bound for any of the parameters. If an upper bound of u_i is returned for a parameter p_i, then the HPSEVERITY procedure assumes that $p_i < u_i$ (strict inequality). If a missing value is returned for some parameter, then the HPSEVERITY procedure assumes that there is no upper bound for that parameter (equivalent to an upper bound of ∞).

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: m, where m is the number of distribution parameters
- *Sequence and type of arguments*:

p1	Output argument that returns the upper bound on the first parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.
p2	Output argument that returns the upper bound on the second parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.
	.....
pm	Output argument that returns the upper bound on the mth parameter. This must be specified in the OUTARGS statement inside the subroutine's definition.

- *Return value*: The results, upper bounds on parameter values, should be returned in the parameter arguments of the subroutine.

Here is a sample structure of the subroutine for a distribution named 'FOO':

```
subroutine FOO_UPPERBOUNDS(p1, p2);
    outargs p1, p2;

    p1 = <upper bound for P1>;
    p2 = <upper bound for P2>;
endsub;
```

***dist_core*GRADIENT**

defines a subroutine that returns the gradient vector of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the gradient should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
	.....
pm	Numeric value of the mth parameter
grad{*}	Output numeric array of size m that contains the gradient vector evaluated at the specified values. If h denotes the value of the *core* function, then the expected order of the values in the array is as follows: $\frac{\partial h}{\partial p_1}\ \frac{\partial h}{\partial p_2} \cdots \frac{\partial h}{\partial p_m}$

- *Return value*: Numeric array that contains the gradient evaluated at x for the parameter values $(p_1, p_2, \ldots, p_m)$

Here is a sample structure of the function for the core function CDF of a distribution named 'FOO':

```
subroutine FOO_CDFGRADIENT(x, P1, P2, grad{*});
    outargs grad;

    /* Code to compute gradient by using x, P1, and P2 */
    grad[1] = <partial derivative of CDF w.r.t. P1
               evaluated at x, P1, P2>;
    grad[2] = <partial derivative of CDF w.r.t. P2
               evaluated at x, P1, P2>;
endsub;
```

*dist_core***HESSIAN**

defines a subroutine that returns the Hessian matrix of the specified *core* function of the distribution at the specified values of the random variable and distribution parameters. The *core* keyword can be PDF, CDF, SDF, LOGPDF, LOGCDF, or LOGSDF.

- *Type*: Subroutine
- *Required*: NO
- *Number of arguments*: $m + 2$, where m is the number of distribution parameters
- *Sequence and type of arguments*:

x	Numeric value of the random variable at which the Hessian matrix should be evaluated
p1	Numeric value of the first parameter
p2	Numeric value of the second parameter
	.....
pm	Numeric value of the mth parameter
hess{*}	Output numeric array of size $m(m+1)/2$ that contains the lower triangular portion of the Hessian matrix in a packed vector form, evaluated at the specified values. If h denotes the value of the *core* function, then the expected order of the values in the array is as follows: $\frac{\partial^2 h}{\partial p_1^2} \mid \frac{\partial^2 h}{\partial p_1 \partial p_2}\ \frac{\partial^2 h}{\partial p_2^2} \mid \cdots \mid \frac{\partial^2 h}{\partial p_1 \partial p_m}\ \frac{\partial^2 h}{\partial p_2 \partial p_m} \cdots \frac{\partial^2 h}{\partial p_m^2}$

- *Return value*: Numeric array that contains the lower triangular portion of the Hessian matrix evaluated at x for the parameter values $(p_1, p_2, \ldots, p_m)$

Here is a sample structure of the subroutine for the core function LOGSDF of a distribution named 'FOO':

```
subroutine FOO_LOGSDFHESSIAN(x, P1, P2, hess{*});
    outargs hess;

    /* Code to compute Hessian by using x, P1, and P2 */
    hess[1] = <second order partial derivative of LOGSDF
               w.r.t. P1 evaluated at x, P1, P2>;
    hess[2] = <second order partial derivative of LOGSDF
               w.r.t. P1 and P2 evaluated at x, P1, P2>;
    hess[3] = <second order partial derivative of LOGSDF
               w.r.t. P2 evaluated at x, P1, P2>;
endsub;
```

Predefined Utility Functions

The following predefined utility functions are provided with the HPSEVERITY procedure and are available in the Sashelp.Svrtdist library:

SVRTUTIL_EDF:

This function computes the empirical distribution function (EDF) estimate at the specified value of the random variable given the EDF estimate for a sample.

- *Type*: Function
- *Signature*: SVRTUTIL_EDF(y, n, x{*}, F{*}, Ftype)
- *Argument Description*:

y	Value of the random variable at which the EDF estimate is desired.
n	Dimension of the *x* and *F* input arrays.
x{*}	Input numeric array of dimension *n* that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension *n* in which each F[i] contains the EDF estimate for x[i]. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the *x* and *F* arrays. See the section "Supplying EDF Estimates to Functions and Subroutines" on page 420 for definition of types.

- *Return value*: The EDF estimate at y.

The type of the sample EDF estimate determines how the EDF estimate at y is computed. For more information, see the section "Supplying EDF Estimates to Functions and Subroutines" on page 420.

SVRTUTIL_EMPLIMMOMENT:

This function computes the empirical estimate of the limited moment of specified order for the specified upper limit, given the EDF estimate for a sample.

- *Type*: Function
- *Signature*: SVRTUTIL_EMPLIMMOMENT(k, u, n, x{*}, F{*}, Ftype)
- *Argument Description*:

k	Order of the desired empirical limited moment.
u	Upper limit on the value of the random variable to be used in the computation of the desired empirical limited moment.
n	Dimension of the *x* and *F* input arrays.
x{*}	Input numeric array of dimension *n* that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension *n* in which each F[i] contains the EDF estimate for x[i]. These values must be sorted in nondecreasing order.

Ftype — Type of the empirical estimate that is stored in the *x* and *F* arrays. See the section "Supplying EDF Estimates to Functions and Subroutines" on page 420 for definition of types.

- *Return value*: The desired empirical limited moment.

The empirical limited moment is computed by using the empirical estimate of the CDF. If $F_n(x)$ denotes the EDF at x, then the empirical limited moment of order k with upper limit u is defined as

$$E_n[(X \wedge u)^k] = k \int_0^u (1 - F_n(x)) x^{k-1} dx$$

The SVRTUTIL_EMPLIMMOMENT function uses the piecewise linear nature of $F_n(x)$ as described in the section "Supplying EDF Estimates to Functions and Subroutines" on page 420 to compute the integration.

SVRTUTIL_HILLCUTOFF:

This function computes an estimate of the value where the right tail of a distribution is expected to begin. The function implements the algorithm described in Danielsson et al. 2001. The description of the algorithm uses the following notation:

n	Number of observations in the original sample.
B	Number of bootstrap samples to draw.
m_1	Size of the bootstrap sample in the first step of the algorithm ($m_1 < n$).
$x_{(i)}^{j,m}$	ith order statistic of jth bootstrap sample of size m ($1 \le i \le m, 1 \le j \le B$).
$x_{(i)}$	ith order statistic of the original sample ($1 \le i \le n$).

Given the input sample x and values of B and m_1, the steps of the algorithm are as follows:

1. Take B bootstrap samples of size m_1 from the original sample.
2. Find the integer k_1 that minimizes the bootstrap estimate of the mean squared error:

$$k_1 = \arg \min_{1 \le k < m_1} Q(m_1, k)$$

3. Take B bootstrap samples of size $m_2 = m_1^2/n$ from the original sample.
4. Find the integer k_2 that minimizes the bootstrap estimate of the mean squared error:

$$k_2 = \arg \min_{1 \le k < m_2} Q(m_2, k)$$

5. Compute the integer k_{opt}, which is used for computing the cutoff point:

$$k_{\text{opt}} = \frac{k_1^2}{k_2} \left(\frac{\log(k_1)}{2\log(m_1) - \log(k_1)} \right)^{2 - 2\log(k_1)/\log(m_1)}$$

6. Set the cutoff point equal to $x_{(k_{\text{opt}}+1)}$.

The bootstrap estimate of the mean squared error is computed as

$$Q(m, k) = \frac{1}{B} \sum_{j=1}^{B} \text{MSE}_j(m, k)$$

The mean squared error of jth bootstrap sample is computed as

$$\text{MSE}_j(m,k) = (M_j(m,k) - 2(\gamma_j(m,k))^2)^2$$

where $M_j(m,k)$ is a control variate proposed by Danielsson et al. 2001,

$$M_j(m,k) = \frac{1}{k}\sum_{i=1}^{k}\left(\log(x^{j,m}_{(m-i+1)}) - \log(x^{j,m}_{(m-k)})\right)^2$$

and $\gamma_j(m,k)$ is the Hill's estimator of the tail index (Hill 1975),

$$\gamma_j(m,k) = \frac{1}{k}\sum_{i=1}^{k}\log(x^{j,m}_{(m-i+1)}) - \log(x^{j,m}_{(m-k)})$$

This algorithm has two tuning parameters, B and m_1. The number of bootstrap samples B is chosen based on the availability of computational resources. The optimal value of m_1 is chosen such that the following ratio, $R(m_1)$, is minimized:

$$R(m_1) = \frac{(Q(m_1,k_1))^2}{Q(m_2,k_2)}$$

The SVRTUTIL_HILLCUTOFF utility function implements the preceding algorithm. It uses the grid search method to compute the optimal value of m_1.

- *Type*: Function
- *Signature*: SVRTUTIL_HILLCUTOFF(n, x{*}, b, s, status)
- *Argument Description*:

n	Dimension of the array *x*.
x{*}	Input numeric array of dimension n that contains the sample.
b	Number of bootstrap samples used to estimate the mean squared error. If *b* is less than 10, then a default value of 50 is used.
s	Approximate number of steps used to search the optimal value of m_1 in the range $[n^{0.75}, n-1]$. If *s* is less than or equal to 1, then a default value of 10 is used.
status	Output argument that contains the status of the algorithm. If the algorithm succeeds in computing a valid cutoff point, then *status* is set to 0. If the algorithm fails, then *status* is set to 1.

- *Return value*: The cutoff value where the right tail is estimated to start. If the size of the input sample is inadequate ($n \le 5$), then a missing value is returned and *status* is set to a missing value. If the algorithm fails to estimate a valid cutoff value (*status* = 1), then the fifth largest value in the input sample is returned.

SVRTUTIL_PERCENTILE:
This function computes the specified empirical percentile given the EDF estimates.

- *Type*: Function
- *Signature*: SVRTUTIL_PERCENTILE(p, n, x{*}, F{*}, Ftype)

- *Argument Description*:

p	Desired percentile. The value must be in the interval (0,1). The function returns the 100pth percentile.
n	Dimension of the *x* and *F* input arrays.
x{*}	Input numeric array of dimension *n* that contains values of the random variable observed in the sample. These values are sorted in nondecreasing order.
F{*}	Input numeric array of dimension *n* in which each F[i] contains the EDF estimate for x[i]. These values must be sorted in nondecreasing order.
Ftype	Type of the empirical estimate that is stored in the *x* and *F* arrays. See the section "Supplying EDF Estimates to Functions and Subroutines" on page 420 for definition of types.

- *Return value*: The 100pth percentile of the input sample.

The method used to compute the percentile depends on the type of the EDF estimate (Ftype argument).

Ftype = 1 Smoothed empirical estimates are computed using the method described in Klugman, Panjer, and Willmot (1998). Let $\lfloor x \rfloor$ denote the greatest integer less than or equal to x. Define $g = \lfloor p(n+1) \rfloor$ and $h = p(n+1) - g$. Then the empirical percentile $\hat{\pi}_p$ is defined as

$$\hat{\pi}_p = (1-h)x[g] + hx[g+1]$$

This method does not work if $p < 1/(n+1)$ or $p > n/(n+1)$. If $p < 1/(n+1)$, then the function returns $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $p > n/(n+1)$, then $\hat{\pi}_p = x[n]$.

Ftype = 2 If $p < F[1]$, then $\hat{\pi}_p = x[1]/2$, which assumes that the EDF is 0 in the interval $[0, x[1])$. If $|p - F[i]| < \epsilon$ for some value of i and $i < n$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = \frac{x[i] + x[i+1]}{2}$$

where ϵ is a machine-precision constant as returned by the SAS function CONSTANT('MACEPS'). If $F[i-1] < p < F[i]$, then $\hat{\pi}_p$ is computed as

$$\hat{\pi}_p = x[i]$$

If $p \geq F[n]$, then $\hat{\pi}_p = x[n]$.

SVRTUTIL_RAWMOMENTS:

This subroutine computes the raw moments of a sample.

- *Type*: Subroutine
- *Signature*: SVRTUTIL_RAWMOMENTS(n, x{*}, nx{*}, nRaw, raw{*})
- *Argument Description*:

n	Dimension of the *x* and *nx* input arrays.

x{*}	Input numeric array of dimension *n* that contains distinct values of the random variable that are observed in the sample.
nx{*}	Input numeric array of dimension *n* in which each *nx*[i] contains the number of observations in the sample that have the value x[i].
nRaw	Desired number of raw moments. The output array *raw* contains the first *nRaw* raw moments.
raw{*}	Output array of raw moments. The kth element in the array (raw{k}) contains the kth raw moment, where $1 \le k \le$ nRaw.

- *Return value*: Numeric array *raw* that contains the first *nRaw* raw moments. The array contains missing values if the sample has no observations (that is, if all the values in the *nx* array add up to zero).

SVRTUTIL_SORT:

This function sorts the given array of numeric values in an ascending or descending order.

- *Type*: Subroutine
- *Signature*: SVRTUTIL_SORT(n, x{*}, flag)
- *Argument Description*:

n	Dimension of the input array *x*.
x{*}	Numeric array that contains the values to be sorted at input. The subroutine uses the same array to return the sorted values.
flag	A numeric value that controls the sort order. If *flag* is 0, then the values are sorted in an ascending order. If *flag* has any value other than 0, then the values are sorted in descending order.

- *Return value*: Numeric array *x*, which is sorted in place (that is, the sorted array is stored in the same storage area occupied by the input array *x*).

You can use the following predefined functions when you use the FCMP procedure to define functions and subroutines. They are summarized here for your information. For more information, see the FCMP procedure documentation in *Base SAS Procedures Guide*.

INVCDF:

This function computes the quantile from any continuous probability distribution by numerically inverting the CDF of that distribution. You need to specify the CDF function of the distribution, the values of its parameters, and the cumulative probability to compute the quantile.

LIMMOMENT:

This function computes the limited moment of order k with upper limit u for any continuous probability distribution. The limited moment is defined as

$$\begin{aligned} E[(X \wedge u)^k] &= \int_0^u x^k f(x)dx + \int_u^\infty u^k f(x)dx \\ &= \int_0^u x^k f(x)dx + u^k(1 - F(u)) \end{aligned}$$

where $f(x)$ and $F(x)$ denote the PDF and the CDF of the distribution, respectively. The LIMMOMENT function uses the following alternate definition, which can be derived using integration-by-parts:

$$E[(X \wedge u)^k] = k \int_0^u (1 - F(x))x^{k-1} dx$$

You need to specify the CDF function of the distribution, the values of its parameters, and the values of k and u to compute the limited moment.

Input Data Sets

PROC HPSEVERITY accepts DATA= and INEST= data sets as input data sets. This section details the information they are expected to contain.

DATA= Data Set

The DATA= data set is expected to contain the values of the analysis variables specified in the LOSS statement and the SCALEMODEL statement.

INEST= Data Set

The INEST= data set is expected to contain the initial values of the parameters for the parameter estimation process. The data set must contain the following variables:

MODEL identifying name of the distribution for which the estimates are provided.

TYPE type of the estimate. The value of this variable must be EST for an observation to be valid.

<Parameter 1> . . . <Parameter M>
M variables, named after the parameters of all candidate distributions, that contain initial values of the respective parameters. M is the cardinality of the union of parameter name sets from all candidate distributions. In an observation, estimates are read only from variables for parameters that correspond to the distribution specified by the _MODEL_ variable.

If you specify a missing value for some parameters, then default initial values are used unless the parameter is initialized by using the INIT= option in the DIST statement. If you want to use the *dist*_PARMINIT subroutine for initializing the parameters of a model, then you should either not specify the model in the INEST= data set or specify missing values for all the distribution parameters in the INEST= data set and not use the INIT= option in the DIST statement.

If regressors are specified, then the initial value provided for the first parameter of each distribution must be the base value of the scale or log-transformed scale parameter. For more information, see the section "Estimating Regression Effects" on page 416.

<Regressor 1> . . . <Regressor K>

If K regressors are specified in the SCALEMODEL statement, then the INEST= data set must contain K variables that are named for each regressor. The variables contain initial values of the respective regression coefficients. If a regressor is linearly dependent on other regressors, then you can indicate this by providing a special missing value of .R for the respective variable. If a variable is marked as linearly dependent for one model, then it must be marked so for all the models. Similarly, if a variable is not marked as linearly dependent for one model, then it must be marked so for all the models.

Output Data Sets

PROC HPSEVERITY writes the OUTEST=, OUTMODELINFO=, and OUTSTAT= data sets when requested by their respective options. The data sets and their contents are described in the following sections.

OUTEST= Data Set

The OUTEST= data set records the estimates of the model parameters. It also contains estimates of their standard errors and optionally their covariance structure.

If the COVOUT option is not specified, then the data set contains the following variables:

MODEL
identifying name of the distribution model. The observation contains information about this distribution.

TYPE
type of the estimates reported in this observation. It can take one of the following two values:

EST	point estimates of model parameters
STDERR	standard error estimates of model parameters

STATUS
status of the reported estimates. The possible values are listed in the section "_STATUS_ Variable Values" on page 444.

<Parameter 1> . . . <Parameter M>

M variables, named after the parameters of all candidate distributions, that contain estimates of the respective parameters. M is the cardinality of the union of parameter name sets from all candidate distributions. In an observation, estimates are populated only for parameters that correspond to the distribution specified by the _MODEL_ variable. If _TYPE_ is EST, then the estimates are missing if the model does not converge. If _TYPE_ is STDERR, then the estimates are missing if covariance estimates cannot be obtained.

If regressors are specified, then the estimate reported for the first parameter of each distribution is the estimate of the base value of the scale or log-transformed scale parameter. For more information, see the section "Estimating Regression Effects" on page 416.

<Regressor 1> . . . <Regressor K>
If K regressors are specified in the SCALEMODEL statement, then the OUTEST= data set contains K variables that are named for each regressor. The variables contain estimates for their respective regression coefficients. If a regressor is deemed to be linearly dependent on other regressors, then a warning message is printed to the SAS log and a special missing value of .R is written in the respective variable. If _TYPE_ is EST, then the estimates are missing if the model does not converge. If _TYPE_ is STDERR, then the estimates are missing if covariance estimates cannot be obtained.

If the COVOUT option is specified, then the OUTEST= data set contains additional observations that contain the estimates of the covariance structure. Given the symmetric nature of the covariance structure, only the lower triangular portion is reported. In addition to the variables listed and described previously, the data set contains the following variables that are either new or have a modified description:

TYPE	type of the estimates reported in this observation. For observations that contain rows of the covariance structure, the value is COV.
STATUS	status of the reported estimates. For observations that contain rows of the covariance structure, the status is 0 if covariance estimation was successful. If estimation fails, the status is 1 and a single observation is reported with _TYPE_=COV and missing values for all the parameter variables.
NAME	Name of the parameter for the row of covariance matrix reported in the current observation.

OUTMODELINFO= Data Set

The OUTMODELINFO= data set records the information about each specified distribution. It contains the following variables:

MODEL	identifying name of the distribution model. The observation contains information about this distribution.
DESCRIPTION	descriptive name of the model. This has a nonmissing value only if the DESCRIPTION function has been defined for this model.
_PARMNAME1 ... _PARMNAMEM	M variables that contain names of parameters of the distribution model, where M is the maximum number of parameters across all the specified distribution models. For a given distribution with m parameters, values of variables _PARMNAMEj $(j > m)$ are missing.

OUTSTAT= Data Set

The OUTSTAT= data set records statistics of fit and model selection information. The data set contains the following variables:

MODEL	identifying name of the distribution model. The observation contains information about this distribution.

NMODELPARM	number of parameters in the distribution.
NESTPARM	number of estimated parameters. This includes the regression parameters, if any regressors are specified.
NOBS	number of nonmissing observations used for parameter estimation.
STATUS	status of the parameter estimation process for this model. The possible values are listed in the section "_STATUS_ Variable Values" on page 444.
SELECTED	indicator of the best distribution model. If the value is 1, then this model is the best model according to the specified model selection criterion. This value is missing if the parameter estimation process does not converge for this model.
Neg2LogLike	value of the log likelihood, multiplied by –2, that is attained at the end of the parameter estimation process. This value is missing if the parameter estimation process does not converge for this model.
AIC	value of the Akaike's information criterion (AIC) that is attained at the end of the parameter estimation process. This value is missing if the parameter estimation process does not converge for this model.
AICC	value of the corrected Akaike's information criterion (AICC) that is attained at the end of the parameter estimation process. This value is missing if the parameter estimation process does not converge for this model.
BIC	value of the Schwarz Bayesian information criterion (BIC) that is attained at the end of the parameter estimation process. This value is missing if the parameter estimation process does not converge for this model.

STATUS Variable Values

The _STATUS_ variable in the OUTEST= and OUTSTAT= data sets contains a value that indicates the status of the parameter estimation process for the respective distribution model. The variable can take the following values in the OUTEST= data set for _TYPE_=EST observations and in the OUTSTAT= data set:

0	The parameter estimation process converged for this model.
301	The parameter estimation process might not have converged for this model because there is no improvement in the objective function value. This might indicate that the initial values of the parameters are optimal, or you can try different convergence criteria in the NLOPTIONS statement.
302	The parameter estimation process might not have converged for this model because the number of iterations exceeded the maximum allowed value. You can try setting a larger value for the MAXITER= options in the NLOPTIONS statement.
303	The parameter estimation process might not have converged for this model because the number of objective function evaluations exceeded the maximum allowed value. You can try setting a larger value for the MAXFUNC= options in the NLOPTIONS statement.
304	The parameter estimation process might not have converged for this model because the time taken by the process exceeded the maximum allowed value. You can try setting a larger value for the MAXTIME= option in the NLOPTIONS statement.
400	The parameter estimation process did not converge for this model.

The _STATUS_ variable can take the following values in the OUTEST= data set for _TYPE_=STDERR and _TYPE_=COV observations:

0	The covariance and standard error estimates are available and valid.
1	The covariance and standard error estimates are not available, because the process of computing covariance estimates failed.

Displayed Output

The HPSEVERITY procedure optionally produces displayed output by using the Output Delivery System (ODS). All output is controlled by the PRINT= option in the PROC HPSEVERITY statement. Table 14.7 relates the PRINT= options to ODS tables.

Table 14.7 ODS Tables Produced in PROC HPSEVERITY

ODS Table Name	Description	Option
DescStats	Descriptive statistics for the response variable	PRINT=DESCSTATS
RegDescStats	Descriptive statistics for the regressor variables	PRINT=DESCSTATS
ModelSelection	Model selection summary	PRINT=SELECTION
AllFitStatistics	Statistics of fit for all the distribution models	PRINT=ALLFITSTATS
PerformanceInfo	Execution environment information that pertains to the computational performance	Default
Timing	Timing information for various computational stages of the procedure	DETAILS (PERFORMANCE statement)
EstimationDetails	Details of the estimation process for all the distribution models	PRINT=ESTIMATIONDETAILS
InitialValues	Initial parameter values and bounds	PRINT=INITIALVALUES
ConvergenceStatus	Convergence status of parameter estimation process	PRINT=CONVSTATUS
IterationHistory	Optimization iteration history	PRINT=NLOHISTORY
OptimizationSummary	Optimization summary	PRINT=NLOSUMMARY
StatisticsOfFit	Statistics of fit	PRINT=STATISTICS
ParameterEstimates	Final parameter estimates	PRINT=ESTIMATES

PRINT= Option

If the PRINT= option is not specified, then by default PROC HPSEVERITY produces the ModelSelection, PerformanceInfo, ConvergenceStatus, OptimizationSummary, StatisticsOfFit, and ParameterEstimates ODS tables.

You can specify the following values for the PRINT= option:

PRINT=DESCSTATS
displays the descriptive statistics for the response variable. If regressor variables are specified, a table with their descriptive statistics is also displayed.

PRINT=SELECTION
displays the model selection table. The table shows the convergence status of each candidate model, and the value of the selection criterion along with an indication of the selected model.

PRINT=ALLFITSTATS
displays the comparison of all the statistics of fit for all the models in one table. The table does not include the models whose parameter estimation process does not converge. If all the models fail to converge, then this table is not produced. If the table contains more than one model, then the best model according to each statistic is indicated with an asterisk (*) in that statistic's column.

PRINT=ESTIMATIONDETAILS
displays the comparative details of the estimation process that is used to fit each candidate distribution. If you specify the DETAILS option in the PERFORMANCE statement, then this table contains a column that indicates the time taken to estimate each candidate model.

PRINT=INITIALVALUES
displays the initial values and bounds used for estimating each model.

PRINT=CONVSTATUS
displays the convergence status of the parameter estimation process.

PRINT=NLOHISTORY
displays the iteration history of the nonlinear optimization process used for estimating the parameters.

PRINT=NLOSUMMARY
displays the summary of the nonlinear optimization process used for estimating the parameters.

PRINT=STATISTICS
displays the statistics of fit for each model. The statistics of fit are not displayed for models whose parameter estimation process does not converge.

PRINT=ESTIMATES
displays the final estimates of parameters. The estimates are not displayed for models whose parameter estimation process does not converge.

Performance Information

The “Performance Information” table is produced by default and displays information about the grid host for distributed execution and about whether the procedure executes in client mode, distributed mode, or alongside-the-database mode. The number of compute nodes and threads are also displayed, depending on the environment.

Timing Information

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a “Timing” table in which the elapsed and relative times for each main task of the procedure is displayed.

Examples: HPSEVERITY Procedure

Example 14.1: Defining a Model for Gaussian Distribution

Suppose you want to fit a distribution model other than one of the predefined ones available to you. Suppose you want to define a model for the Gaussian distribution with the following typical parameterization of the PDF (f) and CDF (F):

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$
$$F(x;\mu,\sigma) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

For PROC HPSEVERITY, a *distribution model* consists of a set of functions and subroutines that are defined with the FCMP procedure. Each function and subroutine should be written following certain rules. For more information, see the section “Defining a Severity Distribution Model with the FCMP Procedure” on page 424.

NOTE: The Gaussian distribution is not a commonly used severity distribution. It is used in this example primarily to illustrate the process of defining your own distribution models. Although the distribution has a support over the entire real line, you can fit the distribution with PROC HPSEVERITY only if the input sample contains nonnegative values.

The following SAS statements define a distribution model named NORMAL for the Gaussian distribution. The OUTLIB= option in the PROC FCMP statement stores the compiled versions of the functions and subroutines in the ‘models’ package of the Work.Sevexmpl library. The LIBRARY= option in the PROC FCMP statement enables this PROC FCMP step to use the SVRTUTIL_RAWMOMENTS utility subroutine that is available in the Sashelp.Svrtdist library. The subroutine is described in the section “Predefined Utility Functions” on page 436.

```
/*-------- Define Normal Distribution with PROC FCMP  ----------*/
proc fcmp library=sashelp.svrtdist outlib=work.sevexmpl.models;
   function normal_pdf(x,Mu,Sigma);
      /* Mu    : Location */
      /* Sigma : Standard Deviation */
      return ( exp(-(x-Mu)**2/(2 * Sigma**2)) /
             (Sigma * sqrt(2*constant('PI'))) );
   endsub;

   function normal_cdf(x,Mu,Sigma);
      /* Mu    : Location */
      /* Sigma : Standard Deviation */
      z = (x-Mu)/Sigma;
      return (0.5 + 0.5*erf(z/sqrt(2)));
   endsub;

   subroutine normal_parminit(dim, x[*], nx[*], F[*], Ftype, Mu, Sigma);
      outargs Mu, Sigma;
      array m[2] / nosymbols;

      /* Compute estimates by using method of moments */
      call svrtutil_rawmoments(dim, x, nx, 2, m);
      Mu    = m[1];
      Sigma = sqrt(m[2] - m[1]**2);
   endsub;

   subroutine normal_lowerbounds(Mu, Sigma);
      outargs Mu, Sigma;
      Mu = .;    /* Mu has no lower bound */
      Sigma = 0; /* Sigma > 0 */
   endsub;
quit;
```

The statements define the two functions required of any distribution model (NORMAL_PDF and NORMAL_CDF) and two optional subroutines (NORMAL_PARMINIT and NORMAL_LOWERBOUNDS). The name of each function or subroutine must follow a specific structure. It should start with the model's short or identifying name, which is 'NORMAL' in this case, followed by an underscore '_', followed by a keyword suffix such as 'PDF'. Each function or subroutine has a specific purpose. For more information about all the functions and subroutines that you can define for a distribution model, see the section "Defining a Severity Distribution Model with the FCMP Procedure" on page 424. Following is the description of each function and subroutine defined in this example:

- The PDF and CDF suffixes define functions that return the probability density function and cumulative distribution function values, respectively, given the values of the random variable and the distribution parameters.
- The PARMINIT suffix defines a subroutine that returns the initial values for the parameters by using the sample data or the empirical distribution function (EDF) estimate computed from it. In this example, the parameters are initialized by using the method of moments. Hence, you do not need to use the EDF estimates, which are available in the F array. The first two raw moments of the Gaussian distribution are as follows:

  $$E[x] = \mu, \; E[x^2] = \mu^2 + \sigma^2$$

Given the sample estimates, m_1 and m_2, of these two raw moments, you can solve the equations $E[x] = m_1$ and $E[x^2] = m_2$ to get the following estimates for the parameters: $\hat{\mu} = m_1$ and $\hat{\sigma} = \sqrt{m_2 - m_1^2}$. The NORMAL_PARMINIT subroutine implements this solution. It uses the SVRTUTIL_RAWMOMENTS utility subroutine to compute the first two raw moments.

- The LOWERBOUNDS suffix defines a subroutine that returns the lower bounds on the parameters. PROC HPSEVERITY assumes a default lower bound of 0 for all the parameters when a LOWERBOUNDS subroutine is not defined. For the parameter μ (*Mu*), there is no lower bound, so you need to define the NORMAL_LOWERBOUNDS subroutine. It is recommended that you assign bounds for all the parameters when you define the LOWERBOUNDS subroutine or its counterpart, the UPPERBOUNDS subroutine. Any unassigned value is returned as a missing value, which PROC HPSEVERITY interprets to mean that the parameter is unbounded, and that might not be what you want.

You can now use this distribution model with PROC HPSEVERITY. Let the following DATA step statements simulate a normal sample with $\mu = 10$ and $\sigma = 2.5$:

```
/*-------- Simulate a Normal sample ----------*/
data testnorm(keep=y);
   call streaminit(12345);
   do i=1 to 100;
      y = rand('NORMAL', 10, 2.5);
      output;
   end;
run;
```

Prior to using your distribution with PROC HPSEVERITY, you must communicate the location of the library that contains the definition of the distribution and the locations of libraries that contain any functions and subroutines used by your distribution model. The following OPTIONS statement sets the CMPLIB= system option to include the FCMP library Work.Sevexmpl in the search path used by PROC HPSEVERITY to find FCMP functions and subroutines.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);
```

Now, you are ready to fit the NORMAL distribution model with PROC HPSEVERITY. The following statements fit the model to the values of Y in the Work.Testnorm data set:

```
/*--- Fit models with PROC HPSEVERITY ---*/
proc hpseverity data=testnorm print=all;
   loss y;
   dist Normal;
run;
```

The DIST statement specifies the identifying name of the distribution model, which is 'NORMAL'. Neither is the INEST= option specified in the PROC HPSEVERITY statement nor is the INIT= option specified in the DIST statement. So, PROC HPSEVERITY initializes the parameters by invoking the NORMAL_PARMINIT subroutine.

Some of the results prepared by the preceding PROC HPSEVERITY step are shown in Output 14.1.1 and Output 14.1.2. The descriptive statistics of variable Y and the "Model Selection Table," which includes just the normal distribution, are shown in Output 14.1.1.

Output 14.1.1 Summary of Results for Fitting the Normal Distribution

```
                       The HPSEVERITY Procedure

                           Input Data Set

                        Name    WORK.TESTNORM

                 Descriptive Statistics for Variable y

         Number of Observations                               100
         Number of Observations Used for Estimation           100
         Minimum                                          3.88249
         Maximum                                         16.00864
         Mean                                            10.02059
         Standard Deviation                               2.37730

                        Model Selection Table

        Distribution    Converged    -2 Log Likelihood    Selected

        Normal          Yes                  455.97541    Yes
```

The initial values for the parameters, the optimization summary, and the final parameter estimates are shown in Output 14.1.2. No iterations are required to arrive at the final parameter estimates, which are identical to the initial values. This confirms the fact that the maximum likelihood estimates for the Gaussian distribution are identical to the estimates obtained by the method of moments that was used to initialize the parameters in the NORMAL_PARMINIT subroutine.

Output 14.1.2 Details of the Fitted Normal Distribution Model

```
                       The HPSEVERITY Procedure

                      Distribution Information

               Name                                 Normal
               Number of Distribution Parameters         2

                  Initial Parameter Values and Bounds
                        for Normal Distribution

                          Initial         Lower         Upper
           Parameter        Value         Bound         Bound

           Mu            10.02059        -Infty         Infty
           Sigma          2.36538    1.05367E-8         Infty

               Optimization Summary for Normal Distribution

              Optimization Technique            Trust Region
              Number of Iterations                         0
              Number of Function Evaluations               4
              Log Likelihood                      -227.98770
```

Output 14.1.2 *continued*

```
                  Parameter Estimates for Normal Distribution

                                        Standard                   Approx
      Parameter       Estimate             Error     t Value     Pr > |t|

      Mu              10.02059           0.23894       41.94       <.0001
      Sigma            2.36538           0.16896       14.00       <.0001
```

The NORMAL distribution defined and illustrated here has no scale parameter, because all the following inequalities are true:

$$
\begin{aligned}
f(x;\mu,\sigma) &\neq \frac{1}{\mu} f(\frac{x}{\mu};1,\sigma) \\
f(x;\mu,\sigma) &\neq \frac{1}{\sigma} f(\frac{x}{\sigma};\mu,1) \\
F(x;\mu,\sigma) &\neq F(\frac{x}{\mu};1,\sigma) \\
F(x;\mu,\sigma) &\neq F(\frac{x}{\sigma};\mu,1)
\end{aligned}
$$

This implies that you cannot estimate the effect of regressors on a model for the response variable based on this distribution.

Example 14.2: Defining a Model for the Gaussian Distribution with a Scale Parameter

If you want to estimate the effects of regressors, then the model needs to be parameterized to have a scale parameter. Although this might not be always possible, it is possible for the Gaussian distribution by replacing the location parameter μ with another parameter, $\alpha = \mu/\sigma$, and defining the PDF (f) and the CDF (F) as follows:

$$
\begin{aligned}
f(x;\sigma,\alpha) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x}{\sigma}-\alpha\right)^2\right) \\
F(x;\sigma,\alpha) &= \frac{1}{2}\left(1 + \text{erf}\left(\frac{1}{\sqrt{2}}\left(\frac{x}{\sigma}-\alpha\right)\right)\right)
\end{aligned}
$$

You can verify that σ is the scale parameter, because both of the following equalities are true:

$$
\begin{aligned}
f(x;\sigma,\alpha) &= \frac{1}{\sigma} f(\frac{x}{\sigma};1,\alpha) \\
F(x;\sigma,\alpha) &= F(\frac{x}{\sigma};1,\alpha)
\end{aligned}
$$

NOTE: The Gaussian distribution is not a commonly used severity distribution. It is used in this example primarily to illustrate the concept of parameterizing a distribution such that it has a scale parameter. Although the distribution has a support over the entire real line, you can fit the distribution with PROC HPSEVERITY only if the input sample contains nonnegative values.

The following statements use the alternate parameterization to define a new model named NORMAL_S. The definition is stored in the Work.Sevexmpl library.

```
/*-------- Define Normal Distribution With Scale Parameter  ----------*/
proc fcmp library=sashelp.svrtdist outlib=work.sevexmpl.models;
   function normal_s_pdf(x, Sigma, Alpha);
      /* Sigma : Scale & Standard Deviation */
      /* Alpha : Scaled mean */
      return ( exp(-(x/Sigma - Alpha)**2/2) /
               (Sigma * sqrt(2*constant('PI'))) );
   endsub;

   function normal_s_cdf(x, Sigma, Alpha);
      /* Sigma : Scale & Standard Deviation */
      /* Alpha : Scaled mean */
      z = x/Sigma - Alpha;
      return (0.5 + 0.5*erf(z/sqrt(2)));
   endsub;

   subroutine normal_s_parminit(dim, x[*], nx[*], F[*], Ftype, Sigma, Alpha);
      outargs Sigma, Alpha;
      array m[2] / nosymbols;

      /* Compute estimates by using method of moments */
      call svrtutil_rawmoments(dim, x, nx, 2, m);
      Sigma = sqrt(m[2] - m[1]**2);
      Alpha = m[1]/Sigma;
   endsub;

   subroutine normal_s_lowerbounds(Sigma, Alpha);
      outargs Sigma, Alpha;
      Alpha = .; /* Alpha has no lower bound */
      Sigma = 0; /* Sigma > 0 */
   endsub;
quit;
```

An important point to note is that the scale parameter *Sigma* is the first distribution parameter (after the ‘x’ argument) listed in the signatures of NORMAL_S_PDF and NORMAL_S_CDF functions. *Sigma* is also the first distribution parameter listed in the signatures of other subroutines. This is required by PROC HPSEVERITY, so that it can identify which is the scale parameter. When regressor variables are specified, PROC HPSEVERITY checks whether the first parameter of each candidate distribution is a scale parameter (or a log-transformed scale parameter if *dist*_SCALETRANSFORM subroutine is defined for the distribution with LOG as the transform). If it is not, then an appropriate message is written the SAS log and that distribution is not fitted.

Let the following DATA step statements simulate a sample from the normal distribution where the parameter σ is affected by the regressors as follows:

$$\sigma = \exp(1 + 0.5\ \mathrm{X1} + 0.75\ \mathrm{X3} - 2\ \mathrm{X4} + \mathrm{X5})$$

The sample is simulated such that the regressor X2 is linearly dependent on regressors X1 and X3.

```
/*--- Simulate a Normal sample affected by Regressors ---*/
data testnorm_reg(keep=y x1-x5 Sigma);
   array x{*} x1-x5;
   array b{6} _TEMPORARY_ (1 0.5 . 0.75 -2 1);
   call streaminit(34567);
   label y='Normal Response Influenced by Regressors';

   do n = 1 to 100;
      /* simulate regressors */
      do i = 1 to dim(x);
         x(i) = rand('UNIFORM');
      end;
      /* make x2 linearly dependent on x1 */
      x(2) = 5 * x(1);

      /* compute log of the scale parameter */
      logSigma = b(1);
      do i = 1 to dim(x);
         if (i ne 2) then
            logSigma = logSigma + b(i+1) * x(i);
      end;

      Sigma = exp(logSigma);
      y = rand('NORMAL', 25, Sigma);

      output;
   end;
run;
```

The following statements use PROC HPSEVERITY to fit the NORMAL_S distribution model along with some of the predefined distributions to the simulated sample:

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*-------- Fit models with PROC HPSEVERITY --------*/
proc hpseverity data=testnorm_reg print=all;
   loss y;
   scalemodel x1-x5;
   dist Normal_s burr logn pareto weibull;
run;
```

The “Model Selection Table” in Output 14.2.1 indicates that all the models, except the Burr distribution model, have converged. Also, only three models, Normal_s, Burr, and Weibull, seem to have a good fit for the data. The table that compares all the fit statistics indicates that Normal_s model is the best according to the likelihood-based statistics.

Output 14.2.1 Summary of Results for Fitting the Normal Distribution with Regressors

```
                           The HPSEVERITY Procedure

                               Input Data Set

                          Name    WORK.TESTNORM_REG

                            Model Selection Table

         Distribution    Converged    -2 Log Likelihood    Selected

         Normal_s        Yes                  603.95786    Yes
         Burr            Maybe                612.80861    No
         Logn            Yes                  749.20125    No
         Pareto          Yes                  841.07013    No
         Weibull         Yes                  612.77496    No

                           All Fit Statistics Table

                      -2 Log
  Distribution     Likelihood          AIC            AICC             BIC

  Normal_s         603.95786*     615.95786*     616.86108*     631.58888*
  Burr             612.80861      626.80861      628.02600      645.04480
  Logn             749.20125      761.20125      762.10448      776.83227
  Pareto           841.07013      853.07013      853.97336      868.70115
  Weibull          612.77496      624.77496      625.67819      640.40598
```

This prompts you to further evaluate why the model with Burr distribution has not converged. The initial values, convergence status, and the optimization summary for the Burr distribution are shown in Output 14.2.2. The initial values table indicates that the regressor X2 is redundant, which is expected. More importantly, the convergence status indicates that it requires more than 50 iterations. PROC HPSEVERITY enables you to change several settings of the optimizer by using the NLOPTIONS statement. In this case, you can increase the limit of 50 on the iterations, change the convergence criterion, or change the technique to something other than the default trust-region technique.

Output 14.2.2 Details of the Fitted Burr Distribution Model

```
                           The HPSEVERITY Procedure

                           Distribution Information

           Name                                                Burr
           Description                             Burr Distribution
           Number of Distribution Parameters                      3
           Number of Regression Parameters                        4
```

Output 14.2.2 *continued*

```
                    Initial Parameter Values and
                    Bounds for Burr Distribution

                          Initial         Lower         Upper
            Parameter       Value         Bound         Bound

            Theta        25.75198    1.05367E-8         Infty
            Alpha         2.00000    1.05367E-8         Infty
            Gamma         2.00000    1.05367E-8         Infty
            x1            0.07345    -709.78271     709.78271
            x2          Redundant             .             .
            x3           -0.14056    -709.78271     709.78271
            x4            0.27064    -709.78271     709.78271
            x5           -0.23230    -709.78271     709.78271

               Convergence Status for Burr Distribution

                    Needs more than 50 iterations.

              Optimization Summary for Burr Distribution

               Optimization Technique            Trust Region
               Number of Iterations                        50
               Number of Function Evaluations             132
               Log Likelihood                      -306.40430
```

The following PROC HPSEVERITY step uses the NLOPTIONS statement to change the convergence criterion and the limits on the iterations and function evaluations, exclude the lognormal and Pareto distributions that have been confirmed previously to fit the data poorly, and exclude the redundant regressor X2 from the model:

```
/*--- Refit and compare models with higher limit on iterations ---*/
proc hpseverity data=testnorm_reg print=all;
   loss y;
   scalemodel x1 x3-x5;
   dist Normal_s burr weibull;
   nloptions absfconv=2.0e-5 maxiter=100 maxfunc=500;
run;
```

The results shown in Output 14.2.3 indicate that the Burr distribution has now converged and that the Burr and Weibull distributions have an almost identical fit for the data. The NORMAL_S distribution is still the best distribution according to the likelihood-based criteria.

Output 14.2.3 Summary of Results after Changing Maximum Number of Iterations

```
                       The HPSEVERITY Procedure

                           Input Data Set

                      Name    WORK.TESTNORM_REG
```

Output 14.2.3 *continued*

```
                         Model Selection Table

         Distribution    Converged    -2 Log Likelihood    Selected

         Normal_s        Yes                  603.95786    Yes
         Burr            Yes                  612.78605    No
         Weibull         Yes                  612.77496    No

                        All Fit Statistics Table

                     -2 Log
  Distribution     Likelihood        AIC           AICC           BIC

  Normal_s         603.95786*     615.95786*     616.86108*     631.58888*
  Burr             612.78605      626.78605      628.00344      645.02224
  Weibull          612.77496      624.77496      625.67819      640.40598
```

Example 14.3: Defining a Model for Mixed-Tail Distributions

In some applications, a few severity values tend to be extreme as compared to the typical values. The extreme values represent the worst case scenarios and cannot be discarded as outliers. Instead, their distribution must be modeled to prepare for their occurrences. In such cases, it is often useful to fit one distribution to the non-extreme values and another distribution to the extreme values. The *mixed-tail* distribution mixes two distributions: one for the *body* region, which contains the non-extreme values, and another for the *tail* region, which contains the extreme values. The tail distribution is typically a generalized Pareto distribution (GPD), because it is usually good for modeling the conditional excess severity above a threshold. The body distribution can be any distribution. The following definitions are used in describing a generic formulation of a mixed-tail distribution:

$g(x)$	PDF of the body distribution
$G(x)$	CDF of the body distribution
$h(x)$	PDF of the tail distribution
$H(x)$	CDF of the tail distribution
θ	scale parameter for the body distribution
Ω	set of nonscale parameters for the body distribution
ξ	shape parameter for the GPD tail distribution
x_r	normalized value of the response variable at which the tail starts
p_n	mixing probability

Given these notations, the PDF $f(x)$ and the CDF $F(x)$ of the mixed-tail distribution are defined as

$$f(x) = \begin{cases} \frac{p_n}{G(x_b)} g(x) & \text{if } x \leq x_b \\ (1 - p_n) h(x - x_b) & \text{if } x > x_b \end{cases}$$

$$F(x) = \begin{cases} \frac{p_n}{G(x_b)} G(x) & \text{if } x \leq x_b \\ p_n + (1 - p_n) H(x - x_b) & \text{if } x > x_b \end{cases}$$

where $x_b = \theta x_r$ is the value of the response variable at which the tail starts.

These definitions indicate the following:

- The body distribution is conditional on $X \leq x_b$, where X denotes the random response variable.
- The tail distribution is the generalized Pareto distribution of the $(X - x_b)$ values.
- The probability that a response variable value belongs to the body is p_n. Consequently the probability that the value belongs to the tail is $(1 - p_n)$.

The parameters of this distribution are θ, Ω, ξ, x_r, and p_n. The scale of the GPD tail distribution θ_t is computed as

$$\theta_t = \frac{G(x_b; \theta, \Omega)}{g(x_b; \theta, \Omega)} \frac{(1 - p_n)}{p_n} = \theta \frac{G(x_r; \theta = 1, \Omega)}{g(x_r; \theta = 1, \Omega)} \frac{(1 - p_n)}{p_n}$$

The parameter x_r is typically estimated using a tail index estimation algorithm. One such algorithm is the Hill's algorithm (Danielsson et al. 2001), which is implemented by the predefined utility function SVRTUTIL_HILLCUTOFF available to you in the Sashelp.Svrtdist library. The algorithm and the utility function are described in detail in the section "Predefined Utility Functions" on page 436. The function computes an estimate of x_b, which can be used to compute an estimate of x_r because $x_r = x_b / \hat{\theta}$, where $\hat{\theta}$ is the estimate of the scale parameter of the body distribution.

The parameter p_n is typically determined by the domain expert based on the fraction of losses that are expected to belong to the tail.

The following SAS statements define the LOGNGPD distribution model for a mixed-tail distribution with the lognormal distribution as the body distribution and GPD as the tail distribution:

```
/*------- Define Lognormal Body-GPD Tail Mixed Distribution -------*/
proc fcmp library=sashelp.svrtdist outlib=work.sevexmpl.models;
   function LOGNGPD_DESCRIPTION() $256;
      length desc $256;
      desc1 = "Lognormal Body-GPD Tail Distribution.";
      desc2 = " Mu, Sigma, and Xi are free parameters.";
      desc3 = " Xr and Pn are constant parameters.";
      desc = desc1 || desc2 || desc3;
      return(desc);
   endsub;

   function LOGNGPD_SCALETRANSFORM() $3;
      length xform $3;
      xform = "LOG";
      return (xform);
   endsub;

   subroutine LOGNGPD_CONSTANTPARM(Xr,Pn);
   endsub;
```

```
function LOGNGPD_PDF(x, Mu,Sigma,Xi,Xr,Pn);
   cutoff = exp(Mu) * Xr;
   p = CDF('LOGN',cutoff, Mu, Sigma);
   if (x < cutoff + constant('MACEPS')) then do;
      return ((Pn/p)*PDF('LOGN', x, Mu, Sigma));
   end;
   else do;
      gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
      h = (1+Xi*(x-cutoff)/gpd_scale)**(-1-(1/Xi))/gpd_scale;
      return ((1-Pn)*h);
   end;
endsub;

function LOGNGPD_CDF(x, Mu,Sigma,Xi,Xr,Pn);
   cutoff = exp(Mu) * Xr;
   p = CDF('LOGN',cutoff, Mu, Sigma);
   if (x < cutoff + constant('MACEPS')) then do;
      return ((Pn/p)*CDF('LOGN', x, Mu, Sigma));
   end;
   else do;
      gpd_scale = p*((1-Pn)/Pn)/PDF('LOGN', cutoff, Mu, Sigma);
      H = 1 - (1 + Xi*((x-cutoff)/gpd_scale))**(-1/Xi);
      return (Pn + (1-Pn)*H);
   end;
endsub;

subroutine LOGNGPD_PARMINIT(dim,x[*],nx[*],F[*],Ftype,
                   Mu,Sigma,Xi,Xr,Pn);
   outargs Mu,Sigma,Xi,Xr,Pn;
   array xe[1] / nosymbols;
   array nxe[1] / nosymbols;

   eps = constant('MACEPS');

   Pn = 0.8; /* Set mixing probability */
   _status_ = .;
   call streaminit(56789);
   Xb = svrtutil_hillcutoff(dim, x, 100, 25, _status_);
   if (missing(_status_) or _status_ = 1) then
      Xb = svrtutil_percentile(Pn, dim, x, F, Ftype);

   /* prepare arrays for excess values */
   i = 1;
   do while (i <= dim and x[i] < Xb+eps);
      i = i + 1;
   end;
   dime = dim-i+1;
   call dynamic_array(xe, dime);
   call dynamic_array(nxe, dime);
   j = 1;
   do while(i <= dim);
      xe[j] = x[i] - Xb;
      nxe[j] = nx[i];
```

```
            i = i + 1;
            j = j + 1;
         end;

         /* Initialize lognormal parameters */
         call logn_parminit(dim, x, nx, F, Ftype, Mu, Sigma);
         if (not(missing(Mu))) then
            Xr = Xb/exp(Mu);
         else
            Xr = .;

         /* Initialize GPD's shape parameter using excess values */
         call gpd_parminit(dime, xe, nxe, F, Ftype, theta_gpd, Xi);
      endsub;

      subroutine LOGNGPD_LOWERBOUNDS(Mu,Sigma,Xi,Xr,Pn);
         outargs Mu,Sigma,Xi,Xr,Pn;

         Mu    = .; /* Mu has no lower bound */
         Sigma = 0; /* Sigma > 0 */
         Xi    = 0; /* Xi > 0 */
      endsub;
   quit;
```

Note the following points about the LOGNGPD definition:

- The parameters x_r and p_n are not estimated with the maximum likelihood method used by PROC HPSEVERITY, so you need to specify them as *constant* parameters by defining the *dist*_CONSTANTPARM subroutine. The signature of LOGNGPD_CONSTANTPARM subroutine lists only the constant parameters *Xr* and *Pn*.

- The parameter x_r is estimated by first using the SVRTUTIL_HILLCUTOFF utility function to compute an estimate of the cutoff point $\hat{x}_b$ and then computing $x_r = \hat{x}_b/e^{\hat{\mu}}$. If SVRTUTIL_HILLCUTOFF fails to compute a valid estimate, then the SVRTUTIL_PERCENTILE utility function is used to set $\hat{x}_b$ to the p_nth percentile of the data. The parameter p_n is fixed to 0.8.

- The Sashelp.Svrtdist library is specified with the LIBRARY= option in the PROC FCMP statement to enable the LOGNGPD_PARMINIT subroutine to use the predefined utility functions (SVRTUTIL_HILLCUTOFF and SVRTUTIL_PERCENTILE) and parameter initialization subroutines (LOGN_PARMINIT and GPD_PARMINIT).

- The LOGNGPD_LOWERBOUNDS subroutine defines the lower bounds for all parameters. This subroutine is required because the parameter *Mu* has a non-default lower bound. The bounds for *Sigma* and *Xi* must be specified. If they are not specified, they are returned as missing values, which PROC HPSEVERITY interprets as having no lower bound. You need not specify any bounds for the constant parameters *Xr* and *Pn*, because they are not subject to optimization.

The following DATA step statements simulate a sample from a mixed-tail distribution with a lognormal body and GPD tail. The parameter p_n is fixed to 0.8, the same value used in the LOGNGPD_PARMINIT subroutine defined previously.

```
/*----- Simulate a sample for the mixed-tail distribution -----*/
data testmixdist(keep=y label='Lognormal Body-GPD Tail Sample');
   call streaminit(45678);
   label y='Response Variable';
   N = 100;
   Mu = 1.5;
   Sigma = 0.25;
   Xi = 1.5;
   Pn = 0.8;

   /* Generate data comprising the lognormal body */
   Nbody = N*Pn;
   do i=1 to Nbody;
      y = exp(Mu) * rand('LOGNORMAL')**Sigma;
      output;
   end;

   /* Generate data comprising the GPD tail */
   cutoff = quantile('LOGNORMAL', Pn, Mu, Sigma);
   gpd_scale = (1-Pn) / pdf('LOGNORMAL', cutoff, Mu, Sigma);
   do i=Nbody+1 to N;
      y = cutoff + ((1-rand('UNIFORM'))**(-Xi) - 1)*gpd_scale/Xi;
      output;
   end;
run;
```

The following statements use PROC HPSEVERITY to fit the LOGNGPD distribution model to the simulated sample. They also fit three other predefined distributions (BURR, LOGN, and GPD). The final parameter estimates are written to the Work.Parmest data set.

```
/*--- Set the search path for functions defined with PROC FCMP ---*/
options cmplib=(work.sevexmpl);

/*-------- Fit LOGNGPD model with PROC HPSEVERITY --------*/
proc hpseverity data=testmixdist print=all outest=parmest;
   loss y;
   dist logngpd burr logn gpd;
run;
```

Some of the results prepared by PROC HPSEVERITY are shown in Output 14.3.1 and Output 14.3.2. The "Model Selection Table" in Output 14.3.1 indicates that all models converged. The last table in Output 14.3.1 shows that the model with LOGNGPD distribution has the best fit according to almost all the statistics of fit. The Burr distribution model is the closest contender to the LOGNGPD model, but the GPD distribution model fits the data very poorly.

Output 14.3.1 Summary of Fitting Mixed-Tail Distribution

```
                    The HPSEVERITY Procedure

                         Input Data Set

             Name                     WORK.TESTMIXDIST
             Label    Lognormal Body-GPD Tail Sample
```

Output 14.3.1 *continued*

```
                              Model Selection Table

          Distribution    Converged    -2 Log Likelihood    Selected

          logngpd         Yes                  418.78232    Yes
          Burr            Yes                  424.93728    No
          Logn            Yes                  459.43471    No
          Gpd             Yes                  558.13444    No

                             All Fit Statistics Table

                          -2 Log
   Distribution       Likelihood          AIC            AICC            BIC

   logngpd            418.78232*     428.78232*     429.42062*     441.80817
   Burr               424.93728      430.93728      431.18728      438.75280*
   Logn               459.43471      463.43471      463.55842      468.64505
   Gpd                558.13444      562.13444      562.25815      567.34478
```

The detailed results for the LOGNGPD distribution are shown in Output 14.3.2. The initial values table indicates the values computed by LOGNGPD_PARMINIT subroutine for the *Xr* and *Pn* parameters. It also uses the bounds columns to indicate the constant parameters. The last table in the figure shows the final parameter estimates. The estimates of all free parameters are significantly different than 0. As expected, the final estimates of the constant parameters *Xr* and *Pn* have not changed from their initial values.

Output 14.3.2 Detailed Results for the LOGNGPD Distribution

```
                             The HPSEVERITY Procedure

                             Distribution Information

Name                                                                    logngpd
Description               Lognormal Body-GPD Tail Distribution. Mu, Sigma, and Xi
                          are free parameters. Xr and Pn are constant parameters.
Number of Distribution                                                        5
Parameters

                       Initial Parameter Values and Bounds
                            for logngpd Distribution

                              Initial          Lower          Upper
              Parameter         Value          Bound          Bound

              Mu              1.49954         -Infty          Infty
              Sigma           0.76306     1.05367E-8          Infty
              Xi              0.36661     1.05367E-8          Infty
              Xr              1.27395       Constant       Constant
              Pn              0.80000       Constant       Constant

                    Convergence Status for logngpd Distribution

                  Convergence criterion (GCONV=1E-8) satisfied.
```

Output 14.3.2 *continued*

```
                  Optimization Summary for logngpd Distribution

                Optimization Technique                   Trust Region
                Number of Iterations                               11
                Number of Function Evaluations                     33
                Number of Failed Function Evaluations               1
                Log Likelihood                             -209.39116

                   Parameter Estimates for logngpd Distribution

                                           Standard                    Approx
             Parameter       Estimate         Error     t Value     Pr > |t|

             Mu               1.57921       0.06426       24.57       <.0001
             Sigma            0.31868       0.04459        7.15       <.0001
             Xi               1.03771       0.38205        2.72       0.0078
             Xr               1.27395      Constant         .           .
             Pn               0.80000      Constant         .           .
```

The following SAS statements use the parameter estimates to compute the value where the tail region is estimated to start ($x_b = e^{\hat{\mu}} \hat{x_r}$) and the scale of the GPD tail distribution ($\theta_t = \frac{G(x_b)}{g(x_b)} \frac{(1-p_n)}{p_n}$):

```
/*-------- Compute tail cutoff and tail distribution's scale --------*/
data xb_thetat(keep=x_b theta_t);
   set parmest(where=(_MODEL_='logngpd' and _TYPE_='EST'));
   x_b = exp(Mu) * Xr;
   theta_t = (CDF('LOGN',x_b,Mu,Sigma)/PDF('LOGN',x_b,Mu,Sigma)) *
             ((1-Pn)/Pn);
run;

proc print data=xb_thetat noobs;
run;
```

Output 14.3.3 Start of the Tail and Scale of the GPD Tail Distribution

```
                                x_b      theta_t

                            6.18005      1.27865
```

The computed values of x_b and θ_t are shown as *x_b* and *theta_t* in Output 14.3.3. Equipped with this additional derived information, you can now interpret the results of fitting the mixed-tail distribution as follows:

- The tail starts at $y \approx 6.18$. The primary benefit of using the scale-normalized cutoff (x_r) as the constant parameter instead of using the actual cutoff (x_b) is that the absolute cutoff is optimized by virtue of optimizing the scale of the body region ($\theta = e^{\mu}$).
- The values $y \leq 6.18$ follow the lognormal distribution with parameters $\mu \approx 1.58$ and $\sigma \approx 0.32$. These parameter estimates are reasonably close to the parameters used for simulating the sample.
- The values $y_t = y - 6.18$ ($y_t > 0$) follow the GPD distribution with scale $\theta_t \approx 1.28$ and shape $\xi \approx 1.04$.

Example 14.4: Fitting a Scaled Tweedie Model with Regressors

The Tweedie distribution is often used in the insurance industry to explain the effect of independent variables (regressors) on the distribution of losses. PROC HPSEVERITY provides a predefined scaled Tweedie distribution (STWEEDIE) that enables you to model the regression effects on the scale parameter. The scale regression model has its own advantages such as the ability to easily account for inflation effects. This example illustrates how that model can be used to evaluate the effect of regressors on the *mean* of the Tweedie distribution, which is useful in problems such rate-making and pure premium modeling.

Assume a Tweedie process, whose mean μ is affected by k regressors x_j, $j = 1, \ldots, k$ as follows:

$$\mu = \mu_0 \exp\left(\sum_{j=1}^{k} \beta_j x_j\right)$$

where μ_0 represents the base value of the mean (you can think of μ_0 as $\exp(\beta_0)$, where β_0 is the intercept). This model for the mean is identical to the popular generalized linear model for the mean with a logarithmic link function. More interestingly, it parallels the model used by PROC HPSEVERITY for the scale parameter θ,

$$\theta = \theta_0 \exp\left(\sum_{j=1}^{k} \beta_j x_j\right)$$

where θ_0 represents the base value of the scale parameter. As described in the section "Tweedie Distributions" on page 402, for the parameter range $p \in (1, 2)$, the mean of the Tweedie distribution is given by

$$\mu = \theta\lambda\frac{2-p}{p-1}$$

where λ is the Poisson mean parameter of the scaled Tweedie distribution. This relationship enables you to use the scale regression model to infer the effect of regressors on the mean of the distribution.

Let the data set Work.Test_Sevtw contain a sample generated from a Tweedie distribution with dispersion parameter $\phi = 0.5$, index parameter $p = 1.75$, and the mean parameter that is affected by three regression variables x1, x2, and x3 as follows:

$$\mu = 5\ \exp(0.25\ \text{x1} - \text{x2} + 3\ \text{x3})$$

Thus, the population values of regression parameters are $\mu_0 = 5$, $\beta_1 = 0.25$, $\beta_2 = -1$, and $\beta_3 = 3$. You can find the code used to generate the sample in the PROC HPSEVERITY sample program *hpseve04.sas*.

The following PROC HPSEVERITY step uses the sample in Work.Test_Sevtw data set to estimate the parameters of the scale regression model for the predefined scaled Tweedie distribution (STWEEDIE) with the dual quasi-Newton (QUANEW) optimization technique:

```
proc hpseverity data=test_sevtw outest=estw covout print=all;
   loss y;
   scalemodel x1-x3;

   dist stweedie;
   nloptions tech=quanew;
run;
```

The dual quasi-Newton technique is used because it requires only the first-order derivatives of the objective function, and it is harder to compute reasonably accurate estimates of the second-order derivatives of Tweedie distribution's PDF with respect to the parameters.

Some of the key results prepared by PROC HPSEVERITY are shown in Output 14.4.1 and Output 14.4.2. The distribution information and the convergence results are shown in Output 14.4.1.

Output 14.4.1 Convergence Results for the STWEEDIE Model with Regressors

```
                         The HPSEVERITY Procedure

                         Distribution Information

Name                                                             stweedie
Description                           Tweedie Distribution with Scale Parameter
Number of Distribution Parameters                                       3
Number of Regression Parameters                                         3

                  Convergence Status for stweedie Distribution

              Convergence criterion (FCONV=2.220446E-16) satisfied.

                 Optimization Summary for stweedie Distribution

             Optimization Technique            Dual Quasi-Newton
             Number of Iterations                             40
             Number of Function Evaluations                  136
             Log Likelihood                              -1044.3
```

The final parameter estimates of the STWEEDIE regression model are shown in Output 14.4.2. The estimate that is reported for the parameter Theta is the estimate of the base value θ_0. The estimates of regression coefficients β_1, β_2, and β_3 are indicated by the rows of x1, x2, and x3, respectively.

Output 14.4.2 Parameter Estimates for the STWEEDIE Model with Regressors

```
                 Parameter Estimates for stweedie Distribution

                                    Standard                    Approx
          Parameter      Estimate      Error    t Value    Pr > |t|

          Theta           0.82763    0.31221       2.65      0.0085
          Lambda         16.47266   15.51146       1.06      0.2891
          P               1.75287    0.24175       7.25      <.0001
          x1              0.27933    0.09884       2.83      0.0050
          x2             -0.76746    0.10323      -7.43      <.0001
          x3              3.03188    0.10149      29.87      <.0001
```

If your goal is to explain the effect of regressors on the scale parameter, then the output displayed in Output 14.4.2 is sufficient. But, if you want to compute the effect of regressors on the mean of the distribution, then you need to do some postprocessing. Using the relationship between μ and θ, μ can be written in terms of the parameters of the STWEEDIE model as

$$\mu = \theta_0 \exp\left(\sum_{j=1}^{k} \beta_j x_j\right) \lambda \frac{2-p}{p-1}$$

This shows that the parameters β_j are identical for the mean and the scale model, and the base value μ_0 of the mean model is

$$\mu_0 = \theta_0 \lambda \frac{2-p}{p-1}$$

The estimate of μ_0 and the standard error associated with it can be computed by using the property of the functions of maximum likelihood estimators (MLE). If $g(\Omega)$ represents a totally differentiable function of parameters Ω, then the MLE of g has an asymptotic normal distribution with mean $g(\hat{\Omega})$ and covariance $C = (\partial \mathbf{g})' \Sigma (\partial \mathbf{g})$, where $\hat{\Omega}$ is the MLE of Ω, Σ is the estimate of covariance matrix of Ω, and $\partial \mathbf{g}$ is the gradient vector of g with respect to Ω evaluated at $\hat{\Omega}$. For μ_0, the function is $g(\Omega) = \theta_0 \lambda (2-p)/(p-1)$. The gradient vector is

$$\begin{aligned} \partial \mathbf{g} &= \left(\frac{\partial g}{\partial \theta_0} \quad \frac{\partial g}{\partial \lambda} \quad \frac{\partial g}{\partial p} \quad \frac{\partial g}{\partial \beta_1} \cdots \frac{\partial g}{\partial \beta_k} \right) \\ &= \left(\frac{\mu_0}{\theta_0} \quad \frac{\mu_0}{\lambda} \quad \frac{-\mu_0}{(p-1)(2-p)} \quad 0 \ldots 0 \right) \end{aligned}$$

You can write a DATA step that implements these computations by using the parameter and covariance estimates prepared by PROC HPSEVERITY step. The DATA step program is available in the sample program *sevex04.sas*. The estimates of μ_0 prepared by that program are shown in Output 14.4.3. These estimates and the estimates of β_j as shown in Output 14.4.2 are reasonably close (that is, within one or two standard errors) to the parameters of the population from which the sample in Work.Test_Sevtw data set was drawn.

Output 14.4.3 Estimate of the Base Value Mu0 of the Mean Parameter

```
                                   Standard                    Approx
              Parameter    Estimate       Error    t Value    Pr > |t|

                 Mu0        4.47503     0.42292    10.5813           0
```

Another effect of using the scaled Tweedie distribution to model the regression effects is that the regressors also affect the variance V of the Tweedie distribution. The variance is related to the mean as $V = \phi\mu^p$, where ϕ is the dispersion parameter. Using the relationship between the parameters TWEEDIE and STWEEDIE distributions as described in the section "Tweedie Distributions" on page 402, the regression model for the dispersion parameter is

$$\begin{aligned}\log(\phi) &= (2-p)\log(\mu) - \log(\lambda(2-p)) \\ &= ((2-p)\log(\mu_0) - \log(\lambda(2-p))) + (2-p)\sum_{j=1}^{k} \beta_j x_j\end{aligned}$$

Subsequently, the regression model for the variance is

$$\begin{aligned}\log(V) &= 2\log(\mu) - \log(\lambda(2-p)) \\ &= (2\log(\mu_0) - \log(\lambda(2-p))) + 2\sum_{j=1}^{k} \beta_j x_j\end{aligned}$$

In summary, PROC HPSEVERITY enables you to estimate regression effects on various parameters and statistics of the Tweedie model.

Example 14.5: Fitting Distributions to Interval-Censored Data

In some applications, the data available for modeling might not be exact. A commonly encountered scenario is the use of grouped data from an external agency, which for several reasons, including privacy, does not provide information about individual loss events. The losses are grouped into disjoint bins, and you know only the range and number of values in each bin. Each group is essentially interval-censored, because you know that a loss magnitude is in certain interval, but you do not know the exact magnitude. This example illustrates how you can use PROC HPSEVERITY to model such data.

The following DATA step generates sample grouped data for dental insurance claims, which is taken from Klugman, Panjer, and Willmot (1998):

```
/* Grouped dental insurance claims data
     (Klugman, Panjer, and Willmot, 1998) */
data gdental;
   input lowerbd upperbd count @@;
   datalines;
0 25 30  25 50 31  50 100 57  100 150 42  150 250 65  250 500 84
500 1000 45  1000 1500 10  1500 2500 11  2500 4000 3
;
run;
```

The following PROC HPSEVERITY step fits all the predefined distributions to the data in Work.Gdental data set:

```
/* Fit all predefined distributions */
proc hpseverity data=gdental print=all criterion=aicc;
   loss / rc=lowerbd lc=upperbd;
   weight count;
   dist _predef_;
   performance nthreads=1;
run;
```

The LOSS statement specifies the left and right boundaries of each group as the right-censoring and left-censoring limits, respectively. The variable count records the number of losses in each group and is specified in the WEIGHT statement. Note that no response variable is specified in the LOSS statement, which is allowed as long as each observation in the input data set is censored. For interval-censored data, the EDF estimates that are used for initializing the parameters are computed first by replacing each interval-censored observation with an uncensored observation that has a loss value equal to the average of the left- and right-censoring limits and then by using either the standard or the Kaplan-Meier algorithm. The PERFORMANCE statement specifies that just one thread of execution be used, to minimize the overhead associated with multithreading, because the input data set is very small.

Some of the key results prepared by PROC HPSEVERITY are shown in Output 14.5.1. According to the "Model Selection Table" in Output 14.5.1, all distribution models have converged. The "All Fit Statistics Table" in Output 14.5.1 indicates that the exponential distribution (EXP) has the best fit for data according to a majority of the likelihood-based statistics.

Output 14.5.1 Statistics of Fit for Interval-Censored Data

```
                        The HPSEVERITY Procedure

                             Input Data Set

                          Name    WORK.GDENTAL

                          Model Selection Table

                                        Corrected
                                         Akaike's
                                       Information
           Distribution    Converged      Criterion    Selected

           Burr            Yes             46.63302    No
           Exp             Yes             39.64750    Yes
           Gamma           Yes             43.03581    No
           Igauss          Yes             41.29497    No
           Logn            Yes             41.50538    No
           Pareto          Yes             42.74001    No
           Gpd             Yes             42.73995    No
           Weibull         Yes             43.07064    No
```

Output 14.5.1 *continued*

```
                      All Fit Statistics Table

                 -2 Log
Distribution   Likelihood        AIC           AICC           BIC

Burr            35.83302      41.83302      46.63302      42.42469
Exp             37.07607      39.07607*     39.64750*     39.27329*
Gamma           37.03581      41.03581      43.03581      41.43026
Igauss          35.29497*     39.29497      41.29497      39.68942
Logn            35.50538      39.50538      41.50538      39.89983
Pareto          36.74001      40.74001      42.74001      41.13446
Gpd             36.73995      40.73995      42.73995      41.13440
Weibull         37.07064      41.07064      43.07064      41.46509
```

Example 14.6: Benefits of Distributed and Multithreaded Computing

One of the key features of the HPSEVERITY procedure is that is takes advantage of the distributed and multithreaded computing machinery in order to solve a given problem faster. This example illustrates the benefits of using multithreading and distributed computing.

The example uses a simulated data set Work.Largedata, which contains 10,000,000 observations, some of which are right-censored or left-truncated. The losses are affected by three external effects. The DATA step program that generates this data set is available in the accompanying sample program *hpseve06.sas*.

The following PROC HPSEVERITY step fits all the predefined distributions to the data in Work.Largedata data set on the client machine with just one thread of computation:

```
/* Fit all predefined distributions without any multithreading or
   distributed computing */
proc hpseverity data=largedata criterion=aicc initsample(size=20000);
   loss y / lt=threshold rc=limit;
   scalemodel x1-x3;
   dist _predef_;
   performance nthreads=1 bufsize=1000000 details;
run;
```

The NTHREADS=1 option in the PERFORMANCE statement specifies that just one thread of computation be used. The absence of the NODES= option in the PERFORMANCE statement specifies that the client mode of execution be used. That is, this step does not use any multithreading or distributed computing. The BUFSIZE= option in the PERFORMANCE statement specifies the number of observations to read at one time. Specifying a larger value tends to decrease the time it takes to load the data. The DETAILS option in the performance statement enables generation of the Timing table. The INITSAMPLE option in the PROC HPSEVERITY statement specifies that a uniform random sample of maximum 20,000 observations be used for parameter initialization.

The PerformanceInfo and Timing tables prepared by PROC HPSEVERITY are shown in Output 14.6.1. The machine used to obtain these results has a Windows operating system and is powered by eight CPU cores, each with a clock speed of 3.4 GHz.

Output 14.6.1 Performance for Client Mode with No Multithreading

```
                    The HPSEVERITY Procedure

                    Performance Information

                 Execution Mode       On client
                 Number of Threads    1

                     Procedure Task Timing

                                              Time
Task                                        (sec.)

Load and Prepare Models                       0.54     0.02%
Load and Prepare Data                         2.28     0.10%
Initialize Parameters                         1.42     0.06%
Estimate Parameters                        2224.96     99.8%
```

The “Performance Information Table” contains the information about the execution environment. The “Procedure Task Timing” indicates the total time and relative time taken by each of the four main steps of PROC HPSEVERITY. As the “Procedure Task Timing” table shows, it takes around 37 minutes for the task of estimating parameters, which is typically the most time-consuming task of all the tasks.

If the grid appliance is not available, you can improve the performance by enabling multiple threads of computation, which is in fact the default. The following PROC HPSEVERITY step fits all the predefined distributions using all available CPU cores of the machine:

```
/* Fit all predefined distributions with multithreading, but no
   distributed computing */
proc hpseverity data=largedata criterion=aicc initsample(size=20000);
   loss y / lt=threshold rc=limit;
   scalemodel x1-x3;
   dist _predef_;
   performance bufsize=1000000 details;
run;
```

When you do not specify the NTHREADS= option in the PERFORMANCE statement, the HPSEVERITY procedure uses the value of the CPUCOUNT= system option to decide how many threads to use. By default, the CPUCOUNT= value is equal to the number of physical CPU cores present in the machine where the SAS software is running. This machine is the client node for the SMP mode of execution and the grid node for the MPP mode of execution. In this case, the default value is 8, which is reflected in the “Performance Information” table shown in Output 14.6.2.

Output 14.6.2 Performance for Client Mode with 8 Threads

```
                    The HPSEVERITY Procedure

                    Performance Information

                 Execution Mode       On client
                 Number of Threads    8
```

Output 14.6.2 *continued*

```
                        Procedure Task Timing

                                                     Time
             Task                                   (sec.)

             Load and Prepare Models                  0.57     0.11%
             Load and Prepare Data                    2.05     0.39%
             Initialize Parameters                    1.11     0.21%
             Estimate Parameters                    528.39     99.3%
```

The results in Output 14.6.2 indicate that threading improves the performance. The time to estimate parameters has been reduced to about 8.8 minutes.

When a grid appliance is available, the performance can be further improved by using more than one node in the MPP mode of execution. Typically, large data sets are distributed on the grid appliance that hosts a distributed database before the procedure is run. In other words, large problems are best suited for the alongside-the-database model of execution. However, for the purpose of illustration, this example assumes that the data set is available on the client machine and is then distributed to the grid nodes by the HPSEVERITY procedure according to the options that are specified in the PERFORMANCE statement.

The next few PROC HPSEVERITY steps are run on a grid appliance by varying the number of nodes and the number of threads that are used within each node.

First, the following statements are submitted to specify the appliance host (GRIDHOST= system option) and the install location of High-Performance Analytics shared libraries on the appliance (GRIDINSTALLLOC= system option):

```
option set=GRIDHOST      ="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";
```

To run the preceding statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to macro variables with the appropriate values. For more information about the GRIDHOST= and GRIDINSTALLLOC= options, see the section "PERFORMANCE Statement" on page 29.

In order to establish a reference point for the performance, the results of using just one node of the grid appliance without any multithreading are presented first. The particular grid appliance that is used to obtain these results has 16 nodes. Each node is running a 64-bit Linux OS and has 24 CPU cores with clock speed of 3.33 GHz. The following PROC HPSEVERITY step fits all the predefined distributions to the data in the Work.Largedata data set using just one node for computation and one computation thread within that node:

```
/* Fit all predefined distributions on 1 grid node without
   any multithreading */
proc hpseverity data=largedata criterion=aicc initsample(size=20000);
   loss y / lt=threshold rc=limit;
   scalemodel x1-x3;
   dist _predef_;
   performance nthreads=1 nodes=1 details;
run;
```

The PERFORMANCE statement specifies that just one node be used to fit the models with just one thread of computation on that node. The "Performance Information Table" and "Procedure Timing Table" prepared by PROC HPSEVERITY are shown in Output 14.6.3. It takes around 37 minutes to complete the task of estimating parameters.

The computations and time taken to fit each model is also shown in the "Estimation Details Table," which is generated whenever you specify the DETAILS option in the PERFORMANCE statement. This table can be useful for comparing the relative effort required to fit each model and draw some broader conclusions. For example, even if the Pareto distribution takes a larger number of iterations, function calls, gradient, and Hessian updates than the gamma distribution, it takes less time to complete, which indicates that the individual PDF and CDF computations for the gamma distribution are more expensive than that of Pareto distribution.

Output 14.6.3 Performance on One Grid Appliance Node with No Multithreading

```
                                 The HPSEVERITY Procedure

                                 Performance Information

                        Host Node                     << your grid host >>
                        Execution Mode                Distributed
                        Number of Compute Nodes       1
                        Number of Threads per Node    1

                                 Estimation Details Table
                             Optimization Technique: Trust Region

                                            Function  Gradient   Hessian       Time
Distribution  Converged  Iterations            Calls   Updates   Updates      (sec.)

Burr          Yes                10               25        90        77      240.08
Exp           Yes                 3               10        20        14       23.26
Gamma         Yes                 5               14        35        27      811.86
Igauss        Yes                 4               13        27        20      374.01
Logn          Yes                 4               12        27        20      129.24
Pareto        Maybe              50              180      1430      1377      498.15
Gpd           Yes                 4               12        27        20       79.00
Weibull       Yes                 4               12        27        20       67.71

                                  Procedure Task Timing

                                                            Time
          Task                                             (sec.)

          Load and Prepare Models                           0.68     0.03%
          Load and Prepare Data                             0.90     0.04%
          Initialize Parameters                             1.42     0.06%
          Estimate Parameters                            2223.22     99.9%
```

To obtain the next reference point for performance, the following PROC HPSEVERITY step specifies that 24 computation threads be used on one node of the grid appliance:

```
/* Fit all predefined distributions on 1 grid node with multithreading */
proc hpseverity data=largedata criterion=aicc initsample(size=20000);
   loss y / lt=threshold rc=limit;
   scalemodel x1-x3;
   dist _predef_;
   performance nthreads=24 nodes=1 details;
run;
```

The performance tables prepared by the preceding statements are shown in Output 14.6.4. As the "Procedure Task Timing" table shows, use of multithreading has improved the performance significantly over the single-threaded case. Now, it takes around 6.15 minutes to complete the task of estimating parameters.

Output 14.6.4 Performance Information with Multithreading, but No Distributed Computing

```
                         The HPSEVERITY Procedure

                         Performance Information

                Host Node                     << your grid host >>
                Execution Mode                Distributed
                Number of Compute Nodes       1
                Number of Threads per Node    24

                          Procedure Task Timing

                                                  Time
         Task                                    (sec.)

         Load and Prepare Models                   0.48     0.13%
         Load and Prepare Data                     0.62     0.17%
         Initialize Parameters                     1.30     0.35%
         Estimate Parameters                     368.80     99.4%
```

The power of multithreading and distributed computing can be combined by specifying that multiple nodes of the grid and all available threads of execution within each node be used to accomplish the task. The following PROC HPSEVERITY step specifies that all 16 nodes of the grid appliance be used with 24 parallel computation threads on each node:

```
/* Fit all predefined distributions with distributed computing and
   multithreading within each node */
proc hpseverity data=largedata criterion=aicc initsample(size=20000);
   loss y / lt=threshold rc=limit;
   scalemodel x1-x3;
   dist _predef_;
   performance nthreads=24 nodes=16 details;
run;
```

The performance tables prepared by the preceding statements are shown in Output 14.6.5. These tables can be compared to the tables in Output 14.6.3 and Output 14.6.4 to determine the benefit of combining multithreading and distributed computed. A problem that would have taken around 37 minutes on a single node with no multithreading can now be solved in around 35 seconds by using the full computational resources of the grid appliance.

Output 14.6.5 Performance Information with Distributed Computing and Multithreading

```
                              The HPSEVERITY Procedure

                              Performance Information

                     Host Node                     << your grid host >>
                     Execution Mode                Distributed
                     Number of Compute Nodes       16
                     Number of Threads per Node    24

                               Procedure Task Timing

                                                       Time
              Task                                    (sec.)

              Load and Prepare Models                   0.69     1.90%
              Load and Prepare Data                     0.05     0.12%
              Initialize Parameters                     1.23     3.37%
              Estimate Parameters                      34.39     94.6%
```

References

D'Agostino, R. and Stephens, M. (1986), *Goodness-of-Fit Techniques,* New York: Marcel Dekker, Inc.

Danielsson, J., De Haan, L., Peng, L., and de Vries, C. G. (2001), "Using a Bootstrap Method to Choose the Sample Fraction in Tail Index Estimation," *Journal of Multivariate Analysis,* 76, 226–248.

Dunn, P. K. and Smyth, G. K. (2005), "Series Evaluation of Tweedie Exponential Dispersion Model Densities," *Statistics and Computing,* 15(4), 267–280.

Frydman, H. (1994), "A Note on Nonparametric Estimation of the Distribution Function from Interval-Censored and Truncated Observations," *Journal of Royal Statistical Society, Series B,* 56(1), 71–74.

Gentleman, R. and Geyer, C. J. (1994), "Maximum Likelihood for Interval Censored Data: Consistency and Computation," *Biometrika,* 81(3), 618–623.

Hill, B. M. (1975), "A Simple General Approach to Inference about the Tail of a Distribution," *Annals of Statistics,* 3(5), 1163–1174.

Jørgensen, B. (1987), "Exponential Dispersion Models (with Discussion)," *Journal of Royal Statistical Society, Series B,* 49(2), 127–162.

Kaplan, E. L. and Meier, P. (1958), "Nonparametric Estimation from Incomplete Observations," *Journal of American Statistical Association,* 53, 457–481.

Klein, J. P. and Moeschberger, M. L. (1997), *Survival Analysis: Techniques for Censored and Truncated Data,* New York: Springer-Verlag.

Klugman, S. A., Panjer, H. H., Willmot, G. E. (1998), *Loss Models: From Data to Decisions,* New York: John Wiley & Sons.

Lai, T. L. and Ying, Z. (1991), "Estimating A Distribution Function with Truncated and Censored Data," *Annals of Statistics,* 19(1), 417–442.

Lynden-Bell, D. (1971), "A Method of Allowing for Known Observational Selection in Small Samples Applied to 3CR Quasars," *Monthly Notices of the Royal Astronomical Society,* 155, 95–118.

Tweedie, M. C. K. (1984), "An Index Which Distinguishes between Some Important Exponential Families," *Statistics: Applications and New Directions, Proceedings of the Indian Statistical Institute Golden Jubilee International Conference, J. K. Ghosh and J. Roy (Eds.),* 579–604.

Chapter 15

The HPSUMMARY Procedure

Contents

Overview: HPSUMMARY Procedure

The HPSUMMARY procedure enables you to summarize data on a SAS High Performance Analytics grid for parallel execution. The output data created by PROC HPSUMMARY can then be written in parallel back to the grid data store.

Although it is possible to use PROC HPSUMMARY on data that do not reside on the grid or produce result tables that do not reside on the grid, this usage is not recommended for production due to the overhead of transferring data to and from the grid.

PROC HPSUMMARY is a high-performance version of the SUMMARY procedure in Base SAS software. For more information about the SUMMARY procedure, see the *Base SAS 9.3 Procedures Guide*.

PROC HPSUMMARY Features

PROC HPSUMMARY provides data summarization tools to compute descriptive statistics for variables across all observations and within groups of observations. For example, PROC HPSUMMARY does the following:

- calculates descriptive statistics based on moments
- calculates and estimates quantiles, which includes the median
- calculates confidence limits for the mean
- identifies extreme values
- performs a *t* test

PROC HPSUMMARY does not display output. You can use the OUTPUT statement to store the statistics in a SAS data set.

PROC HPSUMMARY provides a vehicle for the parallel execution of summarization in a distributed computing environment. The following list summarizes the basic features of PROC HPSUMMARY:

- provides the ability to execute summarization in parallel
- enables you to control the level of parallelism per execution node and the number of nodes to engage
- is highly multithreaded
- manages data migration to the location of execution and movement back to the client machine as needed

Client and Grid Execution Modes

With the HPSUMMARY procedure you can read and write data in distributed form and perform analyses in massively parallel processing (MPP) and symmetric multiprocessing (SMP) mode. See the section "SMP and MPP Modes" on page 10 in Chapter 2, "Shared Concepts and Topics," for details about how to affect the execution mode of SAS High-Performance Analytics procedures.

PROC HPSUMMARY controls the execution of summarization in two dimensions. You can control both the number of parallel threads per execution node and also the number of compute nodes to engage.

Alternatively, PROC HPSUMMARY can be executed on the High-Performance Analytics grid. In grid mode, one or more copies of the summarization code are executed in parallel on each grid node.

The grid mode of execution has two variations:

- In the client-data (local-data) model of grid execution, the input data are not stored on the appliance but are distributed to the distributed computing environment during execution of the HPSUMMARY procedure.
- In the alongside-the-database model of grid execution, the data source is the database on the appliance. The data are stored in the distributed database, and the summarization code running on each node can read and write the data in parallel during execution of the procedure. Instead of data being moved across the network and possibly back to the client machine, data are passed locally between the processes on each node of the appliance. In general, especially with large data sets, the best PROC HPSUMMARY performance can be achieved if execution is alongside the database.

PROC HPSUMMARY Contrasted with Other Procedures

By default, PROC SUMMARY generates all CLASS variable combination types and requires the NWAY option to generate only the n-way. By default, PROC HPSUMMARY generates only the n-way, and requires the ALLTYPES option to generate all of the types.

Getting Started: HPSUMMARY Procedure

This example illustrates a simple use of the HPSUMMARY procedure to summarize using the grid in MPP mode. The following DATA step creates a data set that consists of test scores:

```
data gridlib.grades;
   input Name $ 1-8 Gender $ 11 Status $13 Year $ 15-16
         Section $ 18 Score 20-21 FinalGrade 23-24;
   datalines;
Abbott    F 2 97 A 90 87
Branford  M 1 98 A 92 97
Crandell  M 2 98 B 81 71
Dennison  M 1 97 A 85 72
Edgar     F 1 98 B 89 80
Faust     M 1 97 B 78 73
Greeley   F 2 97 A 82 91
Hart      F 1 98 B 84 80
Isley     M 2 97 A 88 86
Jasper    M 1 97 B 91 93
;
run;
```

The following statements read this data set and analyze the data for the two-way combination of CLASS variables and across all observations. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to the macro variables in the example with the appropriate values.

```
proc hpsummary data=gridlib.grades;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   var Score;
   class Status Year;
   types () status*year;
   output out=gridlib.result;
run;
proc print data=gridlib.result;
run;
```

Figure 15.1 displays the table produced by the HPSUMMARY procedure. The "Performance Information" table shows that HPSUMMARY was executed alongside the database on the grid.

Figure 15.1 HPSUMMARY Output

```
                          Performance Information

             Host Node                      << your grid host >>
             Execution Mode                 Distributed
             Number of Compute Nodes        4
             Number of Threads per Node     8

        Obs    Status    Year    _TYPE_    _FREQ_    _STAT_     Score

          1                         0        10       N        10.0000
          2                         0        10       MIN      78.0000
          3                         0        10       MAX      92.0000
          4                         0        10       MEAN     86.0000
          5                         0        10       STD       4.7140
          6       1       98        3         3       N         3.0000
          7       1       98        3         3       MIN      84.0000
          8       1       98        3         3       MAX      92.0000
          9       1       98        3         3       MEAN     88.3333
         10       1       98        3         3       STD       4.0415
         11       2       97        3         3       N         3.0000
         12       2       97        3         3       MIN      82.0000
         13       2       97        3         3       MAX      90.0000
         14       2       97        3         3       MEAN     86.6667
         15       2       97        3         3       STD       4.1633
         16       2       98        3         1       N         1.0000
         17       2       98        3         1       MIN      81.0000
         18       2       98        3         1       MAX      81.0000
         19       2       98        3         1       MEAN     81.0000
         20       2       98        3         1       STD        .
         21       1       97        3         3       N         3.0000
         22       1       97        3         3       MIN      78.0000
         23       1       97        3         3       MAX      91.0000
         24       1       97        3         3       MEAN     84.6667
         25       1       97        3         3       STD       6.5064
```

Syntax: HPSUMMARY Procedure

The following statements are available in the HPSUMMARY procedure:

PROC HPSUMMARY < *options* > < *statistic-keywords* > **;**
  CLASS *variables* < / *options* > **;**
  FREQ *variable* **;**
  OUTPUT < **OUT=***SAS-data-set* > < *output-statistic-specifications* > < / **AUTONAME** > **;**
  PERFORMANCE *performance-options* **;**
  TYPES *requests* **;**
  VAR *variables* < / **WEIGHT=***weight-variable* > **;**
  WAYS *list* **;**
  WEIGHT *variable* **;**

You can also use the ATTRIB, FORMAT, LABEL, and WHERE statements and any global statements. For more information, see *SAS Statements: Reference*.

PROC HPSUMMARY Statement

PROC HPSUMMARY < *options* > < *statistic-keywords* > **;**

The PROC HPSUMMARY statement invokes the procedure. The HPSUMMARY procedure computes descriptive statistics for variables across all observations or within groups of observations.

Table 15.1 summarizes the available *options* in the PROC HPSUMMARY statement by function. The *options* are then described fully in alphabetical order in the section "Optional Arguments" on page 480. For information about the *statistic-keywords*, see the section "Statistic Keywords" on page 482.

Table 15.1 PROC HPSUMMARY Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
PCTLDEF=	Specifies the mathematical definition used to compute quantiles
Option Related to Classification Level	
MISSING	Uses missing values as valid values to create combinations of classification variables
Options Related to the Output Data Set	
ALLTYPES	Computes statistics for all combinations of classification variables (not just the n-way)
CHARTYPE	Specifies that the _TYPE_ variable contain character values

Table 15.1 *continued*

Option	Description
Options Related to Statistical Analysis	
ALPHA=	Specifies the confidence level for the confidence limits
EXCLNPWGT	Excludes observations with nonpositive weights from the analysis
QMARKERS=	Specifies the sample size to use for the P2 quantile estimation method
QMETHOD=	Specifies the quantile estimation method
QNTLDEF=	Specifies the mathematical definition used to compute quantiles
statistic-keywords	Selects the statistics
VARDEF=	Specifies the variance divisor

Optional Arguments

You can specify the following *options* in the PROC HPSUMMARY statement:

ALLTYPES

ALLWAYS

requests that PROC HPSUMMARY compute descriptive statistics for all combinations of classification variables. By default, PROC HPSUMMARY generates only the n-way. For more information, see the section "How PROC HPSUMMARY Groups Data" on page 491.

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is 100(1–*value*). For example, ALPHA=0.05 results in a 95% confidence limit. You can specify any *value* between 0 and 1. The default is 0.05. To compute confidence limits, specify the *statistic-keyword* CLM, LCLM, or UCLM. See the section "Confidence Limits" on page 493.

CHARTYPE

specifies that the _TYPE_ variable in the output data set is a character representation of the binary value of _TYPE_. The length of the variable equals the number of classification variables. When you specify more than 32 classification variables, _TYPE_ automatically becomes a character variable. See the section "Output Data Set" on page 496.

DATA=*SAS-data-set*

names the SAS data set to be used as the input data set. The default is the most recently created data set.

EXCLNPWGT

EXCLNPWGTS

excludes observations with nonpositive weight values (0 or negative) from the analysis. By default, PROC HPSUMMARY treats observations with negative weights like observations with zero weights and counts them in the total number of observations. See the WEIGHT= option and the section "WEIGHT Statement" on page 490.

MISSING

considers missing values as valid values to create the combinations of classification variables. Special missing values that represent numeric values—the letters A through Z and the underscore (_)

character—are each considered as a separate value. If you omit MISSING, then PROC HPSUMMARY excludes the observations with a missing classification variable value from the analysis. See *SAS Language Reference: Concepts* for a discussion of missing values that have special meanings.

PCTLDEF=1 | 2 | 3 | 4 | 5

is an alias for the QNTLDEF= option.

QMARKERS=*number*

specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

The value of *number* must be an odd integer greater than 3. The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quantiles (P25 and P50), *number* is 25. For the quantiles P1, P5, P10, P75 P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC HPSUMMARY uses the largest value of *number*.

You can improve the accuracy of the estimate by increasing the number of markers above the default settings; you can conserve memory and computing time by reducing the number of markers. See the section "Quantiles" on page 494.

QMETHOD=OS | P2

specifies the method that PROC HPSUMMARY uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results. The QMETHOD= option can take either of the following values:

OS specifies that PROC HPSUMMARY use order statistics.

NOTE: This technique can be very memory-intensive.

P2 specifies that PROC HPSUMMARY use the P^2 method to approximate the quantile. When QMETHOD=P2, PROC HPSUMMARY does not compute MODE or weighted quantiles. In addition, reliable estimations of some quantiles (P1, P5, P95, P99) might not be possible for some data sets.

The default is OS. See the section "Quantiles" on page 494 for more information.

QNTLDEF=1 | 2 | 3 | 4 | 5

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that PROC HPSUMMARY uses to calculate quantiles when QMETHOD=OS. The default is 5. To use QMETHOD=P2, you must use QNTLDEF=5. See the section "Quantile and Related Statistics" on page 501.

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. Table 15.2 shows the possible values for *divisor* and their associated formulas.

Table 15.2 Values for VARDEF= Option

divisor	**Description**	**Formula for Divisor**
DF	Degrees of freedom	$n - 1$
N	Number of observations	n

Table 15.2 (continued)

divisor	**Description**	**Formula for Divisor**
WDF	Sum of weights minus one	$\left(\sum_i w_i\right) - 1$
WEIGHT | WGT	Sum of weights	$\sum_i w_i$

The procedure computes the variance as CSS divided by *divisor*, where the corrected sum of squares CSS is defined by the following formula:

$$\text{CSS} = \sum (x_i - \bar{x})^2$$

When you weight the analysis variables, the formula for CSS is

$$\text{CSS} = \sum w_i\,(x_i - \bar{x}_w)^2$$

where $\bar{x}_w$ is the weighted mean.

The default is DF. To compute the standard error of the mean, confidence limits for the mean, or the Student's t-test, you must use this default value.

When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the ith observation is $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the ith observation. This method yields an estimate of the variance of an observation with unit weight. When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This method yields an asymptotic estimate of the variance of an observation with average weight. See the section "Keywords and Formulas" on page 497.

Statistic Keywords

Optional *statistic-keywords* specify which statistics to compute and the order to display them in the output. Table 15.3 lists the keywords that are available in the PROC HPSUMMARY statement. The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 497.

Table 15.3 Statistic Keywords in the PROC HPSUMMARY Statement

Descriptive Statistic Keywords:	
CLM	NMISS
CSS	RANGE
CV	SKEWNESS | SKEW
KURTOSIS | KURT	STDDEV | STD
LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR

Table 15.3 (continued)

Quantile Statistic Keywords:	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
P20	P30
P40	P60
P70	P80
Q1 | P25	QRANGE
Hypothesis Testing Keywords:	
PROBT | PRT	T

The default values are N, MEAN, STD, MIN, and MAX. To compute standard error, confidence limits for the mean, and the Student's t-test, you must use the default value of the VARDEF= option, which is DF. To compute skewness or kurtosis, you must use VARDEF=N or VARDEF=DF. Use CLM or both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM to compute a one-sided confidence limit. The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 497.

CLASS Statement

CLASS *variables* < / *options* > **;**

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. See the section "Levelization of Classification Variables" on page 33 of Chapter 2, "Shared Concepts and Topics" for details about these mappings.

Levels of classification variables are ordered by their external formatted values, except for numeric variables with no explicit format, which are ordered by their unformatted (internal) values.

Required Argument

variables

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *classification variables*. Classification variables are numeric or character. Classification variables can have continuous values, but they typically have a few discrete values that define levels of the variable. You do not have to sort the data by classification variables.

Use the TYPES statement or the WAYS statement to control which classification variables PROC HPSUMMARY uses to group the data. See the section "How PROC HPSUMMARY Groups Data" on page 491.

To reduce the number of classification variable levels, use a FORMAT statement to combine variable values. When a format combines several internal values into one formatted value, PROC HPSUMMARY outputs the lowest internal value.

Optional Arguments

GROUPINTERNAL

specifies that formats are not to be applied to the classification variables when PROC HPSUMMARY groups the values to create combinations of classification variables. This option saves computer resources when the numeric classification variables contain discrete values. See the section "Computational Resources" on page 492.

MISSING

considers missing values as valid values for the classification variable levels. Special missing values that represent numeric values—the letters A through Z and the underscore (_) character—are each considered as a separate value. If you omit the MISSING option, then PROC HPSUMMARY excludes the observations with a missing classification variable value from the analysis.

By default, if an observation contains a missing value for any classification variable, then PROC HPSUMMARY excludes that observation from the analysis. If you specify the MISSING option in the PROC HPSUMMARY statement, then the procedure considers missing values as valid levels for the combination of classification variables.

Specifying the MISSING option in the CLASS statement enables you to control the acceptance of missing values for individual classification variables.

See *SAS Language Reference: Concepts* for a discussion of missing values that have special meaning.

FREQ Statement

FREQ *variable* **;**

The FREQ statement specifies a numeric variable that contains the frequency of each observation.

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

The sum of the frequency variable represents the total number of observations.

OUTPUT Statement

OUTPUT < **OUT=***SAS-data-set* > < *output-statistic-specifications* > < / **AUTONAME** > **;**

The OUTPUT statement writes statistics to a new SAS data set. You can use multiple OUTPUT statements to create several OUT= data sets.

Optional Arguments

OUT=*SAS-data-set*

names the new output data set. If *SAS-data-set* does not exist, then PROC HPSUMMARY creates it. If you omit the OUT= option, then the data set is named DATAn, where n is the smallest integer that makes the name unique.

output-statistic-specifications

specifies the statistics to store in the OUT= data set and names one or more variables that contain the statistics. The form of the *output-statistic-specification* is

statistic-keyword < **(***variable-list***)** >**=**< *names* >

where

statistic-keyword

specifies which statistic to store in the output data set. Table 15.4 lists the *statistic-keywords* that are available in the OUTPUT statement.

Table 15.4 Statistics Keywords in the OUTPUT Statement

Descriptive Statistic Keywords:	
CSS	RANGE
CV	SKEWNESS | SKEW
KURTOSIS | KURT	STDDEV | STD
LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
NMISS	
Quantile Statistic Keywords:	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
P20	P30
P40	P60

Table 15.4 (continued)

P70	P80
Q1 | P25	QRANGE
Hypothesis Testing Keywords:	
PROBT | PRT	T

By default the statistics in the output data set automatically inherit the analysis variable's format, informat, and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format might be invalid for these statistics (for example, dollar or datetime formats). If you omit a *variable-list* and *names*, then PROC HPSUMMARY allows the *statistic-keyword* only once in a single OUTPUT statement, unless you also use the AUTONAME option.

The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 497.

variable-list

specifies the names of one or more numeric analysis variables whose statistics you want to store in the output data set. By default, statistics are stored for all numeric analysis variables.

names

specifies one or more names for the variables in output data set to contain the analysis variable statistics. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on. The default value is the analysis variable name. If you specify the AUTONAME option, then the default is the combination of the analysis variable name and the *statistic-keyword*. If you use the CLASS statement and an OUTPUT statement without an *output-statistic-specification*, then the output data set contains five observations for each combination of classification variables: the value of N, MIN, MAX, MEAN, and STD. If you use the WEIGHT statement or the WEIGHT option in the VAR statement, then the output data set also contains an observation with the sum of weights (SUMWGT) for each combination of classification variables.

If you specify *variable-list*, then PROC HPSUMMARY uses the order in which you specify the analysis variables to store the statistics in the output data set variables. You can use the AUTONAME option to request that PROC HPSUMMARY generate unique names for multiple variables and statistics.

AUTONAME

requests that PROC HPSUMMARY create a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending the *statistic-keyword* to the input variable name. For example, the following statement produces the x_Min variable in the output data set:

```
output min(x)=/autoname;
```

AUTONAME activates the SAS internal mechanism that automatically resolves conflicts in the variable names in the output data set so that duplicate variables do not generate errors. As a result, the following statement produces two variables, x_Min and x_Min2, in the output data set:

```
output min(x)= min(x)=/autoname;
```

PERFORMANCE Statement

PERFORMANCE *performance-options* **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a High-Performance Analytics procedure.

With the PERFORMANCE statement, you can also control whether a High-Performance Analytics procedure executes in SMP or MPP mode.

It is important to remember the distinction between the NODES= and NTHREADS= options. The NODES= option specifies the number of separate grid nodes that participate in the HPSUMMARY execution, while the NTHREADS= option influences how many threads are used by the HPSUMMARY procedure instance that runs on each node. If the data are located on the grid, then all nodes must be engaged; therefore, the NODES= option might be overridden. Setting NODES=0 causes PROC HPSUMMARY to execute on the client side only. Setting the NTHREADS= option to a value that is greater than the CPU count on each grid node is not likely to improve overall throughput.

The PERFORMANCE statement for High-Performance Analytics procedures is documented in section "PERFORMANCE Statement" on page 29 of Chapter 2, "Shared Concepts and Topics."

TYPES Statement

TYPES *requests* **;**

The TYPES statement identifies which of the possible combinations of classification variables to generate. The TYPES statement requires the specification of a CLASS statement.

Required Argument

requests

specifies which of the 2^k combinations of classification variables PROC HPSUMMARY uses to create the types, where k is the number of classification variables. A *request* includes one classification variable name, several classification variable names separated by asterisks, or ().

To request classification variable combinations quickly, use a grouping syntax by placing parentheses around several variables and joining other variables or variable combinations. The examples in Table 15.5 illustrate grouping syntax:

Table 15.5 Examples of Grouping Syntax

Request	Equivalent To
`types A*(B C);`	`types A*B A*C;`
`types (A B)*(C D);`	`types A*C A*D B*C B*D;`
`types (A B C)*D;`	`types A*D B*D C*D;`

You can use parentheses () to request the overall total (_TYPE_=0). If you do not need all types in the output data set, then use the TYPES statement to specify particular subtypes rather than applying a WHERE clause to the data set. Doing so saves time and computer memory.

Order of Analyses in the Output

The SUMMARY procedure writes analyses to the output in order of increasing values of the _TYPE_ variable. When PROC HPSUMMARY executes on the grid, the order of observations within the output is not deterministic because the output is returned in parallel. You can sort the output as follows:

- If output is directed back to the client, then to achieve an output order that is similar to the output of PROC SUMMARY, you need to subsequently sort the data by _TYPE_ and the classification variables.
- If output is directed back to the grid (so that the results are distributed), then there is no order within the output. To retrieve the observations in order, you can execute an SQL query, specifying that the selecting rows be returned in order by _TYPE_ and the classification variables.

The _TYPE_ variable is calculated even if no output data set is requested. For more information about the _TYPE_ variable, see the section “Output Data Set” on page 496.

VAR Statement

VAR *variables* < / **WEIGHT=***weight-variable* > **;**

The VAR statement identifies the analysis variables and their order in the output. If you omit the VAR statement, then PROC HPSUMMARY analyzes all numeric variables that are not listed in the other statements. When all variables are character variables, PROC SUMMARY produces a simple count of observations. You can use multiple VAR statements.

Required Argument

variables

identifies one or more analysis variables and specifies their order in the results.

Optional Argument

WEIGHT=*weight-variable*

specifies a numeric variable whose values weight the values of the *variables*. The *weight-variable* does not have to be an integer. Table 15.6 describes how PROC HPSUMMARY treats various values of the *weight-variable*.

Table 15.6 Responses to Values of *weight-variable*

Value	PROC HPSUMMARY Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC HPSUMMARY statement.

The *weight-variable* does not change how the procedure determines the range, extreme values, or number of missing values.

To compute weighted quantiles, use QMETHOD=OS in the PROC HPSUMMARY statement. Skewness and kurtosis are not available with the WEIGHT= option.

When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate. Use the WEIGHT= option in multiple VAR statements to specify different weights for the analysis variables.

WAYS Statement

WAYS *list* **;**

The WAYS statement specifies the number of ways to make unique combinations of classification variables. You can use the TYPES statement to specify additional combinations of classification variables.

Required Argument

list

specifies one or more integers that define the number of classification variables to combine to form all the unique combinations of classification variables. For example, you can specify 2 for all possible pairs and 3 for all possible triples. The *list* can be specified in the following ways:

- *m*
- *m1 m2 ... mn*
- *m1*,*m2*,...,*mn*
- *m* TO *n* <BY *increment*>
- *m1*,*m2*, TO *m3* <BY *increment*>,*m4*

The range of *list* is from 0 to the maximum number of classification variables.

The following statements are an example of creating two-way types for the classification variables A, B, and C:

```
class A B C ;
ways 2;
```

The WAYS statement in this example is equivalent to specifying **A*B**, **A*C**, and **B*C** in the TYPES statement.

WEIGHT Statement

WEIGHT *weight-variable* **;**

The WEIGHT statement specifies weights for observations in the statistical calculations.

Required Argument

weight-variable

specifies a numeric *weight-variable* whose values weight the values of the analysis variables. The values of *weight-variable* do not have to be integers. Table 15.7 describes how PROC HPSUMMARY treats various values of *weight-variable*.

Table 15.7 Responses to Values of *weight-variable*

Value	PROC HPSUMMARY Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC HPSUMMARY statement.

CAUTION: Single extreme weight values can cause inaccurate results. When one (and only one) weight value is many orders of magnitude larger than the other weight values (for example, 49 weight values of 1 and one weight value of 1×10^{14}), certain statistics might not be within acceptable accuracy limits. The affected statistics are based on the second moment (such as standard deviation, corrected sum of squares, variance, and standard error of the mean). Under certain circumstances, no warning is written to the SAS log.

To compute weighted quantiles, you must use QMETHOD=OS in the PROC statement. Skewness and kurtosis are not available with the WEIGHT statement.

PROC HPSUMMARY does not compute MODE when a weight variable is active. Instead, you can try using the UNIVARIATE procedure when MODE needs to be computed and a weight variable is active.

If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC HPSUMMARY uses this variable instead to weight those VAR statement variables.

When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the section "Keywords and Formulas" on page 497 for more information.

Details: HPSUMMARY Procedure

How PROC HPSUMMARY Groups Data

Groups of observations are defined by specifying certain variables as classification variables in the CLASS statement. Unique values of the n CLASS variables are used to partition the input data, and the resulting summarized data (one observation per group) is called the "n-way."

PROC HPSUMMARY can also combine the partitioned groups into larger groups by removing one or more CLASS variables from consideration when grouping. There are 2^n different groupings that can be generated from n CLASS variables. Each of these groupings is a "type," which appears in the output data set as a variable named _TYPE_. Type 0 includes no CLASS variables and summarizes the entire input data set, Type 1 includes only the last CLASS variable specified, and so on to Type $2^n - 1$, which is the n-way.

By default, PROC HPSUMMARY generates only the n-way. The option ALLTYPES (or ALLWAYS) in the PROC HPSUMMARY statement generates all 2^n types. You can also use either of the following statements to choose which types appear in the output data set:

- The WAYS statement specifies how many CLASS variables appear in each output type. For example, WAYS 1 produces types for each CLASS variable individually, WAYS 2 generates all $\binom{n}{2}$ possible pairs, and so on.
- The TYPES statement explicitly specifies the desired types by CLASS variable name, such as TYPES A A*B C (), where A*B might specify Type 6 and "()" specifies Type 0.

The TYPES statement controls which of the available classification variables PROC HPSUMMARY uses to subgroup the data. The unique combinations of these active classification variable values that occur together in any single observation of the input data set determine the data subgroups. Each subgroup that PROC HPSUMMARY generates for a given type is called a level of that type. For all types, the inactive classification variables can still affect the total observation count of the rejection of observations with missing values. When you use a WAYS statement, PROC HPSUMMARY generates types that correspond to every possible unique combination of n classification variables chosen from the complete set of classification variables. For example

```
proc hpsummary;
```

```
      class a b c d e;
      ways 2 3;
      output out=results;
   run;
```

is equivalent to

```
   proc hpsummary;
      class a b c d e;
      types a*b a*c a*d a*e b*c b*d b*e c*d c*e d*e
            a*b*c a*b*d a*b*e a*c*d a*c*e a*d*e
            b*c*d b*c*e c*d*e;
      output out=results;
   run;
```

If you omit the TYPES statement and the WAYS statement, then PROC HPSUMMARY uses all classification variables to subgroup the data (the NWAY type) for the output data set.

Computational Resources

The total of unique classification values that PROC HPSUMMARY allows depends on the amount of computer memory that is available. PROC HPSUMMARY uses the same memory allocation scheme across all operating environments. When classification variables are involved, PROC HPSUMMARY must keep a copy of each unique value of each classification variable in memory. You can estimate the memory requirements to group the classification variable by calculating

$$Nc_1(Lc_1 + K) + Nc_2(Lc_2 + K) + \ldots + Nc_n(Lc_n + K)$$

where Nc_i is the number of unique values for the classification variable, Lc_i is the combined unformatted and formatted length of c_i, and K is some constant on the order of 32 bytes (64 for 64-bit architectures). When you use the GROUPINTERNAL option in the CLASS statement, Lc_i is simply the unformatted length of c_i.

The GROUPINTERNAL option can improve computer performance because the grouping process is based on the internal values of the classification variables. If a numeric classification variable is not assigned a format and you do not specify GROUPINTERNAL, then PROC HPSUMMARY uses the default format, BEST12., to format numeric values as character strings. Then PROC HPSUMMARY groups these numeric variables by their character values, which takes additional time and computer memory.

Each unique combination of classification variables c_{1_i} c_{2_j} for a given type forms a level in that type. See the section “TYPES Statement” on page 487. You can estimate the maximum potential space requirements for all levels of a given type, when all combinations actually exist in the data (a complete type), by calculating

$$W * Nc_1 * Nc_2 * \ldots * Nc_n$$

where W is a constant based on the number of variables analyzed and the number of statistics calculated (unless you request QMETHOD=OS to compute the quantiles) and $Nc_1 \ldots Nc_n$ are the number of unique levels for the active classification variables of the given type.

Clearly, the memory requirements of the levels overwhelm the levels of the classification variables. For information about how to adjust your computation resource parameters, see the SAS documentation for your operating environment.

Another way to enhance performance is by carefully applying the TYPES or WAYS statement, limiting the computations to only those combinations of classification variables that you are interested in.

Statistical Computations

Computation of Moment Statistics

PROC HPSUMMARY uses single-pass algorithms to compute the moment statistics (such as mean, variance, skewness, and kurtosis). See the section "Keywords and Formulas" on page 497 for the statistical formulas.

The computational details for confidence limits, hypothesis test statistics, and quantile statistics follow.

Confidence Limits

With the *statistic-keywords* CLM, LCLM, and UCLM, you can compute confidence limits for the mean. A confidence limit is a range (constructed around the value of a sample statistic) that contains the corresponding true population value with given probability (ALPHA=) in repeated sampling. A two-sided $100(1-\alpha)\%$ confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{n}}$$

where $s = \sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$ and $t_{(1-\alpha/2;n-1)}$ is the $(1-\alpha/2)$ critical value of the Student's t statistic with $n-1$ degrees of freedom.

A one-sided $100(1-\alpha)\%$ confidence interval is computed as

$$\bar{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(upper)}$$

$$\bar{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(lower)}$$

A two-sided $100(1-\alpha)\%$ confidence interval for the standard deviation has lower and upper limits

$$s\sqrt{\frac{n-1}{\chi^2_{(1-\alpha/2;n-1)}}}, \quad s\sqrt{\frac{n-1}{\chi^2_{(\alpha/2;n-1)}}}$$

where $\chi^2_{(1-\alpha/2;n-1)}$ and $\chi^2_{(\alpha/2;n-1)}$ are the $(1-\alpha/2)$ and $\alpha/2$ critical values of the chi-square statistic with $n-1$ degrees of freedom. A one-sided $100(1-\alpha)\%$ confidence interval is computed by replacing $\alpha/2$ with α.

A $100(1-\alpha)\%$ confidence interval for the variance has upper and lower limits that are equal to the squares of the corresponding upper and lower limits for the standard deviation.

If you use the WEIGHT statement or the WEIGHT= option in a VAR statement and the default value of the VARDEF= option (which is DF), the $100(1-\alpha)\%$ confidence interval for the weighted mean has upper and lower limits

$$\bar{y}_w \pm t_{(1-\alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, w_i is the weight for the ith observation, and $t_{(1-\alpha/2)}$ is the $(1-\alpha/2)$ critical value for the Student's t distribution with $n-1$ degrees of freedom.

Student's t Test

PROC HPSUMMARY calculates the t statistic as

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where $\bar{x}$ is the sample mean, n is the number of nonmissing values for a variable, and s is the sample standard deviation. Under the null hypothesis, the population mean equals μ_0. When the data values are approximately normally distributed, the probability under the null hypothesis of a t statistic as extreme as, or more extreme than, the observed value (the p-value) is obtained from the t distribution with $n-1$ degrees of freedom. For large n, the t statistic is asymptotically equivalent to a z test.

When you use the WEIGHT statement or the WEIGHT= option in a VAR statement and the default value of the VARDEF= option (which is DF), the Student's t statistic is calculated as

$$t_w = \frac{\bar{y}_w - \mu_0}{s_w \Big/ \sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for the ith observation. The t_w statistic is treated as having a Student's t distribution with $n-1$ degrees of freedom. If you specify the EXCLNPWGT option in the PROC HPSUMMARY statement, then n is the number of nonmissing observations when the value of the WEIGHT variable is positive. By default, n is the number of nonmissing observations for the WEIGHT variable.

Quantiles

The options QMETHOD=, QNTLDEF=, and QMARKERS= determine how PROC HPSUMMARY calculates quantiles. The QNTLDEF= option deals with the mathematical definition of a quantile. See the section "Quantile and Related Statistics" on page 501. The QMETHOD= option specifies how PROC HPSUMMARY handles the input data: When QMETHOD=OS, PROC HPSUMMARY reads all data into memory and sorts it by unique value. When QMETHOD=P2, PROC HPSUMMARY accumulates all data into a fixed sample size that is used to approximate the quantile.

If data set A has 100 unique values for a numeric variable X and data set B has 1,000 unique values for numeric variable X, then QMETHOD=OS for data set B requires 10 times as much memory as it does for data set A. If QMETHOD=P2, then both data sets A and B require the same memory space to generate quantiles.

The QMETHOD=P2 technique is based on the piecewise-parabolic (P^2) algorithm invented by Jain and Chlamtac (1985). P^2 is a one-pass algorithm to determine quantiles for a large data set. It requires a fixed amount of memory for each variable for each level within the type. However, using simulation studies, reliable estimations of some quantiles (P1, P5, P95, P99) cannot be possible for some data sets such as data sets with heavily tailed or skewed distributions.

If the number of observations is less than the QMARKERS= value, then QMETHOD=P2 produces the same results as QMETHOD=OS when QNTLDEF=5. To compute weighted quantiles, you must use QMETHOD=OS.

Results

Missing Values

PROC HPSUMMARY excludes missing values for the analysis variables before calculating statistics. Each analysis variable is treated individually; a missing value for an observation in one variable does not affect the calculations for other variables. PROC HPSUMMARY handles missing values as follows:

- If a classification variable has a missing value for an observation, then PROC HPSUMMARY excludes that observation from the analysis unless you use the MISSING option in the PROC statement or CLASS statement.
- If a FREQ variable value is missing or nonpositive, then PROC HPSUMMARY excludes the observation from the analysis.
- If a WEIGHT variable value is missing, then PROC HPSUMMARY excludes the observation from the analysis.

PROC HPSUMMARY tabulates the number of the missing values. Before the number of missing values are tabulated, PROC HPSUMMARY excludes observations with frequencies that are nonpositive when you use the FREQ statement and observations with weights that are missing or nonpositive (when you use the EXCLNPWGT option) when you use the WEIGHT statement. To report this information in the procedure output use the NMISS *statistic-keyword* in the PROC HPSUMMARY statement.

The N Obs Statistic

By default when you use a CLASS statement, PROC HPSUMMARY displays an additional statistic called N Obs. This statistic reports the total number of observations or the sum of the observations of the FREQ variable that PROC HPSUMMARY processes for each class level. PROC HPSUMMARY might omit observations from this total because of missing values in one or more classification variables. Because of this action and the exclusion of observations when the *weight-variable* (specified in the WEIGHT statement or in the WEIGHT= option in the VAR statement) contains missing values, there is not always a direct relationship between N Obs, N, and NMISS.

In the output data set, the value of N Obs is stored in the _FREQ_ variable.

Output Data Set

PROC HPSUMMARY creates one output data set. The procedure does not print the output data set. Use the PRINT procedure, the REPORT procedure, or another SAS reporting tool to display the output data set.

NOTE: By default, the statistics in the output data set automatically inherit the analysis variable's format and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format can be invalid for these statistics.

The output data set can contain these variables:

- the variables specified in the CLASS statement.
- the variable _TYPE_ that contains information about the classification variables. By default _TYPE_ is a numeric variable. If you specify CHARTYPE in the PROC statement, then _TYPE_ is a character variable. When you use more than 32 classification variables, _TYPE_ is automatically a character variable.
- the variable _FREQ_ that contains the number of observations that a given output level represents.
- the variables requested in the OUTPUT statement that contain the output statistics and extreme values.
- the variable _STAT_ that contains the names of the default statistics if you omit statistic keywords.

The value of _TYPE_ indicates which combination of the classification variables PROC HPSUMMARY uses to compute the statistics. The character value of _TYPE_ is a series of zeros and ones, where each value of one indicates an active classification variable in the type. For example, with three classification variables, PROC HPSUMMARY represents type 1 as 001, type 5 as 101, and so on.

Usually, the output data set contains one observation per level per type. However, if you omit *statistic-keywords* in the OUTPUT statement, then the output data set contains five observations per level (six if you specify a WEIGHT variable). Therefore, the total number of observations in the output data set is equal to the sum of the levels for all the types that you request multiplied by 1, 5, or 6, whichever is applicable.

If you omit the CLASS statement (_TYPE_= 0), then there is always exactly one level of output per output data set. If you use a CLASS statement, then the number of levels for each type that you request has an upper bound equal to the number of observations in the input data set. By default, PROC HPSUMMARY generates all possible types. In this case the total number of levels for each output data set has an upper bound equal to

$$m \cdot (2^k - 1) \cdot n + 1$$

where k is the number of classification variables, n is the number of observations in the input data set, and m is 1, 5, or 6.

PROC HPSUMMARY determines the actual number of levels for a given type from the number of unique combinations of each active classification variable. A single level consists of all input observations whose formatted class values match.

Table 15.8 shows the values of _TYPE_ and the number of observations in the data set when you specify one, two, and three classification variables.

Table 15.8 The Effect of Classification Variables on the OUTPUT Data Set

CLASS Variables							
C	B	A	_WAY_	_TYPE_	Subgroup defined by	Number of observations of this _TYPE_ and _WAY_ in the data set	Total number of observations in the data set
0	0	0	0	0	Total	1	
0	0	1	1	1	A	a	1+a
0	1	0	1	2	B	b	
0	1	1	2	3	A*B	a*b	1+a+b+a*b
1	0	0	1	4	C	c	
1	0	1	2	5	A*C	a*c	
1	1	0	2	6	B*C	b*c	1+a+b+a*b+c
1	1	1	3	7	A*B*C	a*b*c	+a*c+b*c+a*b*c
Character binary equivalent of _TYPE_ (CHARTYPE option in the PROC HPSUMMARY statement)						a,b,c = number of levels of A, B, C, respectively	

Keywords and Formulas

Simple Statistics

The HPSUMMARY procedure uses a standardized set of keywords to refer to statistics. You specify these keywords in SAS statements to request the statistics to be displayed or stored in an output data set.

In the following notation, summation is over observations that contain nonmissing values of the analyzed variable and, except where shown, over nonmissing weights and frequencies of one or more:

x_i
: is the nonmissing value of the analyzed variable for observation i.

f_i
: is the frequency that is associated with x_i if you use a FREQ statement. If you omit the FREQ statement, then $f_i = 1$ for all i.

w_i

is the weight that is associated with x_i if you use a WEIGHT statement. The HPSUMMARY procedure automatically excludes the values of x_i with missing weights from the analysis.

By default, the HPSUMMARY procedure treats a negative weight as if it is equal to 0. However, if you use the EXCLNPWGT option in the PROC HPSUMMARY statement, then the procedure also excludes those values of with nonpositive weights.

If you omit the WEIGHT statement, then $w_i = 1$ for all i.

n

is the number of nonmissing values of x_i, $\sum f_i$. If you use the EXCLNPWGT option and the WEIGHT statement, then n is the number of nonmissing values with positive weights.

$\bar{x}$

is the mean

$$\sum w_i x_i \Big/ \sum w_i$$

s^2

is the variance

$$\frac{1}{d} \sum w_i (x_i - \bar{x})^2$$

where d is the variance divisor (the VARDEF= option) that you specify in the PROC HPSUMMARY statement. Valid values are as follows:

When VARDEF=	d **equals**
N	n
DF	$n - 1$
WEIGHT | WGT	$\sum_i w_i$
WDF	$\left(\sum_i w_i\right) - 1$

The default is DF.

z_i

is the standardized variable

$$(x_i - \bar{x})/s$$

PROC HPSUMMARY calculates the following simple statistics:

- number of missing values
- number of nonmissing values
- number of observations
- sum of weights
- mean
- sum

- extreme values
- minimum
- maximum
- range
- uncorrected sum of squares
- corrected sum of squares
- variance
- standard deviation
- standard error of the mean
- coefficient of variation
- skewness
- kurtosis
- confidence limits of the mean
- median
- mode
- percentiles/deciles/quartiles
- t test for mean=0

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

Descriptive Statistics

The keywords for descriptive statistics are as follows:

CSS
: is the sum of squares corrected for the mean, computed as

$$\sum w_i(x_i - \bar{x})^2$$

CV
: is the percent coefficient of variation, computed as

$$(100s)/\bar{x}$$

KURTOSIS | KURT

is the kurtosis, which measures heaviness of tails. When VARDEF=DF, the kurtosis is computed as

$$c_{4_n} \sum z_i^4 - \frac{3\,(n-1)^2}{(n-2)\,(n-3)}$$

where c_{4_n} is $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$

When VARDEF=N, the kurtosis is computed as

$$\frac{1}{n} \sum z_i^4 - 3$$

The formula is invariant under the transformation $w_i^* = zw_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the kurtosis is set to missing.

MAX

is the maximum value of x_i.

MEAN

is the arithmetic mean $\bar{x}$.

MIN

is the minimum value of x_i.

MODE

is the most frequent value of x_i.

NOTE: When QMETHOD=P2, PROC HPSUMMARY does not compute MODE.

N

is the number of x_i values that are not missing. Observations with $f_i < 1$ and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of N.

NMISS

is the number of x_i values that are missing. Observations with $f_i < 1$ and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of NMISS.

NOBS

is the total number of observations and is calculated as the sum of N and NMISS. However, if you use the WEIGHT statement, then NOBS is calculated as the sum of N, NMISS, and the number of observations excluded because of missing or nonpositive weights.

RANGE

is the range and is calculated as the difference between maximum value and minimum value.

SKEWNESS | SKEW

is skewness, which measures the tendency of the deviations to be larger in one direction than in the other. When VARDEF=DF, the skewness is computed as

$$c_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$.

When VARDEF=N, the skewness is computed as

$$\frac{1}{n}\sum z_i^3$$

The formula is invariant under the transformation $w_i^* = zw_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the skewness is set to missing.

STDDEV | STD
: is the standard deviation s and is computed as the square root of the variance, s^2.

STDERR | STDMEAN
: is the standard error of the mean, computed as

$$\frac{s}{\sqrt{\sum w_i}}$$

when VARDEF=DF, which is the default. Otherwise, STDERR is set to missing.

SUM
: is the sum, computed as

$$\sum w_i x_i$$

SUMWGT
: is the sum of the weights, W, computed as

$$\sum w_i$$

USS
: is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VAR
: is the variance s^2.

Quantile and Related Statistics

The keywords for quantiles and related statistics are as follows:

MEDIAN
: is the middle value.

P*n*
: is the nth percentile. For example, P1 is the first percentile, P5 is the fifth percentile, P50 is the 50th percentile, and P99 is the 99th percentile.

Q1
: is the lower quartile (25th percentile).

Q3
: is the upper quartile (75th percentile).

QRANGE
: is interquartile range and is calculated as

$$Q_3 - Q_1$$

You use the QNTLDEF= option to specify the method that the HPSUMMARY procedure uses to compute percentiles. Let n be the number of nonmissing values for a variable, and let $x_1, x_2, \ldots, x_n$ represent the ordered values of the variable such that x_1 is the smallest value, x_2 is the next smallest value, and x_n is the largest value. For the tth percentile between 0 and 1, let $p = t/100$. Then define j as the integer part of np and g as the fractional part of np or $(n+1)p$, so that

$$\begin{aligned} np &= j + g \quad \text{when QNTLDEF=1, 2, 3, or 5} \\ (n+1)p &= j + g \quad \text{when QNTLDEF=4} \end{aligned}$$

Here, QNTLDEF= specifies the method that the procedure uses to compute the tth percentile, as shown in Table 15.10.

When you use the WEIGHT statement, the tth percentile is computed as

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_j is the weight associated with x_i and $W = \sum_{i=1}^{n} w_i$ is the sum of the weights. When the observations have identical weights, the weighted percentiles are the same as the unweighted percentiles with QNTLDEF=5.

Table 15.10 Methods for Computing Quantile Statistics

QNTLDEF=	Description	Formula	
1	Weighted average at x_{np}	$y = (1-g)x_j + gx_{j+1}$ where x_0 is taken to be x_1	
2	Observation numbered closest to np	$y = x_i$	if $g \neq \frac{1}{2}$
		$y = x_j$	if $g = \frac{1}{2}$ and j is even
		$y = x_{j+1}$	if $g = \frac{1}{2}$ and j is odd
		where i is the integer part of $np + \frac{1}{2}$	
3	Empirical distribution function	$y = x_j$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$
4	Weighted average aimed at $x_{(n+1)p}$	$y = (1-g)x_j + gx_{j+1}$ where x_{n+1} is taken to be x_n	
5	Empirical distribution function with averaging	$y = \frac{1}{2}(x_j + x_{j+1})$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$

Hypothesis Testing Statistics

The keywords for hypothesis testing statistics are as follows:

T
: is the Student's t statistic to test the null hypothesis that the population mean is equal to μ_0 and is calculated as

$$\frac{\bar{x} - \mu_0}{s \Big/ \sqrt{\sum w_i}}$$

By default, μ_0 is equal to zero. You must use VARDEF=DF, which is the default variance divisor; otherwise T is set to missing.

By default, when you use a WEIGHT statement, the procedure counts the x_i values with nonpositive weights in the degrees of freedom. Use the EXCLNPWGT option in the PROC HPSUMMARY statement to exclude values with nonpositive weights.

PROBT | PRT
: is the two-tailed p-value for the Student's t statistic, T, with $n - 1$ degrees of freedom. This value is the probability under the null hypothesis of obtaining a more extreme value of T than is observed in this sample.

Confidence Limits for the Mean

The keywords for confidence limits are as follows:

CLM
: is the two-sided confidence limit for the mean. A two-sided $100(1 - \alpha)$ percent confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2;\, n-1)} \frac{s}{\sqrt{\sum w_i}}$$

where s is $\sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$, $t_{(1-\alpha/2;\, n-1)}$ is the $(1 - \alpha/2)$ critical value of the Student's t statistic with $n - 1$ degrees of freedom, and α is the value of the ALPHA= option which by default is 0.05. Unless you use VARDEF=DF (which is the default variance divisor), CLM is set to missing.

LCLM
: is the one-sided confidence limit below the mean. The one-sided $100(1 - \alpha)$ percent confidence interval for the mean has the lower limit

$$\bar{x} - t_{(1-\alpha;\, n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF (which is the default variance divisor), LCLM is set to missing.

UCLM
: is the one-sided confidence limit above the mean. The one-sided $100(1-\alpha)$ percent confidence interval for the mean has the upper limit

$$\bar{x} + t_{(1-\alpha;\, n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF (which is the default variance divisor), UCLM is set to missing.

Data Requirements for the HPSUMMARY Procedure

The following are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if VARDEF=DF (the default) and the following requirements are not met:

- N and NMISS are computed regardless of the number of missing or nonmissing observations.
- SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.
- VAR, STD, STDERR, CV, T, PRT, and PROBT require at least two nonmissing observations.
- SKEWNESS requires at least three nonmissing observations.
- KURTOSIS requires at least four nonmissing observations.
- SKEWNESS, KURTOSIS, T, PROBT, and PRT require that STD is greater than zero.
- CV requires that MEAN is not equal to zero.
- CLM, LCLM, UCLM, STDERR, T, PRT, and PROBT require that VARDEF=DF.

References

Jain, R. and Chlamtac, I. (1985), “The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations,” *Communications of the ACM*, 28, 1076–1085.

Chapter 16

Installation and System Administration

Contents

Overview of SAS High-Performance Analytics Installation

The SAS High-Performance Analytics software has client-side and server-side components. The High-Performance Analytics client is the machine on which the SAS session executes and is where you submit procedure code. The server is the node on the High-Performance Analytics appliance to which the client connects. The server is frequently referred to as the *grid host head node*, or the *host node* throughout this document.

Installation of SAS High-Performance Analytics involves steps on the client and the appliance. These are discussed separately in this chapter. The steps on the appliance include setup at the operating system level and the database level.

Configure the Client

After installing SAS on the client, you need to perform the following steps to configure SAS High-Performance Analytics:

- Configure the SAS/ACCESS Interface to Greenplum or the SAS/ACCESS Interface to Teradata.
- Obtain a secure shell (SSH) key to support a secure connection to the appliance.

Configure the SAS/ACCESS Interface to Greenplum

Configure Windows

SAS/ACCESS Interface to Greenplum uses the DataDirect Technologies Greenplum Wire Protocol Open Database Connectivity (ODBC) driver component, which should be downloaded from SAS. See the section “SAS/ACCESS Interface to Greenplum” in the *System Requirements for SAS 9.3 Foundation for Microsoft Windows* documentation for information about the download.

After you have received the ODBC driver, use the following steps to unzip it and store it in the appropriate location.

1 Use Windows Explorer to navigate to the *!SASROOT\access\sasmisc* directory where the *<platform>gplm60.zip* is located (for *<platform>*, type the platform that is specific to your operating system).

2 Unzip the *<platform>gplm60.zip* file.

3 Follow the instructions in the *Readme.txt* file that is contained in the zip file.

During the initial installation of SAS/ACCESS Interface to Greenplum, the SAS Deployment Wizard enables you to specify the location for the ODBC driver component that is required. If you want to update that location, use the SAS Deployment Manager. The procedure for updating the location is described in the section “Configure SAS/ACCESS Interface to Greenplum” of the *SAS Deployment Wizard and SAS Deployment Manager 9.3: User’s Guide*, located at `http://support.sas.com/deploywizug93.html`.

Configure Supported UNIX and Linux Platforms

During the initial installation of SAS/ACCESS Interface to Greenplum, the SAS Deployment Wizard enables you to specify the location for the required ODBC driver. If you want to update that location after the initial installation, you can do so by using the SAS Deployment Manager. The procedure for updating the location is described in the section "Configure SAS/ACCESS Interface to Greenplum" of the *SAS Deployment Wizard and SAS Deployment Manager 9.3: User's Guide*, located at http://support.sas.com/deploywizug93.html.

The location for the ODBC driver is the ODBCHOME directory; it is used to set up the paths both to the shared libraries and also to the *odbc.ini* file. You must set the ODBCHOME environment variable to your ODBC home directory before setting the ODBCINI and shared library environment variables as shown in the following examples.

The *odbc.ini* system information file contains a list of possible data sources that you can connect to your Greenplum servers. You must configure at least one data source in order to use the SAS/ACCESS Interface to Greenplum. Edit the *odbc.ini* file with a text editor to configure the data sources. The general format of the *odbc.ini* file is as follows:

```
[ODBC Data Sources]
greenplum=SAS ACCESS to Greenplum

[ODBC]
InstallDir=<my install dir>
Trace=0
TraceDll=<my install dir>/lib/odbctrac.so
TraceFile=odbctrace.out

[greenplum]
Driver=<my install dir>/lib/S0gplm<file version>.so
Description=SAS ACCESS to Greenplum
AlternateServers=
ApplicationUsingThreads=1
ConnectionReset=0
ConnectionRetryCount=0
ConnectionRetryDelay=3
Database=<db>
EnableDescribeParam=1
ExtendedColumnMetadata=0
FailoverGranularity=0
FailoverMode=0
FailoverPreconnect=0
FetchRefCursor=1
FetchTSWTZasTimestamp=0
FetchTWFSasTime=0
HostName=<Greenplum host>
InitializationString=
LoadBalanceTimeout=0
LoadBalancing=0
LoginTimeout=15
LogonID=
MaxPoolSize=100
```

```
MinPoolSize=0
Password=
Pooling=0
PortNumber=<Greenplum server port>
QueryTimeout=0
ReportCodepageConversionErrors=0
TransactionErrorBehavior=1
XMLDescribeType=-10
```

The `<driver version>` and `<file version>` specifications describe specific versions of the DataDirect Greenplum driver that is installed with SAS/ACCESS Interface to Greenplum. The `<driver version>` in your *odbc.ini* already contains the latest version of the DataDirect driver that SAS ships. Also, the `<file version>` contains a two-digit version that describes the version of the actual driver library. You do not need to update these two version specifications in your *odbc.ini* file.

You should replace all occurrences of `<my install dir>` in the sample *odbc.ini* with the names of path and directory where you installed the Greenplum ODBC driver. This is the same directory that is specified by the ODBCHOME environment variable that you set earlier in this section. You should also replace `<Greenplum host>` with the IP address or named server of your Greenplum server, `<Greenplum server port>` with the port number that your Greenplum server is listening on (typically 5432), and `<db>` with the name of your Greenplum database.

In the preceding example, `greenplum` is the name of the configured data source name that is used in the DSN= option when you assign a libname to the SAS/ACCESS Interface to Greenplum engine.

After you configure your data sources, you must set the ODBCINI environment variable to the location and name of your *odbc.ini*:

```
ODBCINI=$ODBCHOME/odbc.ini
export ODBCINI
```

The DataDirect Greenplum ODBC drivers are ODBC API-compliant shared libraries, referred to in UNIX as shared objects. You must include the full path to the shared libraries in the shared library path as shown in the following example so that the ODBC drivers can be loaded dynamically at run time:

```
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ODBCHOME/lib
$ export LD_LIBRARY_PATH
```

It might be convenient to set these environment variables in the *bin/sasenv_local* file of your SAS installation to enable ODBC for all SAS users.

Register SAS/ACCESS Interface to Greenplum

If the GREENPLM name is not known in LIBNAME statements, run the following SAS procedure to register the SAS/ACCESS Interface to Greenplum product with the SAS system catalog:

```
proc nickname cat=sashelp.core engine;
   add nickname=greenplm
       module  =sasiogpl
       desc    ="SAS/ACCESS to Greenplum"
       prefered eng;
quit;
```

Configure SAS/ACCESS Interface to Teradata

Before you use SAS/ACCESS Interface to Teradata software, verify connectivity by logging into your Teradata account with the Teradata BTEQ utility. If BTEQ is unavailable, establish connectivity as documented in the *SAS/ACCESS to Teradata* white paper, available at `http://support.sas.com/resources/papers/teradata.pdf`.

If BTEQ fails to connect to the Teradata server, you might need an entry in the *hosts* file on your PC to provide the network address of the Teradata server. Typically, this means adding a **`dbccop1`** entry to your *hosts* file. For more details about the entry, see the *Teradata Client for Windows Installation Guide*.

FastExport

For optimal reads of large tables, SAS/ACCESS can use the Teradata FastExport Utility, which must be present on the system where you install SAS. The FastExport Utility is not required; SAS/ACCESS reads large tables quite efficiently without it. For further information, see the DBSLICEPARM option in the chapter "SAS/ACCESS Interface to Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*. See also the *SAS/ACCESS to Teradata* white paper, available on the SAS Web site at `http://support.sas.com/resources/papers/teradata.pdf`.

Contact Teradata if you want to obtain the Teradata FastExport Utility.

Configure FastExport for Windows

You must also modify the PATH system variable. Append the following directory paths to the end of PATH:

- The directory where *fexp.exe* (the FastExport Utility) is located. Often this is *C:\Program Files\Teradata\Client\13.10\bin*.
- The directory where *sasaxsm.dll* is located (*sasaxsm.dll* is usually located in your SAS product tree, in the *!sasroot\access\sasexe directory*).

Configure FastExport for Supported UNIX and Linux Platforms

As needed, modify your library path environment variable to include the directory that contains *sasaxsm.so*. These shared objects are delivered in the *$SASROOT/sasexe directory*. You can copy these modules where you want, but ensure that the directory you copy them into is in the LD_LIBRARY_PATH environment variable.

Make sure that the Teradata FastExport utility, *fexp*, has its directory included in the PATH environment variable. This utility is usually installed in the */usr/bin* directory.

MultiLoad

SAS/ACCESS can load large volumes of data to non-empty tables using MultiLoad. To perform this loading, the Teradata MultiLoad Utility must be present on the system where you install SAS.

The MultiLoad Utility is not required; SAS/ACCESS provides other options for loading tables. For further information, see the MULTISTMT option in the chapter "SAS/ACCESS Interface to Teradata" in *SAS ACCESS 9.3 for Relational Databases: Reference.*

Contact Teradata if you want to obtain the Teradata MultiLoad Utility.

Configure MultiLoad for Windows

You must also modify the PATH system variable. Append the following directory paths to the end of PATH:

- The directory where *mload.exe* (the MultiLoad Utility) is located. Often this is *C:\Program Files\Teradata\Client\13.10\bin*
- The directory where *sasmlam.dll* and *sasmlne.dll* are located (*sasmlam.dll* and *sasmlne.dll* are usually located in your SAS product tree, in the *!sasroot\access\sasexe* directory).

Configure MultiLoad for Supported UNIX and Linux Platforms

As needed, modify the LD_LIBRARY_PATH environment variable to include the directory that contains the shared objects *sasmlam.so* and *sasmlne.so*. These shared objects are delivered in the *$SASROOT/sasexe directory*. You can copy these modules where you want, but ensure that the directory you copy them into is in the LD_LIBRARY_PATH environment variable. Make sure that the Teradata MultiLoad utility, *mload*, has its directory included in the PATH environment variable. This utility is usually installed in the */usr/bin* directory.

Teradata Parallel Transporter

SAS/ACCESS software supports the Teradata Parallel Transporter API for loading data using Multiload, Fastload, and Multi-Statement inserts. The API also supports reading data using FastExport. The Teradata Parallel Transporter API is not required; SAS/ACCESS provides other options for loading and reading data.

If you plan to use the Teradata Parallel Transporter API, the API must be installed on the system where SAS is installed. Additional requirements depend on the operating system as follows.

Configure Teradata Parallel Transporter for Windows

The system variable TKPATHIA32 must be modified. If the variable does not exist already, it must be added. To modify the variable, append the directory where *sasiotpt.dll* is located (*sasiotpt.dll* is usually located in your SAS product tree, in the *!sasroot\access\sasexe directory*).

Your PATH environment variable must also be updated to include the following location: *C:\Program Files\Teradata\Client\13.10\Teradata ParallelTransporter\bin\vc7*. For more information, go to `http://support.sas.com/kb/42/351.html`.

Configure Teradata Parallel Transporter for Supported UNIX and Linux Platforms

The PATH system variable must include the location of the Teradata Parallel Transporter API libraries (specifically the location of *libtelapi.**). The LD_LIBRARY_PATH and NLSPATH environment variable also need to include that path:

```
LD_LIBRARY_PATH=TPT-API-LIBRARY-LOCATION:$LD_LIBRARY_PATH
NLSPATH=TPT-API-MESSAGE-CATALOG-LOCATION
```

Install the SSH Key to Access the Appliance

The system administrator of the SAS High-Performance Analytics appliance provides you with an SSH key. This key is necessary for connecting to the appliance and needs to be deposited on the client machine in the location shown as follows:

UNIX or Linux client	SAS reads the private key from ∼*/.ssh/id_rsa*.
Windows client	SAS searches for the key in *%HOMEDRIVE%\%HOMEPATH%\.ssh\id_rsa* and then in *%USERPROFILE%\.ssh\id_rsa*. For example, in a Windows 7 environment, keys are located in *c:\Users\<userid>\.ssh*.

Running with an Alternate User Name and Identity

The user name and the user's key file can by specified directly by setting the TKSSH_USER environment variable to an alternate user name and by setting the TKSSH_IDENTITY environment variable to the full path and filename of an alternate identity file.

Environment variables can be set on the client machine prior to running the SAS System, or they can be set in the SAS program with the OPTION SET= programming statement. For example, the following statements set an alternate user name and identity in the SAS program:

```
options set=TKSSH_USER="altuser";
options set=TKSSH_IDENTITY="/home/altuser/.ssh/alt_rsa";
```

Appliance Installation

Configuring the appliance for SAS High-Performance Analytics involves several simple steps, which are discussed in the following subsections:

1 Ensure that the hostname (both short names and long names) of every node can be resolved by all nodes in the environment.

2 Create a file with a list of all the machine names in the appliance.

3 Install the SAS executables via a Resource Package Manager (RPM) package or self-extracting shell script.

4 Set up user accounts and SSH keys for the users who will access SAS High-Performance Analytics on the appliance.

5 Prepare the database management system for interaction with SAS High-Performance Analytics.

Create a File with All Machine Names

Before the SAS High-Performance Analytics components can be installed on the appliance nodes, you must create a file that contains a list of the machines in the appliance.

The RPM installation assumes that the file is created in */etc/gridhosts*. The shell script installation prompts for its location.

This file lists the host names that resolve to IP addresses for the Message Passing Interface (MPI) communication on the appliance. The head node is listed first.

A typical host file on Greenplum appliances contains the following host names:

```
mdw
sdw1
sdw2
sdw3
sdw4
...
```

A typical host file on Teradata appliances contains the following host names:

```
dbccop0
dbccop1
dbccop2
dbccop3
...
```

Install the SAS Executables

You can install server-side components of SAS High-Performance Analytics via either an RPM package or a self-extracting shell script. The RPM package and the shell script can be found in the *High_Performance_Analytics_Grid\1_3* directory of the install depot image created by your site administrator.

Install via RPM Package

RPM packages can be installed using a variety of delivery methods:

Local
: The RPM is copied to the local machine and installed from there. For example, enter the following:

```
rpm -ivh /path/to/rpmfile
```

HTTP
: The RPM package is made available through a Web server. For example, enter the following:

```
rpm -ivh http://somewebserver/path/to/rpmfile}
```

FTP
: The RPM package is pulled from an FTP server. For example, enter the following:

```
rpm -ivh ftp://someftpserver/path/to/rpmfile}
```

Yum
: The package is added to a Yum repository and pulled from there. Information about using Yum is available at `http://yum.baseurl.org/`.

RHN
: The RPM package is added to an RHN satellite and distributed via the Red Hat Network (RHN). Further information about the RHN satellite is available at `http://www.redhat.com/red_hat_network`.

In the remainder of this document, delivery via the HTTP protocol is assumed.

The RPM package is installed by *root*. To install:

1 Copy the RPM package to a location on the Web server that is available to Web browsers. Standard Apache installations on Red Hat Enterprise Linux use the directory */var/www/html*. From an account with privileges to write to the document root */var/www/html*, copy the RPM file to that directory and change the permission on the file to **chmod 644**.

2 From the host node of the High-Performance Analytics appliance, test that the RPM package is available via the Web server by issuing the following command:

```
rpm -qpi http://someserver/somedir/rpmfile
```

This command echoes package information but does not install the package.

3 After you have verified connectivity, execute the actual installation by issuing the following command on every node:

```
rpm -ivh http://someserver/somedir/rpmfile
```

The RPM installs the SAS High-Performance Analytics components in the directory */opt/TKGrid_$builddate* and links */opt/TKGrid* to that directory. If the */opt/TKGrid* link already exists, it will not be changed.

Install via Self-Extracting Shell Script

To install server-side components of SAS High-Performance Analytics via a self-extracting shell script:

1 Copy the file *TKHPAGrid_Linux_x86_64.sh* to the */tmp* directory of the head node of the appliance.

2 Log on to the head node of the appliance, and change directories to the desired install location (for example, */opt/HPA_1.1*).

3 Run the shell script in this directory. The shell script creates the *TKGrid* subdirectory and places all files under that directory.

4 Respond to the prompts from the configuration program.

Following extraction of the files, the configuration program executes and prompts for the settings as follows:

```
Shared install or replicate to each node? (Y=SHARED/n=replicated)
```

If you are installing to a local drive on each node, indicate that this is a replicated installation. If you are installing to a drive that is shared across all the nodes (for example, NFS), choose the shared installation.

```
MPICH2 1.3.2 is included in this install. Would you like to use it? (Y/n)
```

Select 'Yes' unless you want to use a different version of the Message Passing Interface (MPI).

```
Enter additional options to mpirun.
```

Typically, there are no additional options that you need to add to each **mpirun** command. In that case, simply press RETURN.

```
Enter path to use for Utility files. (default is /tmp).
```

SAS High-Performance Analytics applications might write scratch files. By default, these files are created in the */tmp* directory. You can redirect the files to a different location by entering the path at the prompt.

```
Force Root Rank to run on headnode? (y/N)
```

If the appliance resides behind a firewall and only the head node can connect back to the client machines, select 'Yes'; otherwise accept the default ('No').

```
Enter full path to machine list...
```

Enter the name of the file you created in the section "Create a File with All Machine Names" on page 512, (for example, */etc/gridhosts*).

```
Enter maximum runtime for grid jobs (in seconds). Default 7200 (2 hours).
```

If a SAS High-Performance Analytics application executes for more than the maximum allowable run time, it is automatically terminated. You can adjust that run-time limit here.

If you selected a replicated installation at the first prompt, you are now prompted to choose the technique for distributing the contents to the appliance nodes:

```
The install can now copy this directory to all the machines
listed in '<filename>' using scp, skipping the first entry.
Perform copy?  (YES/no)
```

Select 'Yes' if you want the installation program to perform the replication. Select 'No' if you are distributing the contents of the installation directory by some other technique.

Set Up User Accounts and SSH Keys

Each user must have an operating system account on each node of the appliance in addition to database accounts that are discussed in subsequent sections. Ideally, a user account should have the same user identifier (uid) and group identifier (gid) across all the nodes.

1 After account creation, generate an SSH key for the user by running this command as the new user:

```
ssh-keygen -t rsa
```

Press ENTER for all prompts and leave the passphrase blank. The operation of SAS High-Performance Analytics requires passwordless SSH between the host node and the other nodes on the appliance.

2 After your key pair has been created, add the public key to the list of authorized keys by appending *~/.ssh/id_rsa.pub* to the list of authorized keys in *~/.ssh/authorized_keys2* with a command such as this:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys2
```

3 The file permissions on the *.ssh* directory and its contents need to be set properly. The *.ssh* directory and files contained in it should have read/write privileges for the user only (**chmod 600** for the *id_rsa* file and **chmod 700** for the *.ssh* directory).

4 Copy the *.ssh* directory to the user's home directory on all the nodes in the appliance.

5 Verify that the SSH keys are correct by using the **ssh** command to access all available nodes. The SSH connection should be successful without entering the user's password.

6 You can verify that the user account has been set up properly by using the following command, where *#nodes* denotes the number of nodes in the system, to run an MPI job from the host node of the appliance:

```
export LD_LIBRARY_PATH=/opt/TKGrid/mpich2-install/lib
/opt/TKGrid/mpich2-install/bin/mpirun -f /opt/TKGrid/grid.hosts \
                                      -n \#nodes hostname
```

In the following example, *#nodes* has been replaced by 24 to indicate that the system has 24 nodes:

```
export LD_LIBRARY_PATH=/opt/TKGrid/mpich2-install/lib
/opt/TKGrid/mpich2-install/bin/mpirun -f /opt/TKGrid/grid.hosts \
                     -n \24 hostname
```

7 Provide the private key in ~/*.ssh/id_rsa* to the user. The client can then complete installation of the key in the client home directory as discussed in the section "Install the SSH Key to Access the Appliance" on page 511. The user's user name on the client is also used as the user name on the appliance.

Prepare the Greenplum Database

SAS Protocol and Related Functions

The SAS High-Performance Analytics interface to the Greenplum database is supported by several functions that are associated with a dedicated SAS protocol. To enable the functionality, the database administrator (a superuser such as the *gpadmin* role) should perform the following steps:

1 Install the formatter functions:

```
CREATE OR REPLACE FUNCTION formatter_export(record)
       RETURNS bytea
       AS '/opt/TKGrid/lib/gpformatter.so', 'formatter_export'
       LANGUAGE C STABLE;

CREATE OR REPLACE FUNCTION formatter_import()
       RETURNS record
       AS '/opt/TKGrid/lib/gpformatter.so', 'formatter_import'
       LANGUAGE C STABLE;
```

2 Create the protocol functions and the SAS protocol:

```
CREATE OR REPLACE FUNCTION gpdb_to_sas() RETURNS integer
       AS '/opt/TKGrid/lib/sas_gpext.so', 'sasprot_export'
       LANGUAGE C STABLE;

CREATE OR REPLACE FUNCTION sas_to_gpdb() RETURNS integer
       AS '/opt/TKGrid/lib/sas_gpext.so', 'sasprot_import'
       LANGUAGE C STABLE;

CREATE TRUSTED PROTOCOL sas(readfunc='sas_to_gpdb', writefunc='gpdb_to_sas');
```

The functions must be created in a schema that is either in your schema search path or in the global *pg_catalog* catalog.

Each database role that executes SAS High-Performance Analytics code against the Greenplum database needs to be granted execution rights for the SAS protocol, as described in the next section.

3 These steps need to be repeated for each database that is accessed through SAS High-Performance Analytics procedures. You can define the functions on a database template from which new databases are derived.

Database Roles

1 If multiple users access the SAS High-Performance Analytics environment on the Greenplum database, it is recommended that you set up a group role and associate the database roles for individual users with the group. The Greenplum database administrator can then associate access to the environment at the group level. For example:

```
CREATE GROUP sas_cust_group NOLOGIN;
ALTER ROLE sas_cust_group CREATEEXTTABLE;
```

2 For each user, create a database role and associate it with the group:

```
CREATE ROLE megan  LOGIN IN ROLE sas_cust_group PASSWORD 'megan';
CREATE ROLE calvin LOGIN IN ROLE sas_cust_group PASSWORD 'calvin';
```

3 If a resource queue exists, associate the roles with the queue:

```
CREATE RESOURCE QUEUE sas_cust_queue WITH
               (MIN_COST=10000.0   ,
               ACTIVE_STATEMENTS=20,
               PRIORITY=HIGH       ,
               MEMORY_LIMIT='4GB'  );

ALTER ROLE megan  RESOURCE QUEUE sas_cust_queue;
ALTER ROLE calvin RESOURCE QUEUE sas_cust_queue;
```

4 An important step is to grant the database roles execution rights on the SAS High-Performance Analytics protocol:

```
GRANT ALL ON PROTOCOL sas TO megan;
GRANT ALL ON PROTOCOL sas TO calvin;
```

Prepare the Teradata Database

Install User-Defined Functions

Following the installation, you need to install Teradata user-defined functions (UDFs) by using the scripts provided in the RPM package. On the host node, run the following command:

```
sh /opt/TKGrid/bin/add_udfs.sh
```

You are prompted for a database account name with sufficient privileges to add user-defined functions.

To uninstall the UDFs from Teradata you can follow a similar procedure and execute the following command:

```
sh /opt/TKGrid/bin/rem_udfs.sh
```

Grant Privileges for Database Accounts

The database users who execute SAS High-Performance Analytics code on Teradata need to have the following privileges:

- execute on SAS_SYSFNLIB
- select on SAS_SYSFNLIB
- execute function on SAS_SYSFNLIB

Updating an RPM Installation

An RPM package does not overwrite an existing installation if it has a different build date. You can update the SAS High-Performance Analytics components by running the **rpm** command on all nodes as was done for the initial installation.

You need to supply the *gridhosts* file to the new installation directory after the */opt/TKGrid* link has been reset, as described in the section “Create a File with All Machine Names” on page 512.

On Teradata systems, you should uninstall the UDFs prior to the RPM update and re-install the UDFs using the scripts delivered with the update; see the section “Install User-Defined Functions” on page 517 for details.

Uninstall from the Appliance

To uninstall the SAS High-Performance Analytics components from the appliance, use the **rpm** command to determine which High-Performance Analytics versions have been installed:

```
rpm -qa | grep tkhpa
```

Using the convention that package names contain the build date, you can determine which of the packages to uninstall. On every node where SAS High-Performance Analytics is to be uninstalled, execute the following command:

```
rpm -e $packagename1 $packagename2 ...
```

Verify that the package has been removed by checking that the appropriate */opt/TKGrid_$builddate* directory has been removed or by using the following command:

```
rpm -qa tkhpa
```

Subject Index

Syntax Index

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.