THE POWER TO KNOW.

SAS/STAT® 9.2 User's Guide

Shared Concepts and Topics (Book Excerpt)

SAS® Documentation

This document is an individual chapter from *SAS/STAT® 9.2 User's Guide*.

The correct bibliographic citation for the complete manual is as follows: SAS Institute Inc. 2008. *SAS/STAT® 9.2 User's Guide*. Cary, NC: SAS Institute Inc.

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

1st electronic book, March 2008
2nd electronic book, February 2009

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Chapter 18

Shared Concepts and Topics

Contents

This chapter introduces a number of concepts that are common to one or more procedures in SAS/STAT, such as the parameterization of effects, the use of the experimental EFFECT statement, and so on. The beginning of each major section displays a listing of the procedures for which the shared topic is relevant.

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*.

A sufficient, but not necessary, condition for a procedure to perform levelization of classification variables is the presence of a CLASS statement. Hence, this section applies to all SAS/STAT procedures that support a CLASS statement. Some procedures use different syntax elements to request levelization of variables (for example, the TRANSREG procedure).

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 the GENMOD, GLMSELECT, and LOGISTIC procedures, you can also control the sorting order separately for each variable in the CLASS statement.

Consider the data on nine observations in Table 18.1. The variable A is integer valued, and variable X is a continuous variable with a missing value for the fourth observations. The fourth and fifth columns of Table 18.1 apply two different formats to the variable X.

Table 18.1 Example Data for Levelization

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

By default, levelization of the variables groups observations by the formatted value of the variable, except for numerical variables where no explicit format is provided. These are sorted by their internal value. The levelization of the four columns in table Table 18.1 leads to the level assignment in Table 18.2.

Table 18.2 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 ORDER= option in the PROC statement specifies the sorting order for the levels of CLASS variables. When the default ORDER=FORMATTED 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 this format explicitly for the CLASS variables.

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

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

For FORMATTED 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*.

The GLMSELECT, LOGISTIC, and GENMOD procedures support a MISSING option in the CLASS statement. When this option is in effect, missing values ('.' for a numeric variable and blanks for a character variable) are included in the levelization and are assigned a level. Table 18.3 displays the results of levelizing the values in Table 18.1 when the MISSING option is in effect.

Table 18.3 Values and Levels with 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, or for procedures whose CLASS statement does not support this option, it is important to understand the implications of missing values for your statistical analysis. When a SAS/STAT procedure levelizes the CLASS variables, an observation for which a 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 where 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;
```

Many statistical 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. You should 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.

Parameterization of Model Effects

Recall the general form of a linear regression model defined in Chapter 3, "Regression Models and Models with Classification Effects":

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

This section describes how matrices of regressor effects such as $\mathbf{X}$ are constructed in SAS/STAT. These constructions, or *parameterization* rules, apply to regression models, models with classification effects, generalized linear models, and mixed models. The simplest and most general parameterization rules are the ones used in the GLM procedure, and they are discussed first. Several procedures also support alternate parameterizations of classification variables, including the GENMOD,

GLMSELECT, LOGISTIC, SURVEYLOGISTIC, and PHREG procedures. These are discussed after the GLM parameterization of classification variables and model effects.

All modeling procedures that support classification variables and effects have a CLASS statement. Procedures that additionally support the supplemental parameterizations have a PARAM= option in the CLASS statement.

GLM Parameterization of Classification Variables and Effects

This section applies to the following procedures:
GAM, GENMOD, GLIMMIX, GLM, GLMPOWER, GLMSELECT, LIFEREG, LOGISTIC, MI, MIXED, MULLTEST, ORTHOREG, PHREG, PLS, QUANTREG, ROBUSTREG, and SURVEYLOGISTIC.

Intercept

By default, linear models in SAS/STAT automatically include a column of 1s in $\mathbf{X}$ corresponding to an intercept parameter. In many procedures you can use the NOINT option in the MODEL statement to suppress this intercept. The NOINT option is useful, for example, 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 involving 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 instance, X|X|X expands to X X*X X*X*X, 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 frequently can be controlled with the ORDER= option in the procedure or CLASS statement.

Table 18.4 is an example where β_0 denotes the intercept and A and B are classification variables with two and three levels, respectively.

Table 18.4 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 18.5). Note that in the MIXED and GLIMMIX procedures, which support 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 18.5 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 SAS/STAT software is more of a notational convenience than an operation distinct from crossing. Nested effects are typically characterized by the property that the nested variables never 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 18.6).

Table 18.6 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 18.7).

Table 18.7 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 18.8 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 18.7.

Table 18.8 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 as the CONTRAST or ESTIMATE statements in a number of procedures, used to estimate and test functions of the parameter.

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 slowest because it is leftmost in the nested list.

Other Parameterizations

This section applies to the following procedures:
GENMOD, GLMSELECT, LOGISTIC, and PHREG.

Some SAS/STAT procedures, including GENMOD, GLMSELECT, and LOGISTIC, support nonsingular parameterizations for classification effects. A variety of these nonsingular parameterizations are available. In these procedures you use the PARAM= option in the CLASS statement to specify the parameterization.

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

EFFECT

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

Effect Coding

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

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

Note that the EFFECT parameterization is the default parameterization in the CATMOD procedure. See the section "Generation of the Design Matrix" on page 1153, in Chapter 28, "The CATMOD Procedure," for further details about parameterization of model effects with the CATMOD procedure.

GLM

As in the GLM procedure, four columns are created to indicate group membership. The design matrix columns for A are as follows.

GLM Coding				
	Design Matrix			
A	**A1**	**A2**	**A5**	**A7**
1	1	0	0	0
2	0	1	0	0
5	0	0	1	0
7	0	0	0	1

Parameter estimates of CLASS main effects that use the GLM coding scheme estimate the difference in the effects of each level compared to the last level. See the previous section for details about the GLM parameterization of model effects.

ORDINAL

THERMOMETER

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

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

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

POLYNOMIAL

POLY

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

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

REFERENCE

REF

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

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

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

The REFERENCE parameterization is also available through the MODEL statement in the CATMOD procedure. See the section "Generation of the Design Matrix" on page 1153, in Chapter 28, "The CATMOD Procedure," for further details about parameterization of model effects with the CATMOD procedure.

ORTHEFFECT

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

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

ORTHORDINAL

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

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

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

Orthogonal Polynomial Coding			
	Design Matrix		
A	**AOPOLY1**	**AOPOLY2**	**AOPOLY5**
1	−1.153	0.907	−0.921
2	−0.734	−0.540	1.473
5	0.524	−1.370	−0.921
7	1.363	1.004	0.368

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

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

Constructed Effects and the EFFECT Statement (Experimental)

This section applies to the following procedures:
GLIMMIX, GLMSELECT, and QUANTREG.

The experimental EFFECT statement in SAS 9.2 is supported by the GLIMMIX, GLMSELECT, and QUANTREG procedures. The EFFECT statement enables you to construct special collections of columns for design matrices. These collections are referred to as *constructed effects* to distinguish them from the usual model effects formed from continuous or classification variables, as discussed in the section "GLM Parameterization of Classification Variables and Effects" on page 369. For example, the terms A, B, x, A*x, A*B, and sub in the following statements define fixed, random, and subject effects of the usual type in a mixed model, respectively:

```
proc glimmix;
   class A B sub;
   model y = A B x A*x;
   random A*B / subject=sub;
run;
```

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

```
proc glimmix;
   class A B SUB;
   effect spl = spline(x);
   model y = A B A*spl;
   random A*B / subject=sub;
run;
```

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

Each constructed effect corresponds to a collection of columns that are referred to by using the name you supply. You can specify multiple EFFECT statements, and all EFFECT statements must precede the MODEL statement.

The general syntax for the EFFECT statement with *effect-specification* is

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

The name of the effect is specified after the EFFECT keyword. This name can appear in only one EFFECT statement and cannot be the name of a variable in the input data set. The effect type is specified after an equal sign, followed by a list of variables used in constructing the effect within parentheses. Effect-type specific options can be specified after a slash (/) following the variable list.

The following *effect-types* are available and subsequently discussed.

COLLECTION	is a collection effect defining one or more variables as a single effect with multiple degrees of freedom. The variables in a collection are considered as a unit for estimation and inference.
MULTIMEMBER | MM	is a multimember classification effect whose levels are determined by one or more variables that appear in a CLASS statement.
POLYNOMIAL | POLY	is a multivariate polynomial effect in the specified numeric variables.
SPLINE	is a regression spline effect whose columns are univariate spline expansions of one or more variables. A spline expansion replaces the original variable with an expanded or larger set of new variables.

Collection Effects

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

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

Multimember Effects

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

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

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

- nurses' effects on patient recovery in hospitals
- teachers' effects on student scores
- lineage effects in genetic studies (see Example 38.16 in Chapter 38, "The GLIMMIX Procedure," for an application with random multimember effects in a genetic diallel experiment)

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

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

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

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

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

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

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

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

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

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

DETAILS
requests a table showing the levels of the multimember effect.

NOEFFECT
specifies that, for observations with all missing levels of the multimember variables, the values in the corresponding design matrix columns are set to zero. If, in the preceding example, the teacher effect is defined by

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

then the associated design matrix columns values for Jack are all zero. This enables you to include Jack in the analysis even though there is no effect of teachers on his performance.

A situation where it is important to designate observations as having no effect due to a classification variable is the analysis of crossover designs, where lagged treatment levels are used to

model the carryover effects of treatments between periods. Since there is no carryover effect for the first period, the treatment lag effect in a crossover design can be modeled with a multimember effect that consists of a single classification variable and the NOEFFECT option, as in the following statements:

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

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

STDIZE
specifies that for each observation, the entries in the design matrix corresponding to the multimember effect are scaled to have a sum of one.

WEIGHT=(*wght-list***)**
specifies numeric variables used to weigh the contributions of each of the classification effects that define the constructed multimember effect. The number of variables in the WEIGHT= list must match the number of classification variables defining the effect.

Polynomial Effects

This section discusses the construction of multivariate polynomial effects through the experimental EFFECT statement in the GLIMMIX and GLMSELECT procedures. You request a polynomial effect with the syntax

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

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

The variables in *var-list* must be numeric. A design matrix column is generated for each term of the specified polynomial. By default, each of these terms is treated as a separate effect for the purpose of model building. For example, the statements

```
proc glmselect;
   effect MyPoly = polynomial(x1-x3/degree=2);
   model y = MyPoly;
run;
```

yield the identical analysis to the statements

```
proc glmselect;
   model y = x1 x2 x3 x1*x1 x1*x2 x1*x3 x2*x2 x2*x3 x3*x3;
run;
```

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

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

DETAILS

requests a table showing the details of the specified polynomial, including the number of terms generated. If you specify the STANDARDIZE option, then a table showing the standardization details is also produced.

LABELSTYLE=(*style-opts***)**

LABELSTYLE=*style-opt*

specifies how the terms in the polynomial are labeled. By default, powers are shown with ^ as the exponentiation operator and ∗ as the multiplication operator. For example, a polynomial term such as $x_1^3 x_2 x_3^2$ is labeled x1^3*x2*x3^2. You can change the style of the label by using the following *style-opts* within parentheses. If you specify a single *style-opt*, then you can omit the enclosing parentheses.

EXPAND

specifies that each variable with an exponent greater than one is written as products of that variable. For example, the term $x_1^3 x_2 x_3^2$ receives the label x1*x1*x1*x2*x3*x3.

EXPONENT < =*quoted string* **>**

specifies that each variable with an exponent greater than one is written using exponential notation. By default, the symbol ^ is used as the exponentiation operator. If you supply the optional quoted string after an equal sign, then that string is used as the exponentiation operator. For example, if you specify

```
LABELSTYLE=(EXPONENT="**")
```

then the term $x_1^3 x_2 x_3^2$ receives the label x1**3*x2*x3**2.

INCLUDENAME

specifies that the name of the effect followed by an underscore is used as a prefix for term labels. For example, the statement

```
EFFECT MyPoly=POLYNOMIAL(x1/degree=2 labelstyle=INCLUDENAME)
```

generates terms with labels MyPoly_x1 and MyPoly_x1^2. The INCLUDENAME option is ignored if you specify the NOSEPARATE option in the EFFECT=POLYNOMIAL statement.

PRODUCTSYMBOL =NONE | *quoted string*

specifies that the supplied string be used as the product symbol. For example, the statement

```
EFFECT MyPoly=POLYNOMIAL(x1 x2 / degree=2 mdegree=1
                                 labelstyle=(PRODUCTSYMBOL=" "))
```

generates terms with labels x1, x2, and x1 x2.

If you specify PRODUCTSYMBOL=NONE, then the labels are formed by juxtaposing the constituent variable names.

MDEGREE=*n*

specifies the maximum degree of any variable in a term of the polynomial. This degree must

be a positive integer. The default is the degree of the specified polynomial. For example, the statement

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

generates the terms x_1, x_2, x_1^2, x_1x_2, x_2^2,$x_1^2x_2$, and $x_1x_2^2$.

NOSEPARATE

specifies that the polynomial is treated as a single effect with multiple degrees of freedom. The effect name that you specify is used as the constructed effect name and the labels of the terms are used as labels of the corresponding parameters.

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

specifies that the variables defining the polynomial are standardized. By default, the standardized variables receive prefix "s_" in the variable names.

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

METHOD=MOMENTS

specifies that the center is estimated by the variable mean and the scale is estimated by the standard deviation. Note that if a weight variable is specified using a WEIGHT statement, the observations with invalid weights are ignored when forming the mean and standard deviation, but the weights are otherwise not used. Note that only observations that are used in performing the analysis are used for the standardization.

METHOD=RANGE

specifies that the center is estimated by the midpoint of the variable range and the scale is estimated as half the variable range. Any observation that has a missing value for any regressor used in the model is ignored when computing the range of variables in a polynomial effect. Observations with valid regressor values but missing or invalid values of frequency variables, weight variables, or dependent variables are used in computing variable ranges. The default (if you do not specify the METHOD= suboption) is METHOD=RANGE.

METHOD=WMOMENTS

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

Let

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

Table 18.9 shows how the center and scale are computed for each of the supported methods:

Table 18.9 Center and Scale Estimates by Method

Method	Center	Scale
Range	$(x_{(n)} - x_{(1)})/2$	$(x_{(n)} + x_{(1)})/2$
Moments	$\bar{x} = \Sigma_{i=1}^{n} f_i x_i / F$	$\sqrt{\Sigma_{i=1}^{n} f_i (x_i - \bar{x})^2/(F-1)}$
WMoments	$\bar{x}_w = \Sigma_{i=1}^{n} w_i f_i x_i / WF$	$\sqrt{\Sigma_{i=1}^{n} w_i f_i (x_i - \bar{x}_w)^2/(F-1)}$

PREFIX=NONE | *quoted string*

specifies the prefix that is appended to standardized variables when forming the term labels. If you omit this option, the default prefix is "s_". If you specify PREFIX=NONE, then standardized variables are not prefixed.

You can control whether the standardization is to center, scale, or both center and scale by specifying a *standardize-opt*:

CENTER

specifies that variables are centered but not scaled. For a variable x,

$$\text{s\_x} = x - \text{center}$$

CENTERSCALE

specifies that variables are centered and scaled. This is the default if you do not specify a *standardization-opt*. For a variable x,

$$\text{s\_x} = \frac{x - \text{center}}{\text{scale}}$$

NONE

specifies that no standardization is performed.

SCALE

specifies that variables are scaled but not centered. For a variable x,

$$\text{s\_x} = \frac{x}{\text{scale}}$$

Spline Effects

This section discusses the construction of spline effects through the experimental EFFECT statement in the GLIMMIX, GLMSELECT, and QUANTREG procedures. You can also include spline effects in statistical models by other means. The TRANSREG procedure has dedicated facilities for including regression splines in your model and controlling the construction of the splines. For example, you can use the TRANSREG procedure to fit a spline function but restrict the function to be always increasing or decreasing (monotone). See the section "Using Splines and Knots" on page 7184 in Chapter 90, "The TRANSREG Procedure," for more information about using splines with the TRANSREG procedure. The GAM and TPSPLINE procedures also can model the effects of regressor variables in terms of smooth functions that are generated from spline bases. For more information see Chapter 36, "The GAM Procedure," and Chapter 89, "The TPSPLINE Procedure."

A spline effect expands variables into spline bases whose form depends on the options that you specify. You can find details about regression splines and spline bases in the section "Splines and Spline Bases" on page 387. You request a spline effect with the syntax

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

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

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

BASIS=BSPLINE

specifies a B-spline basis for the spline expansion. For splines of degree d defined with n knots, this basis consists of $n + d + 1$ columns. In order to completely specify the B-spline basis, d left-side boundary knots and $\max\{d, 1\}$ right-side boundary knots are also required. See the suboptions KNOTMETHOD=, DATABOUNDARY=, KNOTMIN=, and KNOTMAX= for details about how to specify the positions of both the internal and boundary knots. This is the default if you do not specify the BASIS= suboption.

BASIS=TPF(*options***)**

specifies a truncated power function basis for the spline expansion. For splines of degree d defined with n knots for a variable x, this basis consists of an intercept, polynomials x, $x^2,\ldots,x^d$ and one truncated power function for each of the n knots. Note that unlike the B-spline basis no boundary knots are required. See the suboption KNOTMETHOD= for details about how you can specify the position of the internal knots.

You can modify the number of columns when you request BASIS=TPF with the following suboptions:

NOINT
excludes the intercept column.

NOPOWERS
excludes the intercept and polynomial columns.

DATABOUNDARY
specifies that the extremes of the data be used as boundary knots when building a B-spline basis.

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

DETAILS
requests tables showing the knot locations and the knots associated with each spline basis function.

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

KNOTMETHOD=*knot-method***<(***knot-options***)>**
specifies how the knots for spline effects are constructed. You can choose from the following *knot-methods* and affect the knot construction further with the type-specific *knot-options*:

EQUAL<(*n***)>**
specifies that n equally spaced knots be positioned between the extremes of the data. The default is $n = 3$. For a B-spline basis, any needed boundary knots continue to be equally spaced unless the DATABOUNDARY option has also been specified. KNOTMETHOD=EQUAL is the default if no *knot-method* is specified.

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

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

MULTISCALE<(*multiscale-options***)>**
specifies that multiple B-spline bases be generated, corresponding to sets with an increasing number of internal knots. As you increase the number of internal knots, the spline basis you generate is able to approximate features of the data at finer scales. So, by generating bases at multiple scales, you facilitate the modeling of both coarse- and fine-grained features of the data. For scale i, the spline basis corresponds to 2^i equally

spaced internal knots. By default, the bases for scales 0–7 are generated. For each scale, a separate spline effect is generated. The name of the constructed spline effect at scale *i* is formed by appending _S*i* to the effect name you specify in the EFFECT statement. If you specify multiple variables in the EFFECT statement, then spline bases are generated separately for each variable at each scale and the name of the corresponding effect is obtained by appending the variable name followed by _S*i* to the name in the EFFECT statement. For example, the following statement generates effects named spl_x1_S0, spl_x1_S1, spl_x1_S2, . . ., spl_x1_S7 and spl_x2_S1, spl_x2_S2, . . ., spl_x2_S7:

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

This option is ignored if you specify the BASIS=TPF suboption. It is not available for spline effects specified in the RANDOM statement of the GLIMMIX procedure.

You can control which scales are included with the following *multiscale-options*:

STARTSCALE=*n*
where *n* is a positive integer. The default is STARTSCALE=0.

ENDSCALE=*n*
where *n* is a positive integer. The default is ENDSCALE=7.

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

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

RANGEFRACTIONS(*fraction-list***)**
requests that internal knots be placed at each fraction of the ranges of the variables in the EFFECT statement. For example, if variable x1 ranges between 1 and 3, and variable x2 ranges between 0 and 20, then the following EFFECT statement uses internal knots 1.2, 2, and 2.5 for variable x1 and internal knots 2, 10, and 15 for variable x2:

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

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

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

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

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

In procedures that support variable selection, such as the GLMSELECT procedure, these two effects can enter or leave the model independently during the selection process. Separated effects are not supported in the RANDOM statement of the GLIMMIX procedure

SPLIT

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

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

The SPLIT option is not supported in the GLIMMIX procedure.

Splines and Spline Bases

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

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

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

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

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

A counting argument yields the number of parameters that define a spline with n knots. There are $n+1$ polynomials of degree d, giving $(n+1)(d+1)$ coefficients. However, there are d restrictions at each of the n knots, so the number of free parameters is $(n+1)(d+1) - nd = n + d + 1$. In mathematical terminology this says that the dimension of the vector space of splines of degree d on n distinct knots is $n + d + 1$. If you have $n + d + 1$ basis vectors, then you can fit a curve to your

data by regressing your dependent variable by using this basis for the corresponding design matrix columns. In this context, such a spline is known as a regression spline. The EFFECT statement provides a simple mechanism for obtaining such a basis.

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

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

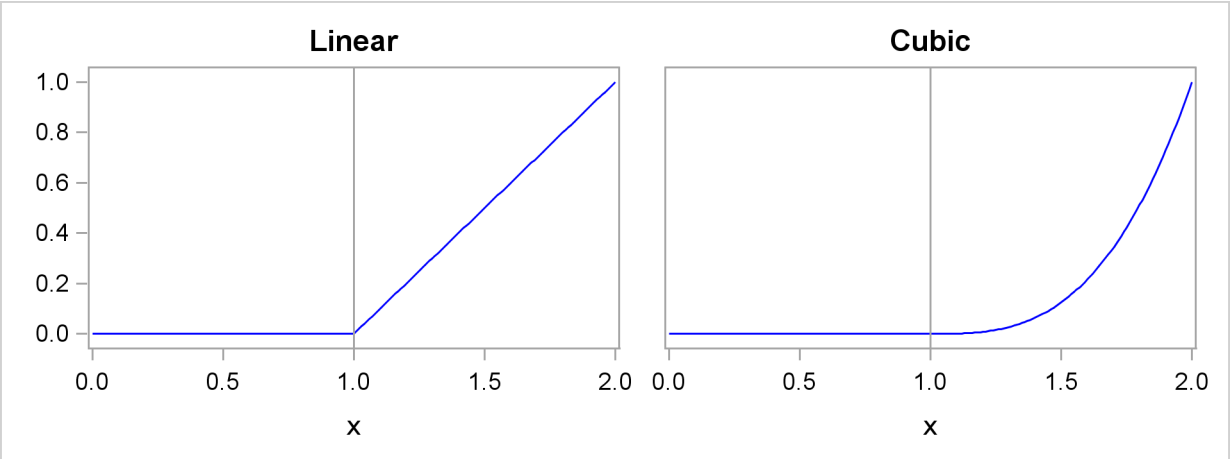
Truncated Power Function Basis

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

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

Figure 18.1 shows such functions for $d = 1$ and $d = 3$ with a knot at $x = 1$.

Figure 18.1 Truncated Power Functions with Knot at $x = 1$



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

These $d + 1$ functions together with the n truncated power functions $t_i(x), i = 1, 2, \ldots, n$ form the truncated power basis.

Note that each time a knot is repeated, the associated exponent used in the corresponding basis function is reduced by 1. For example, for splines of degree d with 3 repeated knots $k_i = k_{i+1} = k_{i+2}$ the corresponding basis functions are $t_i(x) = ((x - k_i)_+^d$, $t_{i+1}(x) = ((x - k_i)_+^{d-1}$, and $t_{i+2}(x) = ((x - k_i)_+^{d-2}$. Provided that the multiplicity of each repeated knot is less than or equal to the degree, this construction continues to yield a basis for the associated space of splines.

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

B-Spline Basis

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

1. The B-spline basis functions are nonzero over an interval spanning at most $d + 2$ knots. This yields design matrix columns each of whose rows contain at most $d + 2$ adjacent nonzero entries.

2. The computation of the basis functions at any x value is numerically stable and does not require evaluating powers of this value.

The following figures show the B-spline bases defined on [0, 1] with 4 equally spaced internal knots at 0.2, 0.4, 0.6, and 0.8.

Figure 18.2 shows a linear B-spline basis. Note that this basis consists of 6 functions each of which is nonzero over an interval spanning at most 3 knots.

Figure 18.2 Linear B-Spline Basis with Four Equally Spaced Interior Knots

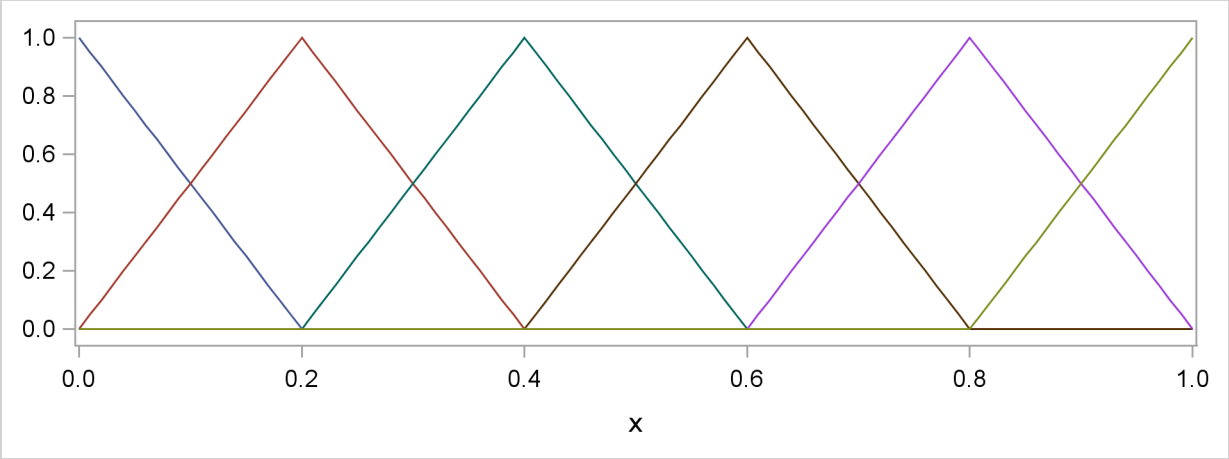


Figure 18.3 shows a cubic B-spline basis where the needed boundary knots are positioned at $x = 0$ and $x = 1$. Note that this basis consists of 8 functions, each of which is nonzero over an interval spanning at most 5 knots.

Figure 18.3 Cubic B-Spline Basis with Four Equally Spaced Interior Knots

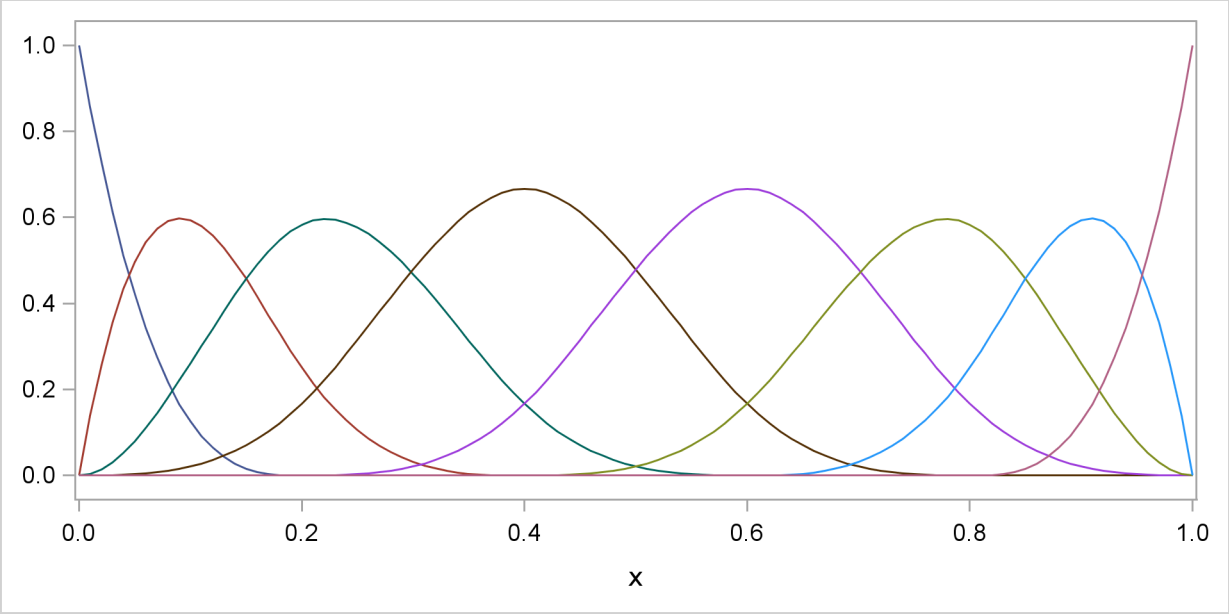
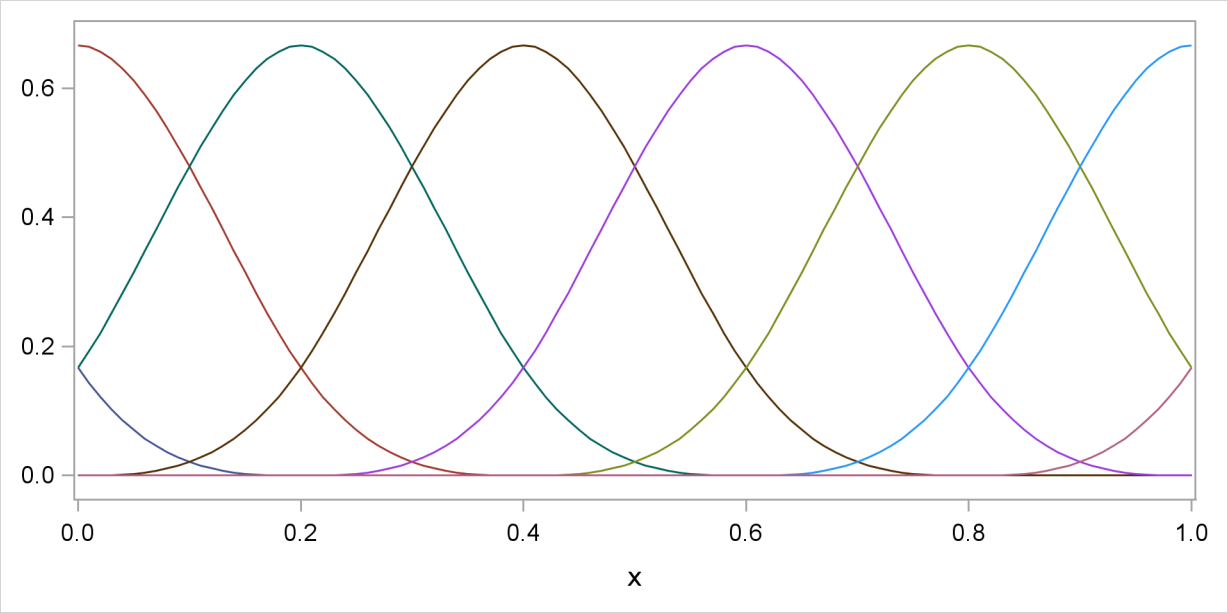


Figure 18.4 shows a different cubic B-spline basis where the needed left-side boundary knots are positioned at −0.6, −0.4, −0.2, and 0. The right-side boundary knots are positioned at 1, 1.2, 1.4, and 1.6. Note that, as in the basis shown in Figure 18.3, this basis consists of 8 functions, each of which is nonzero over an interval spanning at most 5 knots. The different positioning of the boundary knots has merely changed the shape of the individual basis functions.

Figure 18.4 Cubic B-Spline Basis with Equally Spaced Boundary and Interior Knots



You can find details about this construction in Hastie, Tibshirani, and Friedman (2001).

Nonlinear Optimization: The NLOPTIONS Statement

This section applies to the following procedures:
GLIMMIX, HPMIXED, and TCALIS. See the individual procedure chapters on deviations from the common syntax and defaults shown here.

Syntax

The NLOPTIONS statement provides you with syntax to control aspects of the nonlinear optimizations in the GLIMMIX, HPMIXED, and TCALIS procedures.

NLOPTIONS < *options* > **;**

The nonlinear optimization options are described in alphabetical order after Table 18.10, which summarizes the options by category. The notation used in describing the options is generic in the sense that $\boldsymbol{\psi}$ denotes the $p \times 1$ vector of parameters for the optimization and ψ_i is its ith element. The objective function being minimized, its $p \times 1$ gradient vector, and its $p \times p$ Hessian matrix are denoted as $f(\boldsymbol{\psi})$, $\mathbf{g}(\boldsymbol{\psi})$, and $\mathbf{H}(\boldsymbol{\psi})$, respectively. The gradient with respect to the ith parameter is denoted as $g_i(\boldsymbol{\psi})$. Superscripts in parentheses denote the iteration count; for example, $f(\boldsymbol{\psi})^{(k)}$ is the value of the objective function at iteration k. In the mixed model procedures, the

parameter vector ψ might consist of fixed effects only, covariance parameters only, or fixed effects and covariance parameters. In the TCALIS procedure, ψ consists of all independent parameters defined in the models and in the PARAMETERS statement.

Table 18.10 Options to Control Aspects of the Optimization

Option	Description
Optimization	
HESCAL=	determines the type of Hessian scaling
INHESSIAN=	specifies the start for approximated Hessian
LINESEARCH=	specifies the line-search method
LSPRECISION=	specifies the line-search precision
RESTART=	specifies the iteration number for update restart
TECHNIQUE=	determines the minimization technique
UPDATE=	determines the update technique
Termination Criteria	
ABSCONV=	tunes an absolute function convergence criterion
ABSFCONV=	tunes an absolute function convergence criterion
ABSGCONV=	tunes the absolute gradient convergence criterion
ABSXCONV=	tunes the absolute parameter convergence criterion
FCONV=	tunes the relative function convergence criterion
FCONV2=	tunes another relative function convergence criterion
FSIZE=	specifies the value used in FCONV, GCONV criterion
GCONV=	tunes the relative gradient convergence criterion
GCONV2=	tunes another relative gradient convergence criterion
MAXFUNC=	specifies the maximum number of function calls
MAXITER=	specifies the maximum number of iterations
MAXTIME=	specifies the upper limit seconds of CPU time
MINITER=	specifies the minimum number of iterations
XCONV=	specifies the relative parameter convergence criterion
XSIZE=	specifies the value used in XCONV criterion
Step Length	
DAMPSTEP=	dampens steps in line search
INSTEP=	specifies the initial trust region radius
MAXSTEP=	specifies the maximum trust region radius
Printed Output	
PALL	displays (almost) all printed output
PHISTORY	displays optimization history
NOPRINT	suppresses all printed output
Covariance Matrix Tolerances	
ASINGULAR=	absolute singularity for inertia
MSINGULAR=	relative M singularity for inertia
VSINGULAR=	relative V singularity for inertia

Table 18.10 *continued*

Option	Description
Constraint Specifications	
LCEPSILON=	range for active constraints
LCDEACT=	LM tolerance for deactivating
LCSINGULAR=	tolerance for dependent constraints
Remote Monitoring	
SOCKET=	specifies the fileref for remote monitoring

ABSCONV=*r*

ABSTOL=*r*

specifies an absolute function convergence criterion. For minimization, termination requires $f(\boldsymbol{\psi}^{(k)}) \leq r$. 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* **<** *n* **>**

ABSFTOL=*r* **<** *n* **>**

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

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

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:

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

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.

ABSXCONV=*r* **<** *n* **>**

ABSXTOL=*r* **<** *n* **>**

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

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

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

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

or a small simplex size,

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

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

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

The default is r =1E−8 for the NMSIMP technique and $r = 0$ otherwise. The optional integer value n specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

ASINGULAR=*r*

ASING=*r*

specifies an absolute singularity criterion for the computation of the inertia (number of positive, negative, and zero eigenvalues) of the Hessian and its projected forms. The default value is the square root of the smallest positive double-precision value.

DAMPSTEP<=*r***>**

specifies that the initial step length value $\alpha^{(0)}$ for each line search (used by the QUANEW, CONGRA, or NEWRAP technique) cannot be larger than r times the step length value used in the former iteration. If the DAMPSTEP option is specified but r is not specified, the default is $r = 2$. The DAMPSTEP=*r* option can prevent the line-search algorithm from repeatedly stepping into regions where some objective functions are difficult to compute or where they could lead to floating-point overflows during the computation of objective functions and their derivatives. The DAMPSTEP=*r* option can save time-consuming function calls during the line searches of objective functions that result in very small steps.

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,

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

where FSIZE is defined by the FSIZE= option. 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 by default $-\log_{10}\{\epsilon\}$ and ϵ is the machine precision. Some procedures, such as the GLIMMIX procedure, enable you to change the value with the FDIGITS= option in the PROC statement. The optional integer value n specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

FCONV2=*r*< *n* >

FTOL2=*r*< *n* >

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

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

of the objective function. The predicted reduction

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

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

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

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

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

where $\overline{f}(\boldsymbol{\psi}^{(k)}) = \frac{1}{p+1}\sum_l f(\boldsymbol{\psi}_l^{(k)})$. If there are p_{act} boundary constraints active at $\boldsymbol{\psi}^{(k)}$, the mean and standard deviation are computed only for the $n+1-p_{act}$ unconstrained vertices. The default value is $r =$1E−6 for the NMSIMP technique and $r = 0$ otherwise. The optional integer value n specifies the number of successive iterations for which the criterion must be satisfied before the process can terminate.

FSIZE=*r*

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

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,

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

where FSIZE is defined by the FSIZE= option. For the CONGRA technique (where a reliable Hessian estimate $\mathbf{H}$ is not available), the following criterion is used:

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

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.

GCONV2=*r*< *n* >

GTOL2=*r*< *n* >

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

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

This criterion is not used by the other techniques. 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 terminate.

HESCAL=0 | 1 | 2 | 3

HS=0 | 1 | 2 | 3

specifies the scaling version of the Hessian (or crossproduct Jacobian) matrix used in NRRIDG, TRUREG, LEVMAR, NEWRAP, or DBLDOG optimization.

If HS is not equal to 0, the first iteration and each restart iteration set the diagonal scaling matrix $D^{(0)} = \text{diag}(d_i^{(0)})$:

$$d_i^{(0)} = \sqrt{\max(|H_{i,i}^{(0)}|, \epsilon)}$$

where $H_{i,i}^{(0)}$ are the diagonal elements of the Hessian (or crossproduct Jacobian). In every other iteration, the diagonal scaling matrix $D^{(0)} = \text{diag}(d_i^{(0)})$ is updated depending on the HS option:

HS=0 specifies that no scaling be done.

HS=1 specifies the Moré (1978) scaling update:

$$d_i^{(k+1)} = \max\left[d_i^{(k)}, \sqrt{\max(|H_{i,i}^{(k)}|, \epsilon)}\right]$$

HS=2 specifies the Dennis, Gay, and Welsch (1981) scaling update:

$$d_i^{(k+1)} = \max\left[0.6 * d_i^{(k)}, \sqrt{\max(|H_{i,i}^{(k)}|, \epsilon)}\right]$$

HS=3 specifies that d_i be reset in each iteration:

$$d_i^{(k+1)} = \sqrt{\max(|H_{i,i}^{(k)}|, \epsilon)}$$

In each scaling update, ϵ is the relative machine precision. The default value is HS=0. Scaling of the Hessian can be time-consuming in the case where general linear constraints are active.

INHESSIAN< =*r* >

INHESS< =*r* >

specifies how the initial estimate of the approximate Hessian is defined for the quasi-Newton techniques QUANEW and DBLDOG. There are two alternatives:

- If you do not use the r specification, the initial estimate of the approximate Hessian is set to the Hessian at $\psi^{(0)}$.
- If you do use the r specification, the initial estimate of the approximate Hessian is set to the multiple of the identity matrix $r\mathbf{I}$.

By default, if you do not specify the option INHESSIAN=r, the initial estimate of the approximate Hessian is set to the multiple of the identity matrix $r\mathbf{I}$, where the scalar r is computed from the magnitude of the initial gradient.

INSTEP=*r*

SALPHA=*r*

RADIUS=*r*

reduces the length of the first trial step during the line search of the first iterations. For highly nonlinear objective functions, such as the EXP function, the default initial radius of the trust-region algorithm TRUREG or DBLDOG or the default step length of the line-search algorithms can result in arithmetic overflows. If this occurs, you should specify decreasing values of $0 < r < 1$ such as INSTEP=1E−1, INSTEP=1E−2, INSTEP=1E−4, and so on, until the iteration starts successfully.

- For trust-region algorithms (TRUREG, DBLDOG), the INSTEP= option specifies a factor $r > 0$ for the initial radius $\Delta^{(0)}$ of the trust region. The default initial trust-region radius is the length of the scaled gradient. This step corresponds to the default radius factor of $r = 1$.
- For line-search algorithms (NEWRAP, CONGRA, QUANEW), the INSTEP= option specifies an upper bound for the initial step length for the line search during the first five iterations. The default initial step length is $r = 1$.
- For the Nelder-Mead simplex algorithm, by using TECH=NMSIMP, the INSTEP=r option defines the size of the start simplex.

LCDEACT=*r*

LCD=*r*

specifies a threshold r for the Lagrange multiplier that determines whether an active inequality constraint remains active or can be deactivated. For maximization, r must be greater than zero; for minimization, r must be smaller than zero. An active inequality constraint can be deactivated only if its Lagrange multiplier is less than the threshold value. The default value is

$$r = \pm \min(0.01, \max(0.1 \times \text{ABSGCONV}, 0.001 \times \text{gmax}^{(k)}))$$

where "+" stands for maximization, "-" stands for minimization, ABSGCONV is the value of the absolute gradient criterion, and $\text{gmax}^{(k)}$ is the maximum absolute element of the gradient or the projected gradient.

LCEPSILON=*r* ≥ 0

LCEPS=*r* ≥ 0

LCE=*r* ≥ 0

specifies the range for active and violated boundary constraints. If the point $\boldsymbol{\psi}^{(k)}$ satisfies the condition

$$|\sum_{j=1}^{k} a_{ij}\psi_j^{(k)} - b_i| \leq r \times (|b_i| + 1)$$

the constraint i is recognized as an active constraint. Otherwise, the constraint i is either an inactive inequality or a violated inequality or equality constraint. The default value is r =1E−8. During the optimization process, the introduction of rounding errors can force the optimization to increase the value of r by a factor of $10, 100, \ldots$. If this happens, it is indicated by a message displayed in the log.

LCSINGULAR=*r* ≥ 0

LCSING=*r* ≥ 0

LCS=*r* $\geq$

specifies a criterion r, used in the update of the QR decomposition, that determines whether an active constraint is linearly dependent on a set of other active constraints. The default value is r =1E−8. The larger r becomes, the more the active constraints are recognized as being linearly dependent. If the value of r is larger than 0.1, it is reset to 0.1.

LINESEARCH=*i*

LIS=*i*

specifies the line-search method for the CONGRA, QUANEW, and NEWRAP optimization techniques. See Fletcher (1987) for an introduction to line-search techniques. The value of i can be $1, \ldots, 8$. For CONGRA, QUANEW, and NEWRAP, the default value is $i = 2$.

LIS=1 specifies a line-search method that needs the same number of function and gradient calls for cubic interpolation and cubic extrapolation; this method is similar to one used by the Harwell subroutine library.

LIS=2 specifies a line-search method that needs more function than gradient calls for quadratic and cubic interpolation and cubic extrapolation; this method is implemented as shown in Fletcher (1987) and can be modified to an exact line search by using the LSPRECISION= option.

LIS=3 specifies a line-search method that needs the same number of function and gradient calls for cubic interpolation and cubic extrapolation; this method is implemented as shown in Fletcher (1987) and can be modified to an exact line search by using the LSPRECISION= option.

LIS=4 specifies a line-search method that needs the same number of function and gradient calls for stepwise extrapolation and cubic interpolation.

LIS=5 specifies a line-search method that is a modified version of LIS=4.

LIS=6 specifies golden section line search (Polak 1971), which uses only function values for linear approximation.

LIS=7 specifies bisection line search (Polak 1971), which uses only function values for linear approximation.

LIS=8 specifies the Armijo line-search technique (Polak 1971), which uses only function values for linear approximation.

LSPRECISION=*r*

LSP=*r*

specifies the degree of accuracy that should be obtained by the line-search algorithms LIS=2 and LIS=3. Usually an imprecise line search is inexpensive and successful. For more difficult optimization problems, a more precise and expensive line search might be necessary (Fletcher 1987). The second line-search method (which is the default for the NEWRAP, QUANEW, and CONGRA techniques) and the third line-search method approach exact line search for small LSPRECISION= values. If you have numerical problems, you should try to decrease the LSPRECISION= value to obtain a more precise line search. The default values are shown in Table 18.11.

Table 18.11 Default Values for Line-Search Precision

TECH=	UPDATE=	LSP Default
QUANEW	DBFGS, BFGS	$r = 0.4$
QUANEW	DDFP, DFP	$r = 0.06$
CONGRA	all	$r = 0.1$
NEWRAP	no update	$r = 0.9$

For more details, see Fletcher (1987).

MAXFUNC=*i*

MAXFU=*i*

specifies the maximum number i of function calls in the optimization process. The default values are as follows:

- TRUREG, NRRIDG, NEWRAP, LEVMAR: 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 are actually performed can exceed the number that is specified by the MAXFUNC= option.

MAXITER=*i*

MAXIT=*i*

specifies the maximum number i of iterations in the optimization process. The default values are as follows:

- TRUREG, NRRIDG, NEWRAP, LEVMAR: 50

- QUANEW, DBLDOG: 200
- CONGRA: 400
- NMSIMP: 1000

These default values are also valid when *i* is specified as a missing value.

MAXSTEP=*r*<*n*>
specifies an upper bound for the step length of the line-search algorithms during the first *n* iterations. By default, *r* is the largest double-precision value and *n* is the largest integer available. Setting this option can improve the speed of convergence for the CONGRA, QUANEW, and NEWRAP techniques.

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. Note that 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.

MSINGULAR=*r* > 0
MSING=*r* > 0
specifies a relative singularity criterion for the computation of the inertia (number of positive, negative, and zero eigenvalues) of the Hessian and its projected forms. The default value is 1E−12.

NOPRINT
suppresses output related to optimization such as the iteration history. The GLIMMIX and HPMIXED procedures do not support printing options in the NLOPTIONS statement.

PALL
displays all optional output for optimization. This option is supported only by the TCALIS procedure.

PHISTORY
PHIST
displays the optimization history. The PHISTORY option is implied if the PALL option is specified. The PHISTORY option is supported only by the TCALIS procedure.

RESTART=*i > 0*

REST=*i > 0*

specifies that the QUANEW or CONGRA algorithm is restarted with a steepest descent/ascent search direction after, at most, i iterations. Default values are as follows:

- CONGRA: UPDATE=PB: restart is performed automatically, i is not used.
- CONGRA: UPDATE≠PB: $i = \min(10p, 80)$, where p is the number of parameters.
- QUANEW: i is the largest integer available.

SINGULAR=*r*

SING=*r*

specifies the singularity criterion r, $r0 \leq r \leq 1$, that is used for the inversion of the Hessian matrix. The default value is 1E−8.

SOCKET=*fileref*

specifies the fileref that contains the information needed for remote monitoring. See the section "Remote Monitoring" on page 403 for more details.

TECHNIQUE=*value*

TECH=*value*

OMETHOD=*value*

OM=*value*

specifies the optimization technique. You can find additional information about choosing an optimization technique in the section "Choosing an Optimization Algorithm" on page 405. Valid values for the TECHNIQUE= option are as follows:

- CONGRA
  performs a conjugate-gradient optimization, which can be more precisely specified with the UPDATE= option and modified with the LINESEARCH= option. When you specify this option, UPDATE=PB by default.
- DBLDOG
  performs a version of double-dogleg optimization, which can be more precisely specified with the UPDATE= option. When you specify this option, UPDATE=DBFGS by default.
- LEVMAR
  performs a highly stable but, for large problems, memory- and time-consuming Levenberg-Marquardt optimization technique, a slightly improved variant of the Moré (1978) implementation. You can also specify this technique with the alias LM or MARQUARDT. In the TCALIS procedure, this is the default optimization technique if there are fewer than 40 parameters to estimate. The GLIMMIX and HPMIXED procedures do not support this optimization technique.
- NMSIMP
  performs a Nelder-Mead simplex optimization. The TCALIS procedure does not support this optimization technique.
- NONE
  does not perform any optimization. This option can be used for the following:

  - to perform a grid search without optimization
  - to compute estimates and predictions that cannot be obtained efficiently with any of the optimization techniques
  - to obtain inferences for known values of the covariance parameters
- NEWRAP
  performs a Newton-Raphson optimization combining a line-search algorithm with ridging. The line-search algorithm LIS=2 is the default method.
- NRRIDG
  performs a Newton-Raphson optimization with ridging.
- QUANEW
  performs a quasi-Newton optimization, which can be defined more precisely with the UPDATE= option and modified with the LINESEARCH= option.
- TRUREG
  performs a trust-region optimization.

UPDATE=*method*
UPD=*method*

specifies the update method for the quasi-Newton, double-dogleg, or conjugate-gradient optimization technique. Not every update method can be used with each optimizer.

The following are the valid methods for the UPDATE= option:

- BFGS
  performs the original Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update of the inverse Hessian matrix.
- DBFGS
  performs the dual BFGS update of the Cholesky factor of the Hessian matrix. This is the default update method.
- DDFP
  performs the dual Davidon, Fletcher, and Powell (DFP) update of the Cholesky factor of the Hessian matrix.
- DFP
  performs the original DFP update of the inverse Hessian matrix.
- PB
  performs the automatic restart update method of Powell (1977) and Beale (1972).
- FR
  performs the Fletcher-Reeves update (Fletcher 1987).
- PR
  performs the Polak-Ribiere update (Fletcher 1987).
- CD
  performs a conjugate-descent update of Fletcher (1987).

VERSION=1 | 2
VS=1 | 2

specifies the version of the quasi-Newton optimization technique with nonlinear constraints.

VS=1 specifies the update of the μ vector as in Powell (1978a, 1978b) (update like VF02AD).

VS=2 specifies the update of the μ vector as in Powell (1982a, 1982b) (update like VMCWD).

The default is VERSION=2.

VSINGULAR=*r* > 0

VSING=*r* > 0

specifies a relative singularity criterion for the computation of the inertia (number of positive, negative, and zero eigenvalues) of the Hessian and its projected forms. The default value is $r = 1\text{E}{-8}$.

XCONV=*r***<** *n***>**

XTOL=*r***<** *n***>**

specifies the relative parameter convergence criterion. For all techniques except NMSIMP, termination requires a small relative parameter change in subsequent iterations:

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

For the NMSIMP technique, the same formula is used, but $\psi_j^{(k)}$ is defined as the vertex with the lowest function value and $\psi_j^{(k-1)}$ is defined as the vertex with the highest function value in the simplex.

The default value is $r = 1\text{E}{-8}$ for the NMSIMP technique and $r = 0$ otherwise. The optional integer value n specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

XSIZE=*r*

specifies the XSIZE parameter of the relative parameter termination criterion, $r \geq 0$. The default value is $r = 0$. For more details, see the XCONV= option.

Remote Monitoring

SAS/EmMonitor is an application for Windows that enables you to monitor a CPU-intensive application running on a remote server. Remote monitoring is support through the NLOPTIONS statement.

On the server side, a FILENAME statement assigns a fileref to a SOCKET-type device that defines the IP address of the client and the port number for listening. The fileref is then specified in the SOCKET= option in the NLOPTIONS statement to control the SAS/EmMonitor application. The following statements show an example of server-side code:

```
filename sock socket 'your.pc.address.com:6943';

proc glimmix data=bigdataset;
   class year;
   model y1 = x1 x2 x3 / dist=gamma link=log;
   random x1 x3 / type=rsmooth knotmethod=kdtree(bucket=8)
                  subject=year;
   nloptions tech=nrridg gconv=2.e-5 socket=sock;
run;
```

On the client side, the SAS/EmMonitor application is started with the following syntax:

EmMonitor *options*

The options are as follows:

-p port_number	defines the port number.
-t title	defines the title of the SAS/EmMonitor window.
-k	keeps the monitor alive when the iteration is completed.

The default port number is 6943.

By default, the PC client displays on its **Graph** tab the objective function of the optimization process as the first Plot Group. The largest absolute gradient can be monitored as the next Plot Group. You can intervene in the optimization process through the **Stop Current** and **Stop All** buttons. Signaling the server by clicking these buttons effectively terminates the optimization. In a singly iterative optimization, clicking either of the two buttons halts the optimization process when the signal is received on the server side. The current parameter estimates are accepted, and postoptimization processing is based on these values. Note that this is different from a condition where the usual termination criterion is not met, such as when the maximum number of iterations is exceeded. If an optimization does not meet the termination criterion, no postprocessing occurs.

In doubly iterative processes, the **Stop All** button terminates the overall optimization process and accepts the current estimates. The **Stop Current** button stops the current optimization and uses the current parameter estimates to start the next optimization.

The server does not need to be running when you start the SAS/EmMonitor application, and you can start or stop it at any time during the iteration process. You need to remember only the port number.

If you do not start the PC client, or if you close it prematurely, it will have no effect on the server side. In other words, the iteration process will continue until one of the criteria for termination is met.

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 18.12 shows which derivatives are required for each optimization technique (FOD: first-order derivatives (=gradient evaluation); SOD: second-order derivatives (=Hessian evaluation)).

Table 18.12 First-Order and Second-Order Derivatives

Algorithm	FOD	SOD
LEVMAR	x	x
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 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 $p(p+1)/2$ 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 where the objective function and the gradient are much faster to evaluate than the Hessian. The QUANEW and DBLDOG algorithms, in general, 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 $p(p+1)/2$ double words).

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, in general, 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 where derivatives are not continuous or are very difficult to compute.

Each optimization method employs 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 will converge if ABSGCONV <1E−5, FCONV $< 10^{-FDIGITS}$, or GCONV <1E−8.

Algorithm Descriptions

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 corresponding 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 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 (dual) 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. Usually, however, NRRIDG 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 are much faster to compute 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 is one of the following, depending upon the value of the UPDATE= option:

- the original quasi-Newton algorithm, which updates an approximation of the inverse Hessian
- the dual quasi-Newton algorithm, which updates the Cholesky factor of an approximate Hessian (default)

You can specify four update formulas with the UPDATE= option:

- DBFGS performs the dual Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update of the Cholesky factor of the Hessian matrix. This is the default.
- DDFP performs the dual Davidon, Fletcher, and Powell (DFP) update of the Cholesky factor of the Hessian matrix.
- BFGS performs the original BFGS update of the inverse Hessian matrix.
- DFP performs the original DFP update of the inverse Hessian matrix.

In each iteration, a line search is performed along the search direction to find an approximate optimum. The default line-search method uses quadratic interpolation and cubic extrapolation to obtain a step size α satisfying 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. You can specify line-search algorithms other than the default with the LIS= option.

The QUANEW algorithm uses its own line-search technique. Not all options and parameters (except the INSTEP= option) controlling the line search in the other algorithms apply here. In several applications, large steps in the first iterations are troublesome. You can use the INSTEP= option to impose an upper bound for the step size α during the first five iterations. You can also use the INHESSIAN[=r] option to specify a different starting approximation for the Hessian. If you specify only the INHESSIAN option, the Cholesky factor of a (possibly ridged) finite-difference approximation of the Hessian is used to initialize the quasi-Newton update process.

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; see Fletcher (1987, p. 107). Thus, the DBLDOG subroutine uses the dual quasi-Newton update but does not perform a line search. You can specify two update formulas with the UPDATE= option:

- DBFGS performs the dual Broyden, Fletcher, Goldfarb, and Shanno update of the Cholesky factor of the Hessian matrix. This is the default.
- DDFP performs the dual Davidon, Fletcher, and Powell update of the Cholesky factor of the Hessian matrix.

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) 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, or NRRIDG technique, which requires 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. You can specify four different update formulas for generating the conjugate directions by using the UPDATE= option:

- PB performs the automatic restart update method of Powell (1977) and Beale (1972). This is the default.
- FR performs the Fletcher-Reeves update (Fletcher 1987).
- PR performs the Polak-Ribiere update (Fletcher 1987).
- CD performs a conjugate-descent update of Fletcher (1987).

The default often behaves best for typical examples, whereas UPDATE=CD can perform poorly.

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 satisfying 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.

Programming Statements

This section applies to the following procedures:
CALIS, GLIMMIX, MCMC, NLIN, NLMIXED, and TCALIS.

The majority of the modeling procedures in SAS/STAT can take advantage of the fact that the statistical model can easily be translated into programming syntax (statements and options). Several procedures, however, require additional flexibility in specifying models—for example, when the model contains general nonlinear functions, when it is necessary to specify complicated restrictions, or when user-supplied expressions need to be evaluated. These procedures, listed at the beginning of the section, support—in addition to the usual procedure statements and options—programming statements that can be used in the SAS DATA step.

The following are valid 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, these programming statements work the same as they do in the SAS DATA step, as documented in *SAS Language Reference: Concepts*. However, there are several 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, whereas the following statement is not supported:

```
do i = 'A','B','C';
```

- Not all procedures support LAG functionality. For example, the GLIMMIX procedure does not support lags.
- The PUT statement, used mostly for program debugging, supports only some of the features of the DATA step PUT statement, and it has some features not available with the DATA step PUT statement:
  - The PUT statement does not support line pointers, factored lists, iteration factors, overprinting, _INFILE_, the colon (:) format modifier, or "$".
  - The PUT statement does support expressions, but the expression must be enclosed in parentheses. For example, the following statement displays the square root of x:

    ```
    put (sqrt(x));
    ```

  - The PUT statement supports the item _PDV_ to display a formatted listing of all variables in the program. For example:

    ```
    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;
```

Please consult the individual chapters for other, procedure-specific differences between programming statements and the SAS DATA step and for procedure-specific details, limitations, and rules.

The LINK statement is used in a program to jump immediately to the label *statement_label* and to continue program execution at that point. It is not used to specify a link function in a generalized linear model.

When coding your programming statements, you should avoid defining variables that begin with an underscore (_), because they might conflict with internal variables created by procedures that support programming statements.

References

Beale, E. M. L. (1972), "A Derivation of Conjugate Gradients," in *Numerical Methods for Nonlinear Optimization*, ed. F. A. Lootsma, London: Academic Press.

Browne, M. W. (1982), "Covariance Structures," in *Topics in Multivariate Analyses*, ed. D. M. Hawkins, New York: Cambridge University Press.

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," *J. Optim. Theory Appl.*, 28, 453–482.

Eskow, E. and Schnabel, R. B. (1991), "Algorithm 695: Software for a New Modified Cholesky Factorization," *Transactions on Mathematical Software*, 17(3), 306–312.

Fletcher, R. (1987), *Practical Methods of Optimization,* Second Edition, Chichester: John Wiley & Sons.

Gay, D. M. (1983), "Subroutines for Unconstrained Minimization," *ACM Transactions on Mathematical Software*, 9, 503–524.

Hastie, T., Tibshirani, R., and Friedman, J. (2001), *The Elements of Statistical Learning,* New York: Springer-Verlag.

Moré, J. J. (1978), "The Levenberg-Marquardt Algorithm: Implementation and Theory," in *Lecture Notes in Mathematics 630*, ed. G.A. Watson, Berlin-Heidelberg-New York: Springer Verlag.

Moré, J. J. and Sorensen, D. C. (1983), "Computing a Trust-Region Step," *SIAM Journal on Scientific and Statistical Computing*, 4, 553–572.

Polak, E. (1971), *Computational Methods in Optimization*, New York: Academic Press.

Powell, J. M. D. (1977), "Restart Procedures for the Conjugate Gradient Method," *Math. Prog.*, 12, 241–254.

Powell, J. M. D. (1978a), "A Fast Algorithm for Nonlinearly Constraint Optimization Calculations," in *Numerical Analysis, Dundee 1977, Lecture Notes in Mathematics 630*, ed. G. A. Watson, Berlin: Springer-Verlag, 144–175.

Powell, J. M. D. (1978b), "Algorithms for Nonlinear Constraints That Use Lagrangian Functions," *Mathematical Programming*, 14, 224–248.

Powell, J. M. D. (1982a), "Extensions to Subroutine VF02AD," in *Systems Modeling and Optimization, Lecture Notes in Control and Information Sciences 38*, ed. R. F. Drenick and F. Kozin, Berlin: Springer-Verlag, 529–538.

Powell, J. M. D. (1982b), "VMCWD: A Fortran Subroutine for Constrained Optimization," *DAMTP 1982/NA4*, Cambridge, England.

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**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn

sas | THE POWER TO KNOW®