Chapter 5
The NLP Procedure

Chapter Table of Contents

OVERVIEW 371

GETTING STARTED 373
Introductory Examples 373

SYNTAX 383
Functional Summary 383
PROC NLP Statement 386
ARRAY Statement 405
BOUNDS Statement 406
BY Statement 407
CRPJAC Statement 407
GRADIENT Statement 408
HESSIAN Statement 409
INCLUDE Statement 410
JACNLC Statement 410
JACOBIAN Statement 411
LABEL Statement 412
LINCON Statement 412
MATRIX Statement 413
MIN, MAX, and LSQ Statement 414
MINQUAD and MAXQUAD Statements 415
NLINCON Statement 416
DECVAR Statement 418
PROFILE Statement 418
Program Statements 420

DETAILS 424
Criteria for Optimality 424
Optimization Algorithms 427
Finite-Difference Approximations of Derivatives 439
Hessian and CRP Jacobian Scaling 441
Testing the Gradient Specification 442
Termination Criteria 443
Active Set Methods 444
Feasible Starting Point 446

Line-Search Methods 447
Restricting the Step Length 447
Computational Problems 449
Covariance Matrix 451
Input and Output Data Sets 455
Displayed Output 463
Missing Values 465
Computational Resources 466

EXAMPLES 469
Example 5.1 Using the DATA= Option 469
Example 5.2 Using the INQUAD= Option 470
Example 5.3 Using the INEST=Option 472
Example 5.4 Restarting an Optimization 473
Example 5.5 Approximate Standard Errors 475
Example 5.6 Maximum Likelihood Weibull Estimation 481
Example 5.7 Simple Pooling Problem 488
Example 5.8 Chemical Equilibrium 497
Example 5.9 Minimize Total Delay in a Network 502

REFERENCES 506

Chapter 5
The NLP Procedure

Overview

The NLP procedure (**N**on**L**inear **P**rogramming) offers a set of optimization techniques for minimizing or maximizing a continuous nonlinear function $f(x)$ of n decision variables, $x = (x_1, \ldots, x_n)^T$ with lower and upper bound, linear and nonlinear, equality and inequality constraints. This can be expressed as solving

$$\begin{array}{lll} & \min_{x \in \mathcal{R}^n} f(x) & \\ \text{subject to} & c_i(x) = 0 & i = 1, \ldots, m_e \\ & c_i(x) \geq 0 & i = m_e + 1, \ldots, m \\ & u_i \geq x_i \geq l_i & i = 1, \ldots, n \end{array}$$

where f is the objective function, the c_i's are the nonlinear functions, and u_i, l_i's are the upper and lower bounds. Problems of this type are found in many settings ranging from optimal control to maximum likelihood estimation.

The NLP procedure provides a number of algorithms for solving this problem that take advantage of special structure on the objective function and constraints. One example is the quadratic programming problem:

$$f(x) = \frac{1}{2} x^T G x + g^T x + b$$
$$\text{subject to} \quad c_i(x) = 0 \quad i = 1, \ldots, m_e$$

where the $c_i(x)$'s are linear functions; $g = (g_1, \ldots, g_n)^T$ and $b = (b_1, \ldots, b_n)^T$ are vectors and G is an $n \times n$ symmetric matrix.

Another example is the least-squares problem:

$$f(x) = \frac{1}{2} \{ f_1^2(x) + \cdots + f_l^2(x) \}$$
$$\text{subject to} \quad c_i(x) = 0 \quad i = 1, \ldots, m_e$$

where the $c_i(x)$'s are linear functions, and $f_1(x), ..., f_m(x)$ are nonlinear functions of x.

The following problems are handled by PROC NLP.

- quadratic programming with an option for sparse problems
- unconstrained minimization/maximization

- constrained minimization/maximization
- linear complementarity problem

The following optimization techniques are supported in PROC NLP.

- Quadratic Active Set Technique
- Trust-Region Method
- Newton-Raphson Method With Line-Search
- Newton-Raphson Method With Ridging
- Quasi-Newton Methods
- Double-Dogleg Method
- Conjugate Gradient Methods
- Nelder-Mead Simplex Method
- Levenberg-Marquardt Method
- Hybrid Quasi-Newton Methods

These optimization techniques require a continuous objective function f, and all but one (NMSIMP) require continuous first-order derivatives of the objective function f. Some of the techniques also require continuous second-order derivatives. There are three ways to compute derivatives in PROC NLP:

- analytically (using a special derivative compiler), the default method
- via finite difference approximations
- via user-supplied exact or approximate numerical functions

Nonlinear programs can be input into the procedure in various ways. The objective, constraint, and derivative functions are specified using the programming statements of PROC NLP. In addition, information in SAS data sets can be used to define the structure of objectives and constraints as well as specify constants used in objectives, constraints and derivatives.

PROC NLP uses data sets to input various pieces of information.

- The DATA= data set enables you to specify data shared by all functions involved in a least-squares problem.
- The INQUAD= data set contains the arrays appearing in a quadratic programming problem.
- The INVAR= data set specifies initial values for the decision variables, the values of constants that are referred to in the program statements, and simple boundary and general linear constraints.
- The MODEL= data set specifies a model (functions, constraints, derivatives) saved at a previous execution of the NLP procedure.

PROC NLP uses data sets to output various results.

- The OUTVAR= data set saves the values of the decision variables, the derivatives, the solution, and the covariance matrix at the solution.
- The OUT= output data set contains variables generated in the program statements defining the objective function as well as selected variables of the DATA= input data set, if available.
- The OUTMODEL= data set saves the programming statements. It can be used to input a model in the MODEL= input data set.

Getting Started

The NLP procedure solves general nonlinear programs. It has several optimizers that are tuned to best perform on a particular class of problems. Guidelines for choosing a particular optimizer for a problem can be found in the "Details" section.

Regardless of the selected optimizer, it is necessary to specify an objective function and constraints that the optimal solution must satisfy. In PROC NLP, the objective function and the constraints are specified by using SAS programming statements that are similar to those used in the SAS DATA step. Some of the differences are discussed in the sections "Program Statements" and the "ARRAY Statement". As with any programming language, there are many different ways to specify the same problem. Some are more economical than others.

Introductory Examples

The following introductory examples illustrate how to get started using the NLP procedure.

An Unconstrained Problem

Consider the simple example of minimizing the Rosenbrock function (Rosenbrock 1960).

$$\begin{aligned} f(x) &= \frac{1}{2}\{100(x_2 - x_1^2)^2 + (1 - x_1)^2\} \\ &= \frac{1}{2}\{f_1^2(x) + f_2^2(x)\}, \quad x = (x_1, x_2) \end{aligned}$$

The minimum function value is $f(x^*) = 0$ at $x^* = (1, 1)$. This problem does not have any constraints.

The following statements can be used to solve this problem:

```
proc nlp;
   min f;
   decvar x1 x2;
   f1 = 10 * (x2 - x1 * x1);
```

```
   f2 = 1 - x1;
   f  = .5 * (f1 * f1 + f2 * f2);
run;
```

The MIN statement identifies the symbol f that characterizes the objective function in terms of $f1$ and $f2$, and the DECVAR statement names the decision variables $X1$ and $X2$. Because there is no explicit optimizing algorithm option specified (TECH=) PROC NLP uses the Newton-Raphson method with ridging, the default algorithm when there are no constraints.

A better way to solve this problem is to take advantage of the fact that f is a sum of squares of $f1$ and $f2$ and to treat it as a least-squares problem. Using the LSQ statement instead of the MIN statement tells the procedure that this is a least-squares problem, which results in the use of one of the specialized algorithms for solving least-squares problems (for example Levenberg-Marquardt).

```
proc nlp;
   lsq f1 f2;
   decvar x1 x2;
   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;
run;
```

The LSQ statement results in the minimization of a function that is the sum of squares of functions that appear in the LSQ statement. The least-squares specification is preferred because it enableless the procedure to exploit the structure in the problem for numeric stability and performance.

```
                    PROC NLP: Least Squares Minimization

                      Levenberg-Marquardt Optimization

                       Scaling Update of More (1978)

                  Parameter Estimates                  2
                  Functions (Observations)             2

                             Optimization Start

Active Constraints                    0  Objective Function                3.25
Max Abs Gradient Element           25.5  Radius                       358.01571195


                                                                        Actual
                                                        Max Abs           Over
        Rest    Func      Act    Objective  Obj Fun Gradient             Pred
 Iter    arts   Calls      Con     Function   Change  Element  Lambda   Change

    1       0       2        0      3.12500   0.1250  50.0000       0   0.0385
    2       0       3        0   3.6214E-29   3.1250 3.62E-14       0    1.000

                            Optimization Results

Iterations                            2  Function Calls                     4
Jacobian Calls                        3  Active Constraints                 0
Objective Function         3.621365E-29  Max Abs Gradient Element   3.619327E-14
Lambda                                0  Actual Over Pred Change            1
Radius                                5

ABSGCONV convergence criterion satisfied.




                    PROC NLP: Least Squares Minimization

                            Optimization Results
                             Parameter Estimates
                                                    Gradient
                                                   Objective
                N Parameter         Estimate        Function

                1 x1                1.000000    -3.61933E-14
                2 x2                1.000000    2.220446E-14

                Value of Objective Function = 3.621365E-29
```

Figure 5.1. Least-Squares Minimization

PROC NLP displays the iteration history and the solution to this least-squares problem as shown in Figure 5.1. It shows that the solution has $x_1 = 1$ and $x_2 = 1$. As expected in an unconstrained problem, the gradient at the solution is very close to 0.

Boundary Constraints on the Decision Variables

Bounds on the decision variables can be used. Suppose, for example, that it is necessary to constrain the decision variables in the previous example to be less than 0.5. That can be done by adding a BOUNDS statement.

```
proc nlp;
   lsq f1 f2;
```

```
   decvar x1 x2;
   bounds x1 - x2 <= .5;
   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;
run;
```

The solution in Figure 5.2 shows that the decision variables meet the constraint bounds.

```
                PROC NLP: Least Squares Minimization

                       Optimization Results
                        Parameter Estimates
                                        Gradient    Active
                                       Objective    Bound
 N Parameter            Estimate        Function    Constraint

 1 x1                   0.500000       -0.500000     Upper BC
 2 x2                   0.250000               0

              Value of Objective Function = 0.125
```

Figure 5.2. Least-Squares with Bounds Solution

Linear Constraints on the Decision Variables

More general linear equality or inequality constraints of the form

$$\sum_{j=1}^{n} a_{ij}x_j\{\leq \mid = \mid \geq\}b_i for i = 1, \ldots, m$$

can be specified in a LINCON statement. For example, suppose that in addition to the bounds constraints on the decision variables it is necessary to guarantee that the sum $x_1 + x_2$ is less than or equal to 0.6. That can be achieved by adding a LINCON statement:

```
proc nlp;
   lsq f1 f2;
   decvar x1 x2;
   bounds x1 - x2 <= .5;
   lincon x1 + x2 <= .6;
   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;
run;
```

The output in Figure 5.3 displays the iteration history and the convergence criterion.

PROC NLP: Least Squares Minimization

Levenberg-Marquardt Optimization

Scaling Update of More (1978)

Parameter Estimates	2
Functions (Observations)	2
Lower Bounds	0
Upper Bounds	2
Linear Constraints	1

Iter	Rest arts	Func Calls	Act Con	Objective Function	Obj Fun Change	Max Abs Gradient Element	Lambda	Actual Over Pred Change
1	0	3	0	8.19877	21.0512	39.5420	0.0170	0.729
2	0	4	0	1.05752	7.1412	13.6170	0.0105	0.885
3	0	5	1	1.04396	0.0136	18.6337	0	0.0128
4	0	6	1	0.16747	0.8765	0.5552	0	0.997
5	0	7	1	0.16658	0.000895	0.000324	0	0.998
6	0	8	1	0.16658	3.06E-10	5.911E-7	0	0.998

Optimization Results

Iterations	6	Function Calls	9
Jacobian Calls	7	Active Constraints	1
Objective Function	0.1665792899	Max Abs Gradient Element	5.9108825E-7
Lambda	0	Actual Over Pred Change	0.998176801
Radius	0.0000532357		

GCONV convergence criterion satisfied.

PROC NLP: Least Squares Minimization

Value of Objective Function = 0.1665792899

Figure 5.3. Least-Squares with Bounds and Linear Constraints Iteration History

Figure 5.4 shows that the solution satisfies the linear constraint. Note that the procedure displays the active constraints (the constraints that are tight) at optimality.

```
                PROC NLP: Least Squares Minimization

                        Optimization Results
                         Parameter Estimates
                                                    Gradient
                                                   Objective
            N Parameter         Estimate          Function

            1 x1                0.423645         -0.312000
            2 x2                0.176355         -0.312001

            Value of Objective Function = 0.1665792899


             Linear Constraints Evaluated at Solution

     1 ACT -8.327E-17  =    0.6000  -    1.0000 * x1  -    1.0000 * x2
```

Figure 5.4. Least-Squares with Bounds and Linear Constraints Solution

Nonlinear Constraints on the Decision Variables

More general nonlinear equality or inequality constraints can be specified using an NLINCON statement. Consider the least-squares problem with the additional constraint

$$x_1^2 - 2x_2 \geq 0$$

This constraint is specified by a new function $c1$ constrained to be greater than or equal to 0 in the NLINCON statement. The function $c1$ is defined in the programming statements.

```
proc nlp tech=QUANEW;
   min f;
   decvar x1 x2;
   bounds x1 - x2 <= .5;
   lincon x1 + x2 <= .6;
   nlincon c1 >= 0;

   c1 = x1 * x1 - 2 * x2;

   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;

   f = .5 * (f1 * f1 + f2 * f2);
run;
```

Not all of the optimization methods support nonlinear constraints. In particular the Levenberg-Marquardt method, the default for LSQ, does not support nonlinear constraints. (For more information about the particular algorithms, see the section "Optimization Algorithms" on page 427.) The Quasi-Newton method is the prime choice for solving nonlinear programs with nonlinear constraints. The option TECH=QUANEW in the PROC NLP statement causes the Quasi-Newton method to be used.

Figure 5.5 shows the iteration history.

```
                    PROC NLP: Nonlinear Minimization

                     Dual Quasi-Newton Optimization

               Modified VMCWD Algorithm of Powell (1978, 1982)

          Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
                  Lagrange Multiplier Update of Powell(1982)

                    Parameter Estimates                2
                    Lower Bounds                       0
                    Upper Bounds                       2
                    Linear Constraints                 1
                    Nonlinear Constraints              1

                              Optimization Start

Objective Function         2.6202630894  Maximum Constraint                    0
                                         Violation
Maximum Gradient of the    20.729163858
Lagran Func


                                                                      Maximum
                                                                     Gradient
                                                                      Element
                                                 Maximum Predicted      of the
                  Function    Objective Constraint  Function    Step Lagrange
   Iter  Restarts    Calls     Function  Violation Reduction    Size Function

      1         0        7      0.45639          0   0.00935   1.000    0.954
      2'        0        8      0.44713          0    0.0904   1.000    0.963
      3         0        9      0.37332          0    0.1396   1.000    0.584
      4         0       11      0.35295          0    0.0466   0.195    0.376
      5         0       12      0.33948          0    0.0194   1.000    0.944
      6         0       13      0.33082          0   0.00148   1.000   0.0701
      7         0       14      0.33008          0  0.000089   1.000   0.0351
      8         0       15      0.33003          0  5.438E-6   1.000  0.00091
      9         0       16      0.33003          0  2.608E-9   1.000  4.78E-6

                             Optimization Results

Iterations                           9  Function Calls                       17
Gradient Calls                      12  Active Constraints                    1
Objective Function         0.3300307168  Maximum Constraint                    0
                                         Violation
Maximum Projected Gradient  2.153257E-6  Value Lagrange Function    0.3300307155
Maximum Gradient of the    2.0904556E-6  Slope of Search Direction  -2.607736E-9
Lagran Func




                    PROC NLP: Nonlinear Minimization

                  Value of Objective Function = 0.3300307168


                  Value of Lagrange Function = 0.3300307155
```

Figure 5.5. Least-Squares with Bounds, Linear and Nonlinear Constraints, Iteration History

Figure 5.6 shows the solution to this problem.

```
                PROC NLP: Nonlinear Minimization

                     Optimization Results
                      Parameter Estimates
                                          Gradient        Gradient
                                         Objective        Lagrange
   N Parameter         Estimate          Function        Function

   1 x1                0.246955          0.753051     0.000002683
   2 x2                0.030493         -3.049335     0.000000663

            Value of Objective Function = 0.33003072


           Value of Lagrange Function = 0.3300307155



            Linear Constraints Evaluated at Solution

1        0.32255  =    0.6000  -    1.0000 * x1  -    1.0000 * x2


                Values of Nonlinear Constraints

                                                Lagrange
     Constraint          Value Residual Multiplier

   [   2 ]    c1_G    2.941E-9 2.941E-9     1.5247 Active  NLIC
```

Figure 5.6. Least-Squares with Bounds, Linear and Nonlinear Constraints, Solution

A Simple Maximum Likelihood Example

The following is a very simple example of a maximum likelihood estimation problem with the log likelihood function:

$$l(\mu, \sigma) = -\log(\sigma) - \frac{1}{2}(\frac{x-\mu}{\sigma})^2$$

The maximum likelihood estimates of the parameters μ and σ is the solution to

$$\max_{\mu,\sigma>0} \sum_i l_i(\mu, \sigma)$$

where

$$l_i(\mu, \sigma) = -\log(\sigma) - \frac{1}{2}(\frac{x_i-\mu}{\sigma})^2$$

In the following DATA step, values for x are input into SAS data set X; this data set provides the values of x_i.

```
data x;
  input x @@;
```

```
datalines;
1 3 4 5 7
;
```

In the following statements, the DATA=X specification drives the building of the objective function. When each observation in the DATA=X data set is read, a new term $l_i(\mu, \sigma)$ using the value of x_i is added to the objective function LOGLIK specified in the MAX statement.

```
proc nlp data=x vardef=n covariance=h pcov phes;
   profile mean sigma / alpha=.5 .1 .05 .01;
   max loglik;
   parms mean=0, sigma=1;
   bounds sigma > 1e-12;
   loglik=-0.5*((x-mean)/sigma)**2-log(sigma);
run;
```

After a few iterations of the default Newton-Raphson optimization algorithm, PROC NLP procedure produces the following results.

```
                  PROC NLP: Nonlinear Maximization

                       Optimization Results
                        Parameter Estimates
                                    Approx                          Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.894427        4.472136        0.006566
2 sigma             2.000000        0.632456        3.162278        0.025031

                       Optimization Results
                        Parameter Estimates
                              Gradient
                             Objective
                              Function

                           -1.33149E-10
                           5.6064146E-9

              Value of Objective Function = -5.965735903
```

Figure 5.7. Maximum Likelihood Estimates

In unconstrained maximization, the gradient (that is, the vector of first derivatives) at the solution must be very close to zero and the Hessian matrix at the solution (that is, the matrix of second derivatives) must have nonpositive eigenvalues.

```
                    Hessian Matrix

                          mean             sigma

mean        -1.250000003        1.33149E-10
sigma        1.33149E-10       -2.500000014

          Determinant = 3.1250000245

     Matrix has Only Negative Eigenvalues
```

Figure 5.8. Hessian Matrix

Under reasonable assumptions, the approximate standard errors of the estimates are the square roots of the diagonal elements of the covariance matrix of the parameter estimates which (because of the COV=H specification) is the same as the inverse of the Hessian matrix:

```
          Covariance Matrix 2: H = (NOBS/d) inv(G)

                          mean             sigma

mean        0.7999999982       4.260769E-11
sigma       4.260769E-11       0.3999999978

                    Factor sigm = 1


          Determinant = 0.3199999975

     Matrix has 2 Positive Eigenvalue(s)
```

Figure 5.9. Covariance Matrix

The PROFILE statement computes the values of the profile likelihood confidence limits on SIGMA and the MEAN as specified.

Matrix has 2 Positive Eigenvalue(s)

Wald and PL Confidence Limits

N	Parameter	Estimate	Alpha	Profile Likelihood Confidence Limits	
1	mean	4.000000	0.500000	3.384431	4.615569
1	mean	.	0.100000	2.305716	5.694284
1	mean	.	0.050000	1.849538	6.150462
1	mean	.	0.010000	0.670351	7.329649
2	sigma	2.000000	0.500000	1.638972	2.516078
2	sigma	.	0.100000	1.283506	3.748633
2	sigma	.	0.050000	1.195936	4.358321
2	sigma	.	0.010000	1.052584	6.064107

Wald and PL Confidence Limits

Wald Confidence Limits

3.396718	4.603282
2.528798	5.471202
2.246955	5.753045
1.696108	6.303892
1.573415	2.426585
0.959703	3.040297
0.760410	3.239590
0.370903	3.629097

Figure 5.10. Confidence Limits

Syntax

The following statements are used with the NLP procedure:

PROC NLP *options* **;**
MIN *function names* **;**
MAX *function names* **;**
LSQ *function names* **;**
MINQUAD *matrix, vector, or number* **;**
MAXQUAD *function names* **;**
DECVAR *function names* **;**
VAR *function names* **;**
PARMS *function names* **;**
PARAMETERS *function names* **;**
ARRAY *function names* **;**
BOUNDS *boundary constraints* **;**
BY *variables* **;**
CRPJAC *variables* **;**
GRADIENT *variables* **;**
HESSIAN *variables* **;**
INCLUDE *model files* **;**
JACNLC *variables* **;**
JACOBIAN *function names* **;**
LABEL *decision variable labels* **;**
LINCON *linear constraints* **;**
MATRIX *matrix specification* **;**
NLINCON *nonlinear constraints* **;**
PROFILE *profile specification* **;**
Program Statements ;

Functional Summary

The following table outlines the options in the NLP statement classified by function.

Description	Option
Input Data Set Specifications	
data set	DATA=
initial values and constraints	INEST=
quadratic objective function	INQUAD=
program statements	MODEL=
skip missing value observations	NOMISS
Output Data Set Specifications	
variables and derivatives	OUT=
result parameter values	OUTEST=
program statements	OUTMODEL=

Description	Option
combining various OUT... statements	OUTALL
CRP Jacobian to OUTEST= data set	OUTCRPJAC
derivatives in the OUT= data set	OUTDER
grid in the OUTEST= data set	OUTGRID
Hessian to OUTEST= data set	OUTHESSIAN
iterative output to the OUTEST= data set	OUTITER
Jacobian in the OUTEST= data set	OUTJAC
NLC Jacobian in the OUTEST= data set	OUTNLCJAC
time in the OUTEST= data set	OUTTIME
Optimization Specifications	
minimization method	TECHNIQUE=
update technique	UPDATE=
version of optimization technique	VERSION=
line-search method	LINESEARCH=
line-search precision	LSPRECISION=
type of Hessian scaling	HESCAL=
start for approximated Hessian	INHESSIAN=
iteration number for update restart	RESTART=
Initial Value Specifications	
best grid point number	BEST=
infeasible points in grid search	INFEASIBLE=
pseudorandom initial values	RANDOM=
constant initial values	INITIAL=
Derivatives Specifications	
finite-difference derivatives	FD[=]
finite-difference derivatives	FDHESSIAN[=]
compute finite-difference interval	FDINT=
use only diagonal of Hessian	DIAHES
test gradient specification	GRADCHECK[=]
Constraint Specifications	
range for active constraints	LCEPSILON=
LM tolerance for deactivating	LCDEACT=
tolerance for dependent constraints	LCSINGULAR=
Termination Criteria Specifications	
maximum number of function calls	MAXFUNC=
maximum number of iterations	MAXITER=
minimum number of iterations	MINITER=
upper limit seconds of CPU time	MAXTIME=
absolute function convergence criterion	ABSCONV=
absolute function convergence criterion	ABSFCONV =

Description	Option
absolute gradient convergence criterion	ABSGCONV=
absolute parameter convergence criterion	ABSXCONV=
relative function convergence criterion	FCONV =
relative function convergence criterion	FCONV 2=
relative gradient convergence criterion	GCONV=
relative gradient convergence criterion	GCONV2=
relative parameter convergence criterion	XCONV=
used in FCONV , GCONV criterion	FSIZE=
used in XCONV criterion	XSIZE=
Covariance Matrix Specifications	
kind of covariance matrix	COVARIANCE=
σ^2 factor of COV matrix	SIGSQ=
determines factor of COV matrix	VARDEF=
absolute singularity for inertia	ASINGULAR=
relative M singularity for inertia	MSINGULAR=
relative V singularity for inertia	VSINGULAR=
threshold for Moore-Penrose inverse	G4
tolerance for singular COV matrix	COVSING=
profile confidence limits	CLPARM=
Printed Output Specifications	
print (almost) all printed output	PALL
suppresses all printed output	NOPRINT
reduces some default output	PSHORT
reduces most default output	PSUMMARY
initial values	PINIT
optimization history	PHISTORY
Jacobian matrix	PJACOBI
crossproduct Jacobian matrix	PCRPJAC
Hessian matrix	PHESSIAN
Jacobian of nonlinear constraints	PNLCJAC
values of grid points	PGRID
values of functions in LSQ, MIN, MAX	PFUNCTION
approximate standard errors	PSTDERR
covariance matrix	PCOV
eigenvalues for covariance matrix	PEIGVAL
prints code evaluation problems	PERROR
model program, variables	LIST
compiled model program	LISTCODE
Step Length Specifications	
damped steps in line-search	DAMPSTEP[=]
maximum trust-region radius	MAXSTEP=
initial trust-region radius	INSTEP=

Description	Option
Miscellaneous Options	
number accurate digits in objective function	FDIGITS=
number accurate digits in nonlinear constraints	CDIGITS=
general singularity criterion	SINGULAR=
do not compute inertia of matrices	NOEIGNUM
check optimality in neighborhood	OPTCHECK[=]

PROC NLP Statement

This statement invokes the NLP procedure.

PROC NLP *options* **;**

The following options are used with the PROC NLP statement.

ABSCONV=r
ABSTOL=r
specifies an absolute function convergence criterion. For minimization (maximization), termination requires $f(x^{(k)}) \leq (\geq) r$. The default value of ABSTOL is the negative (positive) square root of the largest double precision value.

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:

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

For the NMSIMP technique the same formula is used, but $x^{(k)}$ is defined as the vertex with the lowest function value, and $x^{(k-1)}$ 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 the absolute gradient convergence criterion. Termination requires the maximum absolute gradient element to be small:

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

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

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

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 of the simplex vertex $y^{(k)}$ with the smallest function value to the other n simplex points $x_l^{(k)} \neq y^{(k)}$:

$$\delta^{(k)} = \sum_{x_l \neq y} \| x_l^{(k)} - y^{(k)} \|_1$$

The default value is $r = 1e-4$ for the COBYLA NMSIMP technique, $r = 1e-8$ for the standard 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.

ASINGULAR=r
ASING=r

specifies an absolute singularity criterion for measuring singularity of Hessian and crossproduct Jacobian and their projected forms, which may have to be converted to compute the covariance matrix. The default is the square root of the smallest positive double precision value. For more information, see the section "Covariance Matrix" on page 451.

BEST=i

produces the i best grid points only. This option not only restricts the output, it also can significantly reduce the computation time needed for sorting the grid point information.

CDIGITS=r

specifies the number of accurate digits in nonlinear constraint evaluations. Fractional values such as CDIGITS=4.7 are allowed. The default value is r=-log10(ϵ), where ϵ is the machine precision. The value of r is used to compute the interval size h for the computation of finite-difference approximations of the Jacobian matrix of nonlinear constraints.

CLPARM= PL | WALD | BOTH

The CLPARM = option is similar to but not the same as that used by other SAS procedures. Using CLPARM=BOTH is equivalent to specifying

```
PROFILE / ALPHA=0.5 0.1 0.05 0.01 OUTTABLE;
```

The CLMPARM=BOTH option specifies that PL CLs for all parameters and for $\alpha = .5, .1, .05, .01$ are computed and displayed or written to the OUTEST= data set. Computing the profile confidence limits for all parameters can be very expensive and should be avoided when a difficult optimization problem or one with many parameters is solved. The OUTTABLE option is valid only when an OUTEST= data set is specified in the PROC NLP statement. For CLPARM=BOTH, the table of displayed output contains the Wald confidence limits computed from the standard errors as well as the PL CLs. The Wald confidence limits are not computed (displayed or written to the OUTEST= data set) unless the approximate covariance matrix of parameters is computed.

COVARIANCE=1 | 2 | 3 | 4 | 5 | 6 | M | H | J | B | E | U
COV=1 | 2 | 3 | 4 | 5 | 6 | M | H | J | B | E | U

specifies one of six formulas for computing the covariance matrix. For more information, see the section "Covariance Matrix" on page 451.

COVSING=$r > 0$

specifies a threshold that determines whether the eigenvalues of a singular Hessian matrix or crossproduct Jacobian matrix are considered to be zero. For more information, see the section "Covariance Matrix" on page 451.

DAMPSTEP[=r**]**
DS[=r**]**

specifies that the initial step size value $\alpha^{(0)}$ for each line-search (used by the QUANEW, HYQUAN, CONGRA, or NEWRAP technique) cannot be larger than r times the step size value used in the former iteration. If the DAMPSTEP option is specified but not factor r, 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-costly function calls during the line-searches of objective functions that result in very small step. For more information, see the section "Restricting the Step Length" on page 447.

DATA=*SAS-data-set*

allows variables from the specified data set to be used in the specification of the objective function f. For more information, see the section "DATA= Input Data Set" on page 455.

DIAHES

specifies that only the diagonal of the Hessian or crossproduct Jacobian is used. This saves function evaluations but may considerably slow the convergence process. Note that the DIAHES option refers to both the Hessian and the crossproduct Jacobian when using the LSQ statement. When derivatives are specified using the HESSIAN or CRPJAC statement, these statements must refer only to the n diagonal derivative elements (otherwise, the $n(n + 12)/2$ derivatives of the lower triangle must be specified). The DIAHES option is ignored if a quadratic programming with a constant Hessian is specified by TECH=QUADAS or TECH=LICOMP.

FCONV =$r[n]$
FTOL=$r[n]$

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

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

where FSIZE is defined by the FSIZE= option. For the NMSIMP technique, the same formula is used, but $x^{(k)}$ is defined as the vertex with the lowest function value, and $x^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is r=$10^{-FDIGITS}$ where FDIGITS is the value of the FDIGITS= option. The optional integer value n specifies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

FCONV 2=$r[n]$
FTOL2=$r[n]$

specifies another function convergence criterion. For least-squares problems and all techniques except NMSIMP, termination requires a small predicted reduction

$$df^{(k)} \approx f(x^{(k)}) - f(x^{(k)} + s^{(k)})$$

of the objective function. The predicted reduction

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

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

$$s^{(k)} = -G^{(k)-1}g^{(k)}$$

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

$$\sqrt{\frac{1}{n+1}\sum_l (f(x_l^{(k)}) - \overline{f}(x^{(k)}))^2} \leq r$$

where $\overline{f}(x^{(k)}) = \frac{1}{n+1}\sum_l f(x_l^{(k)})$. If there are n_{act} boundary constraints active at $x^{(k)}$, the mean and standard deviation are computed only for the $n+1-n_{act}$ unconstrained vertices. The default value is r=1e-6 for the NMSIMP technique and the QUANEW technique with nonlinear constraints 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.

FD[=*FORWARD | CENTRAL | number***]**
specifies that all derivatives be computed using finite difference approximations. The following specifications are permitted:

FD=FORWARD uses forward differences.

FD=CENTRAL uses central differences.

FD=*number* uses central differences for the initial and final evaluations of the gradient, Jacobian, and Hessian. During iteration, start with forward differences and switch to a corresponding central-difference formula during the iteration process when one of the following two criteria is satisfied:

- The absolute maximum gradient element is less than or equal to *number* times the ABSGTOL threshold.
- The term left of the GTOL criterion is less than or equal to $\max(1.0E-6,$*number*$*$GTOL threshold). The $1.0E-6$ ensures that the switch is done, even if you set the GTOL threshold to zero.

FD is equivalent to FD=100.

Note that the FD and FDHESSIAN options cannot apply at the same time. The FDHESSIAN option is ignored when only first-order derivatives are used, for example, when the LSQ statement is used and the HESSIAN is not explicitly needed (displayed or written to a data set). For more information, see the section "Finite-Difference Approximations of Derivatives" on page 439.

FDHESSIAN[=*FORWARD | CENTRAL***]**
FDHES[=*FORWARD | CENTRAL***]**
FDH[=*FORWARD | CENTRAL***]**
specifies that second-order derivatives be computed using finite difference approximations based on evaluations of the gradients.

FDHESSIAN=FORWARD uses forward differences.

FDHESSIAN=CENTRAL uses central differences.

FDHESSIAN uses forward differences for the Hessian except for the initial and final output.

Note that the FD and FDHESSIAN options cannot apply at the same time. For more information, see the section "Finite-Difference Approximations of Derivatives" on page 439

FDIGITS=*r*
specifies the number of accurate digits in evaluations of the objective function. Fractional values such as FDIGITS=4.7 are allowed. The default value is r=-log10(ϵ), where ϵ is the machine precision. The value of r is used to compute the interval size h for the computation of finite-difference approximations of the derivatives of the objective function and for the default value of the FCONV = option.

FDINT= OBJ | CON | ALL

specifies how the finite difference intervals h should be computed. For FDINT=OBJ, the interval h is based on the behavior of the objective function; for FDINT=CON, the interval h is based on the behavior of the nonlinear constraints functions; and for FDINT=ALL, the interval h is based on the behavior of the objective function and the nonlinear constraints functions. For more information, see the section "Finite-Difference Approximations of Derivatives" on page 439.

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 in the the section "PROC NLP Statement" on page 386.

G4=$n > 0$

The G4= option is used when the covariance matrix is singular. The value n determines which generalized inverse is computed. The default value of n is 60. For more information, see the section "Covariance Matrix" on page 451.

GCONV=$r[n]$
GTOL=$r[n]$

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

$$\frac{g(x^{(k)})^T [G^{(k)}]^{-1} g(x^{(k)})}{\max(|f(x^{(k)})|, \text{FSIZE})} \leq r$$

where FSIZE is defined by the FSIZE= option. For the CONGRA technique (where a reliable Hessian estimate G is not available)

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

is used. This criterion is not used by the NMSIMP technique. The default value is $r = 1e-8$. The optional integer value n speicfies the number of successive iterations for which the criterion must be satisfied before the process can be terminated.

GCONV2=$r[n]$
GTOL2=$r[n]$

specifies another relative gradient convergence criterion.

$$\max_j \frac{|g_j(x^{(k)})|}{\sqrt{f(x^{(k)}) G^{(k)}_{j,j}}} \leq r$$

This option is valid only when using the TRUREG, LEVMAR, NRRIDG, and NEWRAP techniques on least-squares problems. 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.

GRADCHECK[= NONE | FAST | DETAIL]
GC[= NONE | FAST | DETAIL]

Specifying GRADCHECK= DETAIL computes a test vector and test matrix to check whether the gradient g specified by a GRADIENT (or indirectly by a JACOBIAN) statement is appropriate for the function f computed by the program statements. If the specification of the first derivatives is correct, the elements of the test vector and test matrix should be relatively small. For very large optimization problems, the algorithm can be too expensive in terms of computer time and memory. If the GRADCHECK option is not specified, a fast derivative test identical to the GRADCHECK= FAST specification is performed by default. It is possible to suppress the default derivative test by specifying GRADCH=NONE. For more information, see the section "Testing the Gradient Specification" on page 442.

HESCA=$0|1|2|3$
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 the value of the HS=option is not equal to zero, the first iteration and each restart iteration sets the diagonal scaling matrix $D^{(0)} = diag(d_i^{(0)})$:

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

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

HS=0 specifies that no scaling is done.

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

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

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

$$d_i^{(k+1)} = \max(0.6 * d_i^{(k)}, \sqrt{\max(|G_{i,i}^{(k)}|, \epsilon)})$$

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

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

where ϵ is the relative machine precision. The default value is HS=1 for LEVMAR minimization and HS=0 otherwise. Scaling of the Hessian or crossproduct Jacobian matrix can be time-consuming in the case where general linear constraints are active.

INEST=*SAS-data-set*
INVAR=*SAS-data-set*
ESTDATA=*SAS-data-set*

can be used to specify the initial values of the parameters defined in a PARMS or VAR statement as well as simple boundary constraints and general linear constraints. The INEST= data set can contain additional variables with names corresponding to constants used in the program statements. At the beginning of each run of PROC NLP, the values of the constants are read from the PARMS observation, initializing the constants in the program statements. For more information, see the section "INEST= Input Data Set" on page 455.

INFEASIBLE
IFP

specifies that the function values of both feasible and infeasible grid points are to be computed, displayed, and written to the OUTEST= or OUTVAR= data set, although only the feasible grid points are candidates for the starting point $x^{(0)}$. This option enables you to explore the shape of the objective function of points surrounding the feasible region. For the output, the grid points are sorted first with decreasing values of the maximum constraint violation. Points with the same value of the maximum constraint violation are then sorted with increasing (minimization) or decreasing (maximization) value of the objective function. Using the BEST= option restricts only the number of best grid points in the displayed output, not those in the data set. The INFEASIBLE option affects both the displayed output and the output saved to the OUTEST= data set. The OUTGRID option can be used to write the grid points and their function values to an OUTEST= or OUTVAR= data set. After small modifications (deleting unneeded information), this data set can be used with the G3D procedure of the SAS/GRAPH® product to generate a three-dimensional surface plot of the objective function depending on two selected parameters. For more information on grids, see the section "DECVAR Statement" on page 418.

INHESSIAN[=*r***]**
INHESS[=*r***]**

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

- The $= r$ specification is not used: the initial estimate of the approximate Hessian is set to the true Hessian or crossproduct Jacobian at $x^{(0)}$.
- The $= r$ specification is used: the initial estimate of the approximate Hessian is set to the multiple of the identity matrix rI.

By default, if INHESSIAN=r is not specified, the initial estimate of the approximate Hessian is set to the multiple of the identity matrix rI, where the scalar r is computed from the magnitude of the initial gradient. For most applications, this is a sufficiently good first approximation.

INITIAL=*r*

specifies a value r as the common initial value for all parameters for which no other initial value assignments by the PARMS or VAR statement or an INEST= (or INVAR= or ESTDATA=) data set are made. For more information, see the description of the INITIAL option in the section "PROC NLP Statement" on page 386.

INQUAD=*SAS-data-set*

can be used to specify (the nonzero elements of) the matrix H, vector g, and scalar c of a quadratic programming problem, $f(x) = \frac{1}{2}x^T Hx + g^T x + c$. This option cannot be used together with the NLINCON statement. Two forms (*dense* and *sparse*) of the INQUAD= data set can be used. For more information,see the section "INQUAD= Input Data Set" on page 456.

INSTEP=*r*

For highly nonlinear objective functions, such as the EXP function, the default initial radius of the trust-region algorithm TRUREG, DBLDOG, or LEVMAR or the default step length of the line-search algorithms can result in arithmetic overflows. If this occurs, decreasing values of $0 < r < 1$ should be specified, 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, LEVMAR) the INSTE=P 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, HYQUAN) 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, using TECH= NMSIMP, the INSTEP=r option defines the size of the initial simplex.

For more details, see the section "Computational Problems" on page 449.

LCDEACT=*r*

LCD=*r*

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

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

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

LCEPSILON=$r > 0$
LCEPS=$r > 0$
LCE=$r > 0$

specifies the range for active and violated boundary and linear constraints. During the optimization process, the introduction of rounding errors can force PROC NLP to increase the value of r by a factor of 10, 100,... If this happens it is indicated by a message written to the log. For more information, see the section "Linear Complementarity (LICOMP)" on page 432.

LCSINGULAR=$r > 0$
LCSING=$r > 0$
LCS=$r > 0$

specifies a criterion r used in the update of the QR decomposition that decides 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, HYQUAN, and NEWRAP optimization techniques. Refer to 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 $i = 2$. A special line-search method is the default for the least-squares technique HYQUAN that is based on an algorithm developed by Lindström & Wedin (1984). Although it needs more memory, this default line-search method sometimes works better with large least-squares problems. However, by specifying LIS=i, $i = 1, \ldots, 8$, it is possible to use one of the standard techniques with HYQUAN.

LIS=1 specifies a line-search method that needs the same number of function and gradient calls for cubic interpolation and cubic extrapolation.

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.

LIST

displays the model program and variable lists. The LIST option is a debugging feature and is not normally needed. This output is not included in either the default output or the output specified by the PALL option.

LISTCODE

diplays the derivative tables and the compiled program code. The LISTCODE option is a debugging feature and is not normally needed. This output is not included in either the default output or the output specified by the PALL option. The option is similar to that used in MODEL procedure in SAS/ETS software.

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 sufficient for convergence to the optimum. For difficult optimization problems, a more precise and expensive line-search may be necessary (Fletcher 1987). The second (default for NEWRAP, QUANEW, and CONGRA) and third line-search methods approach exact line-search for small LSPRECISION= values. In the presence of numerical problems, it is advised to decrease the LSPRECISION= value to obtain a more precise line-search. The default values are as follows.

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

For more details, refer to Fletcher (1987).

MAXFUNC=i
MAXFU=i

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

- TRUREG, LEVMAR, NRRIDG, NEWRAP: 125
- QUANEW, HYQUAN, DBLDOG: 500
- CONGRA, QUADAS: 1000
- NMSIMP: 3000

Note that the optimization can be terminated 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*[*n*]
MAXIT=*i*[*n*]
specifies the maximum number i of iterations in the optimization process. The default values are:

- TRUREG, LEVMAR, NRRIDG, NEWRAP: 50
- QUANEW, HYQUAN, DBLDOG: 200
- CONGRA, QUADAS: 400
- NMSIMP: 1000

This default value is valid also when i is specified as a missing value. The optional second value n is valid only for TECH= QUANEW with nonlinear constraints. It specifies an upper bound n for the number of iterations of an algorithm used to reduce the violation of nonlinear constraints at a starting point. The default value is n=20.

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 reduce the speed of convergence for TECH=CONGRA, TECH=QUANEW, TECH=HYQUAN, and TECH=NEWRAP.

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 the 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 of the PROC NLP job may be longer than that specified by the MAXTIME= option. The actual running time includes the rest of the time needed to finish the iteration, time for the output of the (temporary) results, and (if required) the time for saving the results in an OUTEST= or OUTVAR= data set. Using the MAXTIME= option with a permanent OUTEST= data set enables you to separate large optimization problems into a series of smaller problems that need smaller amounts of CPU time.

MINITER=*i*
MINIT=*i*
specifies the minimum number of iterations. The default value is zero. If more iterations than are actually needed are requested 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.

MODEL=*model-name, model-list*
MOD=*model-name, model-list*
MODFILE=*model-name, model-list*
reads the program statements from one or more input model files created by previous PROC NLP steps using the OUTMODEL= option. If it is necessary to include the

program code at a special location in newly written code, the INCLUDE statement can be used instead of using the MODEL= option. Using both the MODEL= option and the INCLUDE statement with the same model file will include the same model twice, which can produce different results than including it once. The MODEL= option is similar to the option used in PROC MODEL in SAS/ETS software.

MSINGULAR=$r > 0$
MSING=$r > 0$

specifies a relative singularity criterion for measuring singularity of Hessian and crossproduct Jacobian and their projected forms. The default value is $1e - 12$ if the SINGULAR= option is not specified and $\max(10 * \epsilon$,1e-4$*SINGULAR$ otherwise. For more information, see the section "Covariance Matrix" on page 451.

NOEIGNUM

suppresses the computation and output of the determinant and the inertia of the Hessian, crossproduct Jacobian, and covariance matrices. The inertia of a symmetric matrix are the numbers of negative, positive, and zero eigenvalues. For large applications, the NOEIGNUM option can save computer time.

NOMISS

is valid only for those variables of the DATA= data set that are referred to in program statements. If the NOMISS option is specified, observations with any missing value for those variables are skipped. If the NOMISS option is not specified, the missing value may result in a missing value of the objective function, implying that the corresponding BY group of data is not processed.

NOPRINT
NOP

suppresses the output.

OPTCHECK [=r**]**

computes the function values $f(x_l)$ of a grid of points x_l in a small neighborhood of x^*. The x_l are located in a ball of radius of r about x^*. If the OPTCHECK optioin is specified without r, the default value is $r = 0.1$ at the starting point and $r = 0.01$ at terminating point. If a point x_l^* is found with a better function value than $f(x^*)$, then optimization is restarted at x_l^*. For more information on grids, see the section "DECVAR Statement" on page 418.

OUT=*SAS-data-set*

creates an output data set that contains those variables of a DATA= input data set referred to in the program statements plus additional variables computed by performing the program statements of the objective function, derivatives, and nonlinear constraints. The OUT= data set can also contain first- and second-order derivatives of these variables if the OUTDER= option is specified. The variables and derivatives are evaluated at x^*; for TECH=NONE, they are evaluated at x^0.

OUTALL

if an OUTEST= data set is specified, this option sets the OUTHESSIAN option if the MIN or MAX statement is used. If the LSQ statement is used, the OUTALL option sets the OUTCRPJAC option. If nonlinear constraints are specified using the NLINCON statement, the OUTALL option sets the OUTNLCJAC option.

OUTCRPJAC
if an OUTVAR= data set is specified, the crossproduct Jacobian matrix of the m functions composing the least-squares function is written to the OUTVAR= data set.

OUTDER= 0, 1, 2
specifies whether or not derivatives are written to the OUT= data set. For OUTDER=2, first- and second-order derivatives are written to the data set; for OUTDER=1, only first-order derivatives are written; for OUTDER=0, no derivatives are written to the data set. The default value is OUTDER=0. Derivatives are evaluated at x^*.

OUTVAR=*SAS-data-set*
OUTEST=*SAS-data-set*
creates an output data set that contains the results of the optimization. This is useful for reporting and for restarting the optimzation in a subsequent execution of the procedure. Information in the data set can include parameter estimates, gradient values, constraint information, Lagrangian values, Hessian values, Jacobian values, covariance, standard errors, and confidence intervals.

OUTGRID
writes the grid points and their function values to the OUTEST= data set. By default, only the feasible grid points points are saved; however, if the INFEASIBLE option is specified, all feasible and infeasible grid points are saved. Note that the BEST= option does not affect the output of grid points to the OUTEST= or OUTVAR= data set. For more information on grids, see the section "DECVAR Statement" on page 418.

OUTHESSIAN
OUTHES
writes the Hessian matrix of the objective function to the OUTEST= data set. If the Hessian matrix is computed for some other reason (if, for example, the PHESSIAN option is specified), the OUTHESSIAN option is set by default.

OUTITER
during each iteration writes the parameter estimates, the value of the objective function, the gradient (if available), and (if OUTTIME is specified) the time in seconds from the start of the optimization to the OUTEST= or OUTVAR= data set.

OUTJAC
writes the Jacobian matrix of the m functions composing the least-squares function to the OUTEST= or OUTVAR= data set. If the PJAC option is specified, the OUTJAC option is set by default.

OUTMODEL=*model-name*
OUTMOD=*model-name*
OUTM=*model-name*
the name of an output model file to which the program statements are to be written. The program statements of this file can be included into the program statements of a succeeding PROC NLP run using the MODEL= option or the INCLUDE program statement. The OUTMODEL= option is similar to the option used in PROC MODEL in SAS/ETS software. Note that the following statements are not part of the program code that is written to an OUTMODEL= data set: MIN, MAX, LSQ, MINQUAD,

MAXQUAD, PARMS, BOUNDS, BY, CRPJAC, GRADIENT, HESSIAN, JACNLC, JACOBIAN, LABEL, LINCON, MATRIX, NLINCON.

OUTNLCJAC

if an OUTEST= or OUTVAR= data set is specified, the Jacobian matrix of the nonlinear constraint functions specified by the NLINCON statement is written to the OUTEST= data set. If the Jacobian matrix of the nonlinear constraint functions is computed for some other reason (if, for example, the PNLCJAC option is specified), the OUTNLCJAC option is set by default.

OUTTIME

if an OUTEST= or OUTVAR= data set is specified and if the OUTITER option is specified, during each iteration, the time in seconds from the start of the optimization is written to the OUTEST= or OUTVAR= data set.

PALL

ALL

displays all optional output except the output generated by the PSTDERR , PCOV , LIST, or LISTCODE option.

PCOV

diplays the covariance matrix specified by the COV= option. The PCOV option is set automatically if the PALL and COV= options are set.

PCRPJAC

PJTJ

diplays the $n \times n$ crossproduct Jacobian matrix $J^T J$. If the PALL option is specified and the LSQ statement is used, this option is set automatically. If general linear constraints are active at the solution, the projected crossproduct Jacobian matrix is also displayed.

PEIGVAL

diplays the distribution of eigenvalues if a G4 inverse is computed for the covariance matrix. The PEIGVAL option is useful for observing which eigenvalues of the matrix are recognized as zero eigenvalues when the generalized inverse is computed, and it is the basis for setting the COVSING= option in a subsequent execution of PROC NLP. For more information, see the section “Covariance Matrix” on page 451

PERROR

the PERROR option specifies additional output for such applications where the program code for objective function or nonlinear constraints cannot be evaluated during the iteration process. The PERROR option is set by default during the evaluations at the starting point but not during the optimization process.

PFUNCTION

diplays the values of all functions specified in a LSQ, MIN, or MAX statement for each observation read fom the DATA= input data set. The PALL option sets the PFUNCTION option automatically.

PGRID

diplays the function values from the grid search. For more information on grids, see the section “DECVAR Statement” on page 418.

PHESSIAN
PHES
diplays the $n \times n$ Hessian matrix G. If the PALL option is specified and the MIN or MAX statement is used, this option is set automatically. If general linear constraints are active at the solution, the projected Hessian matrix is also displayed.

PHISTORY
PHIS
diplays the optimization history. No optimization history is displayed for TECH=LICOMP. This output is included in both the default output and the output specified by the PALL option.

PINIT
PIN
diplays the initial values and derivatives (if available). This output is included in both the default output and the output specified by the PALL option.

PJACOBI
PJAC
diplays the $m \times n$ Jacobian matrix J. Because of the memory requirement for large least-squares problems, this option is not invoked by using the PALL option.

PNLCJAC
diplays the Jacobian matrix of nonlinear constraints specified by the NLINCON statement. The PNLCJAC option is set automatically if the PALL option is specified.

PSHORT
SHORT
PSH
restricts the amount of default output. If PSHORT is specified, then

- the initial values are not displayed
- the listing of constraints is not displayed
- if there is more than one function in the MIN, MAX, or LSQ statement, their values are not displayed
- if the GRADCHECK[=DETAIL] option is used, only the test vector is displayed

PSTDERR
STDERR
SE
standard errors that are defined as square roots of the diagonal elements of the covariance matrix. The t values and probabilities $> |t|$ are displayed together with the approximate standard errors. The type of covariance matrix must be specified using the COV= option. The SIGSQ= option, the VARDEF= option, and the special variables _NOBS_ and _DF_ defined in the program statements can be used to define a scalar factor σ^2 of the covariance matrix and the approximate standard errors. For more information, see the section “Covariance Matrix” on page 451.

PSUMMARY
SUMMARY
SUM
restricts the amount of default displayed output to a short form of iteration history and notes, warnings and errors.

PTIME
specifies the output of four different but partially overlapping differences of CPU time:

- total running time
- total time for the evaluation of objective function, nonlinear constraints, and derivatives: shows the total time spent executing the programming statements specifying the objective function, derivatives, and nonlinear constraints, and (if necessary) their firs-t and second-order derivatives. This is the total of the time needed for code evaluation before, during, and after iterating
- total time for optimization shows the total time spent iterating.
- time for some CMP parsing: shows the time needed for parsing the program statements and its derivatives. In most applications this is a negligible number, but for applications that contain ARRAY statements or DO loops or use an optimization technique with analytic second-order derivatives, it can be a considerable.

RANDOM=*i*
specifies a positive integer as a seed value for the pseudorandom number generator. Pseudorandom numbers are used as initial value $x^{(0)}$. For more information, see the section "PROC NLP Statement" on page 386.

RESTART=*i* > 0
REST=*i* > 0
specifies that the QUANEW, HYQUAN, 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 done automatically so specification of i is not used.
- CONGRA: UPDATE$\neq$PB: $i = \min(10n, 80)$, where n is the number of parameters.
- QUANEW, HYQUAN: i is the largest integer available.

SIGSQ=*sq* > 0
specifies a scalar factor σ^2 for computing the covariance matrix. If the SIGSQ= option is specified, VARDEF=N is the default. For more information, see the section "Covariance Matrix" on page 451.

SINGULAR=$r > 0$
SING=$r > 0$
specifies the singularity criterion r for the inversion of the Hessian matrix and crossproduct Jacobian. The default value is $1e - 8$. See the MSINGULAR= and the VSINGULAR= options in the section "PROC NLP Statement" on page 386.

TECHNIQUE=x
TECH=x
specifies the optimization technique. Valid values for it are as follows:

- CONGRA
  chooses one of four different conjugate-gradient optimization algorithms, which can be more precisely specified with the UPDATE= option and modified with the LINESEARCH= option. When this option is selected, UPDATE=PB by default. For $n \geq 400$, CONGRA is the default optimization technique.
- DBLDOG
  performs a version of double dogleg optimization, which can be more precisely specified with the UPDATE= option. When this option is selected, UPDATE=DBFGS by default.
- HYQUAN
  chooses one of three different hybrid quasi-Newton optimization algorithms which can be more precisely defined with the VERSION= option and modified with the LINESEARCH= option. By default,VERSION=2 and UPDATE=DBFGS.
- LM
  performs the Levenberg-Marquardt minimization. For $n < 40$, this is the default minimization technique for least-squares problems.
- LCP
  solves a quadratic program as a linear complementarity problem.
- NMSIMP
  performs the Nelder-Mead simplex optimization method.
- NONE
  does not perform any optimization. This option can be used
  - to do grid search without optimization
  - to compute and display derivatives and covariance matrices which cannot be obtained efficiently with any of the optimization techniques
- NEWRAP
  performs the Newton-Raphson optimization technique. The algorithm combines a line-search algorithm with ridging. The line-search algorithm LIS=2 is the default.

- NRRIDG
  performs the Newton-Raphson optimization technique. For $n \leq 40$ and nonlinear least-squares, this is the default.
- QUADAS performs a special quadratic version of the active set strategy.
- QUANEW
  chooses one of four quasi-Newton optimization algorithms which can be defined more precisely with the UPDATE= option and modified with the LINESEARCH= option. This is the default for $40 < n < 400$ or if there are nonlinear constraints.
- TRUREG
  performs the trust region optimization technique.

UPD=*method*
specifies the update method for the (dual) quasi-Newton, double dogleg, hybrid quasi-Newton, or conjugate-gradient optimization technique. Not every update method can be used with each optimizer. For more information, see the section "Optimization Algorithms" on page 427. Valid values for m are as follows:

BFGS performs the original BFGS (Broyden, Fletcher, Goldfarb, & Shanno) update of the inverse Hessian matrix.

DBFGS performs the dual BFGS (Broyden, Fletcher, Goldfarb, & Shanno) update of the Cholesky factor of the Hessian matrix.

DDFP performs the dual DFP (Davidon, Fletcher, & Powell) update of the Cholesky factor of the Hessian matrix.

DFP performs the original DFP (Davidon, Fletcher, & Powell) 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).

VARDEF=DF,N
specifies the divisor d used in the calculation of the covariance matrix and approximate standard errors. If the SIGSQ= option is not specified, the default value is VARDEF= DF; otherwise, VARDEF=N is default. For more information, see the section "Covariance Matrix" on page 451.

VERSION=1, 2, 3
VS=1, 2, 3
zx specifies the version of the hybrid quasi-Newton optimization technique or the version of the quasi-Newton optimization technique with nonlinear constraints.
For hybrid quasi-Newton optimization technique

VS=1 specifies version HY1 of Fletcher & Xu (1987)

VS=2 specifies version HY2 of Fletcher & Xu (1987)

VS=3 specifies version HY3 of Fletcher & Xu (1987)

For quasi-Newton optimization technique with nonlinear constraints,

VS=1 specifies the update of the μ vector like Powell (1978) (update like VF02AD)

VS=2 specifies the update of the μ vector like Powell (1982) (update like VMCWD)

In both cases, the default value is VS=2.

VSINGULAR=$r > 0$
VSING=$r > 0$
specifies a relative singularity criterion for measuring singularity of Hessian and crossproduct Jacobian and their projected forms, which may have to be converted to compute the covariance matrix. The default value for VSING is $1e-8$ if the SINGULAR= option is not specified and the value of SINGULAR otherwise. For more information, see the section "Covariance Matrix" on page 451.

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,

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

For the NMSIMP technique, the same formula is used, but $x_j^{(k)}$ is defined as the vertex with the lowest function value and $x_j^{(k-1)}$ is defined as the vertex with the highest function value in the simplex. The default value is $r = 1e-8$ for the NMSIMP technique and $r = 0$ otherwise. 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=$r > 0$
specifies the XSIZE parameter of the relative parameter termination criterion. The default value is $r = 0$. For more detail, see the XCONV= option in the section "PROC NLP Statement" on page 386.

ARRAY Statement

ARRAY *arrayname [{ dimensions }] [$] [variables and constants]* **;**

The ARRAY statement is similar to, but not the same as, the ARRAY statement in the SAS DATA step. The ARRAY statement is used to associate a name (of no more

than eight characters) with a list of variables and constants. The array name is used with subscripts in the program to refer to the array elements. The following code illustrates this

```
array r[8] r1-r8;

do i = 1 to 8;
   r[i] = 0;
   end;
```

The ARRAY statement does not support all the features of the DATA step ARRAY statement. It cannot be used to give initial values to array elements. Implicit indexing of variables cannot be used; all array references must have explicit subscript expressions. Only exact array dimensions are allowed; lower-bound specifications are not supported and a maximum of six dimensions is allowed.

On the other hand, the ARRAY statement does allow both variables and constants to be used as array elements. (Constant array elements cannot have values assigned to them.) Both dimension specification and the list of elements are optional, but at least one must be given. When the list of elements is not given or fewer elements than the size of the array are listed, array variables are created by suffixing element numbers to the array name to complete the element list.

BOUNDS Statement

BOUNDS *b_con [, b_con...]* **;**

where *b_con* := number *operator* parameter_list *operator* number
or *b_con* := number *operator* parameter_list
or *b_con* := parameter_list *operator* number
and *operator :=* $<=, <, >=, >, =$

Boundary constraints are specified with a BOUNDS statement. One- or two-sided boundary constraints are allowed. The list of boundary constraints are separated by commas. For example,

```
bounds 0 <= a1-a9 X <= 1, -1 <= c2-c5;
bounds b1-b10 y >= 0;
```

More than one BOUNDS statement can be used. If more than one lower (upper) bound for the same parameter is specified, the maximum (minimum) of these is taken. If the maximum l_j of all lower bounds is larger than the minimum of all upper bounds u_j for the same variable x_j, the boundary constraint is replaced by $x_j := l_j := \min(u_j)$ defined by the minimum of all upper bounds specified for x_j.

BY Statement

BY *variables* **;**

A BY statement can be used with PROC NLP to obtain separate analyses on DATA= data set observations in groups defined by the BY variables. That means, for values of the TECH=option other than NONE, an optimization problem is solved for each BY group separately. When a BY statement appears, the procedure expects the input DATA= data set to be sorted in order of the BY variables. If the input data set is not sorted in ascending order, it is necessary to use one of the following alternatives:

- Use the SORT procedure with a similar BY statement to sort the data.
- Use the BY statement option NOTSORTED or DESCENDING in the BY statement for the NLP procedure. As a cautionary note, the NOTSORTED option does not mean that the data are unsorted but rather that the data are arranged in groups (according to values of the BY variables) and that these groups are not necessarily in alphabetical or increasing numeric order.
- Use the DATASETS procedure (in Base SAS software) to create an index on the BY variables.

For more information on the BY statement, refer to the discussion in *SAS Language Reference: Concepts*. For more information on the DATASETS procedure, refer to the discussion in the *SAS Procedures Guide, Version 6.*

CRPJAC Statement

CRPJAC *variables* **;**

The CRPJAC statement defines the crossproduct Jacobian $J^T J$ matrix used in solving least-squares problems. For more information, see the section "Derivatives" on page 426. If the DIAHES option is not specified, the CRPJAC statement lists $n(n+1)/2$ variable names, which correspond to the elements $J^T J_{j,k}, j \geq k$, of the lower triangle of the symmetric crossproduct Jacobian matrix listed by rows. For example, the statements

```
lsq f1-f3;
decvar x1-x3;
crpjac jj1-jj6;
```

correspond to the crossproduct Jacobian matrix

$$J^T J = \begin{bmatrix} JJ1 & JJ2 & JJ4 \\ JJ2 & JJ3 & JJ5 \\ JJ4 & JJ5 & JJ6 \end{bmatrix}$$

If the DIAHES option is specified, only the n diagonal elements must be listed in the CRPJAC statement. The n rows and columns of the crossproduct Jacobian matrix must be in the same order as the n corresponding parameter names listed in the DECVAR statement. To specify the values of nonzero derivatives, the variables specified in the CRPJAC statement have to be defined at the left-hand side of algebraic expressions in programming statements. For example, consider the Rosenbrock Function:

```
proc nlp tech=levmar;
   lsq f1 f2;
   decvar x1 x2;
   gradient g1 g2;
   crpjac cpj1-cpj3;

   f1   = 10 * (x2 - x1 * x1);
   f2   = 1 - x1;
   g1   = -200 * x1 * (x2 - x1 * x1) - (1 - x1);
   g2   =  100 * (x2 - x1 * x1);

   cpj1 = 400 * x1 * x1 _ 1 ;
   cpj2 = -200 * x1;
   cpj3 = 100;
run;
```

GRADIENT Statement

GRADIENT *variables* **;**

The GRADIENT statement defines the gradient vector which contains the first-order derivatives of the objective function f with respect to $x_1, \ldots, x_n$. For more information, see the section "Derivatives" on page 426. To specify the values of nonzero derivatives, the variables specified in the GRADIENT statement must be defined at the left-hand side of algebraic expressions in programming statements. For example,consider the Rosenbrock function:

```
proc nlp tech=congra;
   min y;
   decvar x1 x2;
   gradient g1 g2;

   y1 = 10 * (x2 - x1 * x1);
   y2 = 1 - x1;

   y  = .5 * (y1 * y1 + y2 * y2);

   g1 = -200 * x1 * (x2 - x1 * x1) - (1 - x1);
   g2 = 100 * (x2 - x1 * x1);
run;
```

HESSIAN Statement

HESSIAN *variables* **;**

The HESSIAN statement defines the Hessian matrix G containing the second-order derivatives of the objective function f with respect to $x_1, \ldots, x_n$. For more information, see the section "Derivatives" on page 426.

If the DIAHES option is not specified, the HESSIAN statement lists $n(n+1)/2$ variable names which correspond to the elements $G_{j,k}, j \geq k$, of the lower triangle of the symmetric Hessian matrix listed by rows. For example, the statements

```
min f;
decvar x1 - x3;
hessian g1-g6;
```

correspond to the Hessian matrix

$$G = \begin{bmatrix} G1 & G2 & G4 \\ G2 & G3 & G5 \\ G4 & G5 & G6 \end{bmatrix} = \begin{bmatrix} \partial^2 f/\partial x_1^2 & \partial^2 f/\partial x_1 \partial x_2 & \partial^2 f/\partial x_1 \partial x_3 \\ \partial^2 f/\partial x_2 \partial x_1 & \partial^2 f/\partial x_2^2 & \partial^2 f/\partial x_2 \partial x_3 \\ \partial^2 f/\partial x_3 \partial x_1 & \partial^2 f/\partial x_3 \partial x_2 & \partial^2 f/\partial x_3^2 \end{bmatrix} .$$

If the DIAHES option is specified, only the n diagonal elements must be listed in the HESSIAN statement. The n rows and columns of the Hessian matrix G must correspond to the order of the n parameter names listed in the DECVAR statement. To specify the values of nonzero derivatives, the variables specified in the HESSIAN statement have to be defined in on the left-hand side of algebraic expressions in the programming statements. For example, consider the Rosenbrock function:

```
proc nlp tech=nrridg;
   min f;
   decvar x1 x2;
   gradient g1 g2;
   hessian h1-h3;

   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;

   f = .5 * (f1 * f1 + f2 * f2);

   g1 = -200 * x1 * (x2 - x1 * x1) - (1 - x1);
   g2 = 100 * (x2 - x1 * x1);

   h1 = -200 * (x2 - 3 * x1 * x1) + 1;
   h2 = -200 * x1;
   h3 = 100;
run;
```

INCLUDE Statement

INCLUDE *model files* **;**

The INCLUDE statement can be used to append model code to the current model code. The contents of included model files, created using the OUTMOUDEL= option, are inserted into the model program at the position in which the INCLUDE statement appears.

JACNLC Statement

JACNLC *variables* **;**

The JACNLC statement defines the Jacobian matrix for the system of constraint functions $c_1(x), \ldots, c_{mc}(x)$. The statements lists the $mc * n$ variable names which correspond to the elements $CJ_{i,j}$, $i = 1, \ldots, mc$, $j = 1, \ldots, n$, of the Jacobian matrix by rows.

For example, the statements

```
nlincon c1-c3;
decvar  x1-x2;
jacnlc  cj1-cj6;
```

correspond to the Jacobian matrix

$$CJ = \begin{bmatrix} CJ1 & CJ2 \\ CJ3 & CJ4 \\ CJ5 & CJ6 \end{bmatrix} = \begin{bmatrix} \partial c_1/\partial x_1 & \partial c_1/\partial x_2 \\ \partial c_2/\partial x_1 & \partial c_2/\partial x_2 \\ \partial c_3/\partial x_1 & \partial c_3/\partial x_2 \end{bmatrix} .$$

The mc rows of the Jacobian matrix must be in the same order as the mc corresponding names of nonlinear constraints listed in the NLINCON statement. The n columns of the Jacobian matrix must be in the same order as the n corresponding parameter names listed in the DECVAR statement. To specify the values of nonzero derivatives, the variables specified in the JACOBIAN statement have to be defined on the left-hand side of algebraic expressions in programming statements.

For example,

```
array cd[3,4] cd1-cd12;
nlincon c1-c3 >= 0;
jacnlc cd1-cd12;

c1 = 8 - x1 * x1 - x2 * x2 - x3 * x3 - x4 * x4 -
       x1 + x2 - x3 + x4;
c2 = 10 - x1 * x1 - 2 * x2 * x2 - x3 * x3 - 2 * x4 * x4 +
       x1 + x4;
c3 = 5 - 2 * x1 * x2 - x2 * x2 - x3 * x3 - 2 * x1 + x2 + x4;
```

```
cd[1,1]= -1 - 2 * x1;   cd[1,2]= 1 - 2 * x2;
cd[1,3]= -1 - 2 * x3;   cd[1,4]= 1 - 2 * x4;
cd[2,1]=  1 - 2 * x1;   cd[2,2]= -4 * x2;
cd[2,3]= -2 * x3;       cd[2,4]= 1 - 4 * x4;
cd[3,1]= -2 - 4 * x1;   cd[3,2]= 1 - 2 * x2;
cd[3,3]= -2 * x3;       cd[3,4]= 1;
```

JACOBIAN Statement

JACOBIAN *variables* **;**

The JACOBIAN statement defines the JACOBIAN matrix J for a system of objective functions. For more information, see the section "Derivatives" on page 426.

The JACOBIAN statement lists $m * n$ variable names that correspond to the elements $J_{i,j}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, of the Jacobian matrix listed by rows.

For example, the statements

```
lsq f1-f3;
decvar x1 x2;
jacobian j1-j6;
```

correspond to the Jacobian matrix

$$J = \begin{bmatrix} J1 & J2 \\ J3 & J4 \\ J5 & J6 \end{bmatrix} = \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \\ \partial f_3/\partial x_1 & \partial f_3/\partial x_2 \end{bmatrix} .$$

The m rows of the Jacobian matrix must correspond to the order of the m function names listed in the MIN, MAX, or LSQ statement. The n columns of the Jacobian matrix must correspond to the order of the n decision variables listed in the DECVAR statement. To specify the values of nonzero derivatives, the variables specified in the JACOBIAN statement have to be defined in the left-hand side of algebraic expressions in programming statements.

For example, consider the Rosenbrock Function:

```
proc nlp tech=levmar;
   array j[2,2] j1-j4;
   lsq f1 f2;
   decvar x1 x2;
   jacobian j1-j4;

   f1 = 10 * (x2 - x1 * x1);
   f2 = 1 - x1;

   j[1,1] = -20 * x1;
   j[1,2] = 10;
```

```
   j[2,1] = -1;
   j[2,2] = 0;   /* is not needed */
run;
```

The JACOBIAN statement is useful only if more than one objective function is given in the MIN, MAX, or LSQ statement, or if a DATA= input data set specifies more than one function. If the MIN, MAX, or LSQ statement contains only one objective function and no DATA= input data set is used, the JACOBIAN and GRADIENT statements are equivalent. In the case of least-squares minimization, the crossproduct Jacobian is used as an approximative Hessian matrix.

LABEL Statement

LABEL*variable='label' [,variable='label'...]* **;**

The LABEL statement can be used to assign labels (up to 40 chararcters) to the decision variables listed in the DECVAR statement. The INVAR= data set can also be used to assign labels. The labels are attached to the output and are used in an OUTVAR= data set.

LINCON Statement

LINCON*l_con [, l_con ...]* **;**

where *l_con* := *linear_term operator* number
or *l_con* := number *operator linear_term*
for *linear_term* := < +|− > < number * > parameter < +|− < number * > variable ... >
operator := <= | < | >= | > | =

The LINCON statement specifies equality or inequality constraints

$$\sum_{j=1}^{n} a_{ij}x_j\{\leq \mid = \mid \geq\}b_i \text{ for } i = 1, \ldots, m$$

separated by commas. For example, the constraint $4x_1 - 3x_2 = 0$ is expressed as

```
decvar x1 x2;
lincon 4 * x1 - 3 * x2 = 0;
```

and the constraints

$$10x_1 - x_2 \geq 10$$
$$x_1 + 5x_2 \geq 15$$

are programmed as

```
decvar x1 x2;
lincon 10 <= 10 * x1 - x2,
       x1 + 5 * x2 >= 15;
```

MATRIX Statement

MATRIX *M_name pattern_definitions* **;**

The MATRIX statement defines a matrix H and the vector g, which can be given in the MINQUAD or MAXQUAD statement. The matrix H and vector g are initialized to zero, so that only the nonzero elements are given. The five different forms of the MATRIX statement are illustrated with the following example.

$$H = \begin{bmatrix} 100 & 10 & 1 & 0 \\ 10 & 100 & 10 & 1 \\ 1 & 10 & 100 & 10 \\ 0 & 1 & 10 & 100 \end{bmatrix} \quad g = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad c = 0.$$

Each MATRIX statement first names the matrix or vector and then lists its elements. If more than one MATRIX statement is given for the same matrix, then later definitions override the earlier ones.

The rows and columns in matrix H and vector g correspond to the order of decision variables in the DECVAR statement.

- **Full Matrix Definition**: The MATRIX statement consists of *H_name* or *g_name* followed by an equal sign and all (nonredundant) numerical values of the matrix H or vector g. Assuming symmetry, only the elements of the lower triangular part of the matrix H must be listed. This specification should be used mainly for small problems with almost dense H matrices.

```
MATRIX H=  100
           10 100
            1  10 100
            0   1  10 100;
MATRIX G= 1  2  3  4;
```

- **Band-diagonal Matrix Definition**: This form of *pattern definition* is useful if the H matrix has (almost) constant band-diagonal structure. The MATRIX statement consists of *H_name* followed by empty brackets $[,]$, an equal sign, and a list of $k+1$ numbers are assigned to the diagonal elements and the following k numbers are assigned to the adjacent k subdiagonals.

```
MATRIX H[,]=  100 10 1;
MATRIX G= 1  2  3  4;
```

- **Sparse Matrix Definitions**: In each of the following three specification types, the *H_name* or *g_name* is followed by a list of *pattern definitions* separated by commas. Each *pattern definition* consists of a location specification in brackets on the left side of an equal sign that is followed by a list of $k+1$ numbers.
  - **(Sub)Diagonalwise**: This form of *pattern definition* is useful if the H matrix contains nonzero elements along diagonals or subdiagonals. The starting location is specified by an index pair in brackets $[i, j]$. The $k+1$ numbers at the right-hand side are assigned to the elements $[i, j], \ldots, [i+$

$k, j+k]$ in a diagonal direction of the H matrix. The special case $k = 0$ can be used to assign values to single nonzero element locations in H.

```
MATRIX H [1,1]= 4 * 100,
         [2,1]= 3 *  10,
         [3,1]= 2 *   1;
MATRIX G [1,1]= 1  2  3  4;
```

– **Columnwise Starting in Diagonal**: This form of *pattern definition* is useful if the H matrix contains nonzero elements columnwise starting in the diagonal. The starting location is specified by only one index j in brackets $[, j]$. The $k + 1$ numbers at the right-hand side are assigned to the elements $[j, j], \ldots, [min(j + k, n), j]$.

```
MATRIX H [,1]= 100  10   1,
         [,2]= 100  10   1,
         [,3]= 100  10,
         [,4]= 100;
MATRIX G [,1]= 1  2  3  4;
```

– **Rowwise Starting in First Column**: This form of *pattern definition* is useful if the H matrix contains nonzero elements rowwise ending in the diagonal. The starting location is specified by only one index i in brackets $[i,]$. The $k+1$ numbers at the right-hand side are assigned to the elements $[i, 1], \ldots, [i, min(k + 1, i)]$.

```
MATRIX H [1,]= 100,
         [2,]=  10 100,
         [3,]=   1  10 100,
         [4,]=   0   1  10 100;
MATRIX G [1,]= 1  2  3  4;
```

MIN, MAX, and LSQ Statement

MIN *variables* **;**
MAX *variables* **;**

LSQ *variables* **;**

The MIN, MAX, or LSQ statement specifies the objective functions. Only one of the three statements can be used at a time and at least one must be given. The MIN and LSQ statements are for minimizing the objective function, and the MAX statement is for maximizing the objective function. The MIN, MAX, or LSQ statement lists one or more variables naming the objective functions f_i, $i = 1, \ldots, m$, (later defined by SAS program code).

- If the MIN or MAX statement lists m function names $f_1, \ldots, f_m$, the objective function f is

$$f(x) = \sum_{i=1}^{m} f_i$$

- If the LSQ statement lists m function names $f_1, \ldots, f_m$, the objective function f is

$$f(x) = \frac{1}{2} \sum_{i=1}^{m} f_i^2(x)$$

Note that the LSQ statement can be used only if TECH=LEVMAR or TECH=HYQUAN.

MINQUAD and MAXQUAD Statements

MINQUAD *H_name [, g_name [, c_number]]* **;**
MAXQUAD *H_name [, g_name [, c_number]]* **;**

The MINQUAD and MAXQUAD statements specify the H, g, and c, matrices that define a quadratic objective function. The MINQUAD statement is for minimizing the objective and the MAXQUAD statement is for maximizing the objective function.

The rows and columns in H and g correspond to the order of decision variables given in the DECVAR statement. Specifying the objective function with a MINQUAD or MAXQUAD statement indirectly defines the analytic derivatives for the objective function. Therefore, statements specifying derivatives are not valid in these cases. Also, only use these statements when TECH=LINCOMP or TECH=QUADAS and no nonlinear constraints are imposed.

There are three ways of using the MINQUAD or MAXQUAD statement:

- **Using ARRAY Statements**:
  The names *H_name* and *g_name* specified in the MINQUAD or MAXQUAD statement can be used in ARRAY statements. This specification is mainly for small problems with almost dense H matrices.

```
proc nlp pall;
   array h[2,2] .4 0
                0 4;
   minquad h, -100;
   decvar x1 x2 = -1;
   bounds  2 <= x1 <= 50,
         -50 <= x2 <= 50;
   lincon 10 <= 10 * x1 - x2;
run;
```

- **Using Elementwise Setting**:
  The names *H_name* and *g_name* specified in the MINQUAD or MAXQUAD statement can be followed directly by one-dimensional indices specifying the corresponding elements of matrix H and vector g. These element names can be used at the left side of numerical assignments. The one-dimensional index value l following *H_name* which corresponds to the element H_{ij} is computed by $l = (i-1)n + j, i \geq j$. The matrix H and vector g are initialized to zero,

so that only the nonzero elements need be given. This specification is efficient for small problems with sparse H matrices.

```
proc nlp pall;
   minquad h, -100;
   decvar x1 x2;
   bounds  2 <= x1 <= 50,
         -50 <= x2 <= 50;
   lincon 10 <= 10 * x1 - x2;
   h1 = .4; h4 = 4;
run;
```

- **Using MATRIX Statements**:
  The names *H_name* and *g_name* specified in the MINQUAD or MAXQUAD statement can be used in MATRIX statements. There are different ways to specify the nonzero elements of matrix H and vector g by MATRIX statements. The following example illustrates one way to use the MATRIX statement.

```
proc nlp all;
   matrix h[1,1] = .4 4;
   minquad h, -100;
   decvar x1 x2 = -1;
   bounds 2 <= x1 <= 50;
        -50 <= x2 <= 50;
   lincon 10 <= 10 * x1 - x2;
run;
```

NLINCON Statement

NLINCON *nlcon [, nlcon ...] [/ option]* **;**
NLC *nlcon [, nlcon ...] [/ option]* **;**

where *nlcon :=* number *operator* variable_list *operator* number
or *nlcon := -number operator variable_list*
or *nlcon := variable_list operator number*
and *operator :=* $<= \mid < \mid >= \mid > \mid =$
and *option := SUMOBS | EVERYOBS*

General nonlinear equality and inequality constraints are specified with the NLINCON statement. The syntax of the NLINCON statement is similar to that of the BOUNDS statement with two small additions:

- The BOUNDS statement can contain only the names of decision variables. The NLINCON statement can also contain the names of continuous functions of the decision variables. These functions must be computed in the program statements, and since they can depend on the values of some of the variables in the DATA= data set there are two possibilities:

  - If the continuous functions should be summed across all observations read from the DATA= data set, the NLINCON statement must be terminated by the / SUMOBS option.
  - If the continuous functions should be evaluated separately for each observation in the data set, the NLINCON statement must be terminated by the / EVERYOBS option. One constraint is generated for each observation in the data set.

- If the continuous function has to be evaluated only once for the entire data set, the NLINCON statement has the same form as the BOUNDS statement. If this constraint does depend on the values of variables in the DATA= data set, it is evaluated using the data of the first observation.

One- or two-sided constraints can be specified in the NLINCON statement. However, equality constraints must be one-sided. The pairs of operators (<,<=) and (>,>=) are treated in the same way.

These three statements require the values of the three functions v_1, v_2, v_3 to be between zero and ten, and they are equivalent:

```
nlincon 0 <= v1 - v3,
        v1 - v3 <= 10;

nlincon 0 <= v1 - v3 <= 10;

nlincon 10 >= v1 - v3 >= 0;
```

Also, consider the Rosen-Suzuki problem. It has three nonlinear inequality constraints:

$$8 - x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 \geq 0$$

$$10 - x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 \geq 0$$

$$5 - 2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 \geq 0$$

These are specified as:

```
nlincon c1 - c3 >= 0;

c1 = 8 - x1 * x1 - x2 * x2 - x3 * x3 - x4 * x4 -
         x1 + x2 - x3 + x4;
c2 = 10 - x1 * x1 - 2 * x2 * x2 - x3 * x3 - 2 * x4 * x4 +
         x1 + x4;
c3 = 5 - 2 * x1 * x1 - x2 * x2 - x3 * x3 - 2 * x1 + x2 + x4;
```

Note: QUANEW and NMSIMP are the only optimization subroutines that support the NLINCON statement.

DECVAR Statement

DECVAR *name_list [=numbers] [, name_list [=numbers] ...]* **;**
VAR *name_list [=numbers] [, name_list [=numbers] ...]* **;**

PARMS *name_list [=numbers] [, name_list [=numbers] ...]* **;**

PARAMETERS *name_list [=numbers] [, name_list [=numbers] ...]* **;**

The DECVAR statement lists the names of the $n > 0$ decision variables and specifies grid search and initial values for an iterative optimization process. The decision variables listed in the DECVAR statement cannot also be used in the MIN, MAX, MINQUAD, MAXQUAD, LSQ, GRADIENT, HESSIAN, JACOBIAN, CRPJAC, or NLINCON statement.

The DECVAR statement contains a list of decision variable names (not separated by commas) optionally followed by an equal sign and a list of numbers. If the number list consists of only one number, this number defines the initial value for all the decision variables listed to the left of the equal sign.

If the number list consists of more than one number, these numbers specify the grid locations for each of the decision variables listed left of the equal sign. You can use the TO and BY keywords to specify a number list for a grid search can be used. When a grid of points is specified with a DECVAR statement, PROC NLP computes the objective function value at each grid point and chooses the best (feasible) grid point as a starting point for the optimization process. The use of the BEST= option is recommended to save computing time and memory for the storing and sorting of all grid point information. Usually only feasible grid points are included in the grid search. If the specified grid contains points located outside the feasible region and you are interested in the function values at those points, it is possible to use the INFEASIBLE option to compute (and display) their function values as well.

PROFILE Statement

PROFILE *parms [/ [ALPHA= values] [options]]* **;**

where *parms := pnam_1 pnam_2 ... pnam_n*
values := list of alpha values in (0,1)
options := additional options

The PROFILE statement

- writes the (x, y) coordinates of profile points for each of the listed parameters to the OUTEST= data set
- displays, or writes to the OUTEST= data set, the profile likelihood confidence limits (PL CL) for the listed parameters for the specified α values. If the ap-

proximate standard errors are available, the corresponding Wald confidence limits can be computed.

When computing the profile points or likelihood profile confidence intervals, PROC NLP assumes that a maximization of the log likelihood function is desired. Each point of the profile and each endpoint of the confidence interval is computed by solving corresponding nonlinear optimization problems.

The keyword PROFILE must be followed by the names of parameters for which the profile or the PL CLs should be computed. If the parameter name list is empty, the profiles and PL CLs for all parameters are computed. Then, optionally, the alpha values follow. The list of α values may contain TO and BY keywords. Each element must satisfy $0 < \alpha < 1$. The following is an example:

```
profile l11-l15 u1-u5 c /
        alpha= .9 to .1 by -.1 .09 to .01 by -.01;
```

Duplicate α values or values outside $(0, 1)$ are automatically eliminated from the list.

A number of additional options can be specified.

FFACTOR=*r* specifies the factor relating the discrepancy function $f(\theta)$ to the χ^2 quantile. The default value is $r = 2$.

FORCHI= F | CHI : defines the scale for the y values written to the OUTEST= data set. For FORCHI=F, the y values are scaled to the values of the log likelihood function $f = f(\theta)$; for FORCHI=CHI, the y values are scaled so that $\hat{y} = \chi^2$. The default value is FORCHI=F.

FEASRATIO=*r* specifies a factor of the Wald confidence limit (or an approximation of it if standard errors are not computed) defining an upper bound for the search for confidence limits. In general, the range of x values in the profile graph is between $r = 1$ and $r = 2$ of the size of the corresponding Wald interval. For many examples, the χ^2 quantiles corresponding to small α values define a y level $\hat{y} - \frac{1}{2}q_1(1 - \alpha)$, which is too far away from $\hat{y}$ to be reached by $y(x)$ for x within the range of twice the Wald confidence limit. The search for an intersection with such a y level at a practically infinite value of x can be computationally expensive. A smaller value for r can speed up computation time by restricting the search for confidence limits to a region closer to $\hat{x}$. The default value of $r = 1000$ practically disables the FEASRATIO= option.

OUTTABLE specifies that the complete set θ of parameter estimates rather than only $x = \theta_j$ for each confidence limit is written to the OUTEST= data set. This output can be helpful for further analyses on how small changes in $x = \theta_j$ affect the changes in the $\theta_i, i \neq j$.

For some applications, it may be computationally less expensive to compute the PL confidence limits for a few parameters than to compute the approximate covariance matrix of many parameters, whichis the basis for the Wald confidence limits. However, the computation of the profile of the discrepancy function and the corresponding CLs in general will be much more time consuming than that of the Wald CLs.

Program Statements

This section lists the program statements used to code the objective function and nonlinear constraints and their derivatives, and it documents the differences between program statements in the NLP procedure and program statements in the DATA step. The syntax of program statements used in PROC NLP is identical to those used in the CALIS, GENMOD, and MODEL procedures (refer to the *SAS ETS User's Guide*).

Most of the program statements which can be used in the SAS DATA step can also be used in the NLP procedure. See the *SAS Language Guide* or the SAS Base documentation for a description of the SAS program statements.

```
ABORT;
CALL name [ ( expression [, expression ... ] ) ];
DELETE;
DO [ variable = expression
              [ TO expression ] [ BY expression ]
              [, expression [ TO expression ] [ BY expression ] ... ]
   ]
   [ WHILE expression ] [ UNTIL expression ];
END;
GOTO statement_label;
IF expression;
IF expression THEN program_statement;
   ELSE program_statement;
variable = expression;
variable + expression;
LINK statement_label;
PUT [ variable] [=] [...] ;
RETURN;
SELECT [( expression )];
STOP;
SUBSTR( variable, index, length ) = expression;
WHEN ( expression) program_statement;
   OTHERWISE program_statement;
```

For the most part, the SAS program statements work as they do in the SAS DATA step as documented in the *SAS Language Guide*. However, there are several differences that should be noted.

- 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; however,

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

  is not.
- The PUT statement, used mostly for program debugging in PROC NLP, supports only some of the features of the DATA step PUT statement, and has some new features that the DATA step PUT statement does not:
  - The PROC NLP PUT statement does not support line pointers, factored lists, iteration factors, overprinting, _INFILE_, the colon (:) format modifier, or "$".
  - The PROC NLP PUT statement does support expressions, but the expression must be enclosed inside of parentheses. For example, the following statement diplays the square root of x `put (sqrt(x));`
  - The PROC NLP PUT statement supports the print item _PDV_ to print a formatted listing of all variables in the program. For example, the following statement displays a more readable listing of the variables than the _all_ print item `put _pdv_;`
- The WHEN and OTHERWISE statements allow 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;
```

It is recommended to keep some kind of order in the input of NLP, that is, between the statements that define decision variables and constraints and the program code used to specify objective functions and derivatives.

Use of Special Variables in Program Code

Except for the quadratic programming techniques (QUADAS and LICOMP) that do not execute program statements during the iteration process, several special variables in the program code can be used to communicate with PROC NLP in special situations:

- **_OBS_** If a DATA= input data sets used, it is possible to access a variable _OBS_ which contains the number of the observation processed from the data set. You should not change the content of the _OBS_ variable. This variable enables you to modify the programming statements depending on the

observation number processed in the DATA= input data set. For example, to set variable A to 1 when observation 10 is processed, and otherwise to 2, it is possible to specify

```
IF _OBS_ = 10 THEN A=1; ELSE A=2;
```

- **_ITER_** This variable is set by PROC NLP, and it contains the number of the current iteration of the optimization technique as it is displayed in the optimization history. You should not change the content of the _ITER_ variable. It is possible to read the value of this variable in order to modify the programming statements depending on the iteration number processed. For example, to display the content of the variables A, B, and C when there are more than 100 iterations processed, it is possible to use

```
IF _ITER_ > 100 THEN PUT A B C;
```

- **_DPROC_** This variable is set by PROC NLP to indicate whether the code is called only to obtain the values of the m objective functions f_i (_DPROC_=0) or whether specified derivatives (defined by the GRADIENT, JACOBIAN, CRPJAC, or HESSIAN statement) also have to be computed (_DPROC_=1). Checking the _DPROC_ variable makes it possible to save computer time by not performing derivative code that is not needed by the current call. In particular, when a DATA= input data set is used, the code is processed many times to compute only the function values. If the programming statements in the program contain the specification of computationally expensive first- and second-order derivatives, you can put the derivative code in an IF statement that is processed only if _DPROC_ is not zero. You should not change the content of the _DPROC_ variable.
- **_INDF_** The _INDF_ variable is set by PROC NLP to inform you of the source of calls to the function or derivative programming.

  INDF=0 indicates the first function call in a grid search. This is also the first call evaluating the programming statements if there is a grid search defined by grid values in the PARMS or VAR statement.

  INDF=1 indicates further function calls in a grid search.

  INDF=2 indicates the call for the feasible starting point. This is also the first call evaluating the programming statements if there is no grid search defined.

  INDF=3 indicates calls from a gradient checking algorithm.

  INDF=4 indicates calls from the minimization algorithm. The _ITER_ variable contains the iteration number.

  INDF=5 if the active set algorithm leaves the feasible region (due to rounding errors), an algorithm tries to return it into the feasible region; _INDF_=5 indicates a call that is done when such a step is successful.

  INDF=6 indicates calls from a factorial test subroutine that tests the neighborhood of a point x for optimality.

INDF=7,8 indicates calls from subroutines needed to compute finite difference derivatives using only values of the objective function. No nonlinear constraints are evaluated.

INDF=9 indicates calls from subroutines needed to compute second-order finite difference derivatives using analytic (specified) first-order derivatives. No nonlinear constraints are evaluated.

INDF=10 indicates calls where only the nonlinear constraints but no objective function are needed. The analytic derivatives of the nonlinear constraints are computed.

INDF=11 indicates calls where only the nonlinear constraints but no objective function are needed. The analytic derivatives of the nonlinear constraints are not computed.

INDF=-1 indicates the last call at the final solution.

You should not change the content of the _INDF_ variable.

- **_LIST_**: You can set the _LIST_ variable to control the output during the iteration process:

  LIST=0 is equivalent to the NOPRINT option. It suppresses all output.

  LIST=1 is equivalent to the PSUM but not the PHIST option. The optimization start and termination messages are displayed. However, the PSUM option suppresses the output of the iteration history.

  LIST=2 is equivalent to the PSHORT option or to a combination of the PSUM and PHIST options. The optimization start information, the iteration history, and termination message are displayed.

  LIST=3 Equivalent to: Not PSUM, not PSHORT, and not PALL: The optimization start information, the iteration history, and the termination message are displayed.

  LIST=4 is equivalent to the PALL option. The extended optimization start information (also containing settings of termination criteria and other control parameters) is displayed.

  LIST=5 In addition to the iteration history, the vector $x^{(k)}$ of parameter estimates is displayed for each iteration k.

  LIST=6 In addition to the iteration history, the vector $x^{(k)}$ of parameter estimates and the gradient $g^{(k)}$ (if available) of the objective function are displayed for each iteration k.

  It is possible to set the _LIST_ variable in the program code to obtain more or less output in each iteration of the optimization process. For example,

```
IF _ITER_ = 11      THEN _LIST_=5;
ELSE IF _ITER_ > 11 THEN _LIST_=1;
                    ELSE _LIST_=3;
```

- **_TOOBIG_** The value of _TOOBIG_ is initialized to zero by PROC NLP, but you can set it to one during the iteration, indicating problems evaluating the program statements. The objective function and derivatives must be computable at the starting point. However, during the iteration it is possible to

set the _TOOBIG_ variable to 1, indicating that the programming statements (computing the value of the objective function or the specified derivatives) cannot be performed for the current value of x_k. Some of the optimization techniques check the value of _TOOBIG_ and try to modify the parameter estimates so that the objective function (or derivatives) can be computed in a following trial.

- **_NOBS_** The value of the _NOBS_ variable is initialized by PROC NLP to the product of the number of functions $mfun$ specified in the MIN, MAX or LSQ statement and the number of valid observations $nobs$ in the current BY group of the DATA= input data set. The value of the _NOBS_ variable is used for computing the scalar factor of the covariance matrix (see options COV=, VARDEF=, nmd SIGSQ= in the section "PROC NLP Statement" on page 386). If you reset the value of the _NOBS_ variable, the value that is available at the end of the iteration is used by PROC NLP to compute the scalar factor of the covariance matrix.
- **_DF_** The value of the _DF_ variable is initialized by PROC NLP to the number n of parameters specified in the DECVAR statement. The value of the _DF_ variable is used for computing the scalar factor d of the covariance matrix. See the options COV=, VARDEF=, and SIGSQ= in the section "PROC NLP Statement" on page 386. If you reset the value of the _DF_ variable, the value that is available at the end of the iteration is used by PROC NLP to compute the scalar factor of the covariance matrix.
- **_LASTF_** In each iteration (except the first one), the value of the _LASTF_ variable is set by PROC NLP to the final value of the objective function that was achieved during the last iteration. This value should agree with the value that is displayed in the iteration history and that is written in the OUTEST= data set when the OUTITER option is specified.

Details

Criteria for Optimality

PROC NLP solves

$$\begin{array}{ll} & \min f(x), \quad x \in \mathcal{R}^n \\ \text{s.t.} & c_i(x) = 0, \quad i = 1, \ldots, m_e \\ & c_i(x) \geq 0, \quad i = m_e + 1, \ldots, m \end{array}$$

where f is the objective function and the m c_i's are the constraint functions.

A point x is feasible if it satisfies all the constraints. The feasible region $\mathcal{G}$ is the set of all the feasible points. x^* is a global solution of the preceeding problem if no point in $\mathcal{G}$ has a lower function value than f(x^*). x^* is a local solution of the problem if there exists some open neighborhood surrounding x^* in that no point has a lower function value than f(x^*). Nonlinear Programming algorithms cannot consistently find global minima. All the algorithms in PROC NLP find a local minimum for this problem.

If you need to check whether the obtained solution is a global minimum, you may have to run PROC NLP with different starting points obtained either at random or by selecting a point on a grid that contains $\mathcal{G}$.

The local minimizer x^* of this problem satisfies the following local optimality conditions:

- The grdient (vector of first derivatives) $g(x^*) = \nabla f(x^*)$ of the objective function f (projected toward the feasible region if the problem is constrained) at the point x^* is zero.
- The Hessian (matrix of second derivatives) $G(x^*) = \nabla^2 f(x^*)$ of the objective function f (projected toward the feasible region $\mathcal{G}$ in the constrained case) at the point x^* is positive definite.

Most of the optimization algorithms in PROC NLP use iterative techniques that result in a sequence of points $x^0, ..., x^n, ...$, that converges to a local solution x^*. At the solution, PROC NLP performs tests to confirm that the (projected) gradient is close to zero and that the (projected) Hessian matrix is positive definite.

Karush-Kuhn-Tucker Conditions

An important tool in the analysis and design of algorithms in constrained optimization is the *Lagrangian Function,* that is a linear combination of the objective function and the constraints:

$$L(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x)$$

The coefficients λ_i are called *Lagrange multipliers*. This tool makes it possible to state necessary and sufficient conditions for a local minimum. The various algorithms in PROC NLP create sequences of points, each of that is closer than the previous one to satisfying these conditions.

Assuming that the functions f and c_i are twice continuously differentiable, the point x^* is a *local minimum* of the nonlinear programming problem, if there exists a vector $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ that meets the following conditions.

1. first-order, Karush-Kuhn-Tucker conditions:

$$\begin{array}{lll} c_i(x^*) = 0, & & i = 1, \ldots, m_e, \\ c_i(x^*) \geq 0, & \lambda_i^* \geq 0, \quad \lambda_i^* c_i(x^*) = 0, & i = m_e + 1, \ldots, m \\ \nabla_x L(x^*, \lambda^*) = 0 & & \end{array}$$

2. Second-order conditions:

Each nonzero vector $y \in \mathcal{R}^n$ for which

$$y^T \nabla_x c_i(x^*) = 0 \begin{cases} i = 1, \ldots, m_e, \\ \forall i \in \{m_e + 1, \ldots, m : \lambda_i^* > 0\} \end{cases}$$

satisfies

$$y^T \nabla_x^2 L(x^*, \lambda^*) y > 0$$

Most of the algorithms to solve this problem attempt to find a combination of vectors x and λ for which the gradient of the Langrangian function in respect to x is zero.

Derivatives

The first and second order conditions of optimality are based on first and second derivates of the object function f and the constraints c_i.

The gradient vector contains the first derivatives of the objective function f with respect to the parameters $x_1, \ldots, x_n$, as follows:

$$g(x) = \nabla f(x) = (\frac{\partial f}{\partial x_j})$$

The $n \times n$ symmetric Hessian matrix contains the second derivatives of the objective function f with respect to the parameters $x_1, \ldots, x_n$, as follows:

$$G(x) = \nabla^2 f(x) = (\frac{\partial^2 f}{\partial x_j \partial x_k}).$$

For Least-Squares problems, the $m \times n$ Jacobian matrix contains the first-order derivatives of m objective functions $f_i(x)$ with respect to the parameters $x_1, \ldots, x_n$, as follows:

$$J(x) = (\nabla f_1, \ldots, \nabla f_m) = (\frac{\partial f_i}{\partial x_j})$$

In case of Least-Squares problems, the crossproduct Jacobian $J^T J$,

$$J^T J = (\sum_{i=1}^{m} \frac{\partial f_i}{\partial x_j} \frac{\partial f_i}{\partial x_k})$$

is used as an approximate Hessian matrix. It is a very good approximation of the Hessian if the residuals at the solution are "small." (If the residuals are not sufficiently small at the solution, this approach may result in slow convergence.) The fact that it is possible to obtain Hessian approximations for this problem that do not require any computation of second derivatives means that Least-Squares algorithms are more efficient than unconstrained optimization algorithms. Using the vector $f = f(x)$ of

function values, $f(x) = (f_1(x), \ldots, f_m(x))^T$, PROC NLP computes the gradient $g = g(x)$ by

$$g(x) = J^T(x)f(x)$$

The $mc \times n$ Jacobian matrix contains the first-order derivatives of mc nonlinear constraint functions $c_i(x)$, $i = 1, \ldots, mc$, with respect to the parameters $x_1, \ldots, x_n$, as follows:

$$CJ(x) = (\nabla c_1, \ldots, \nabla c_{mc}) = (\frac{\partial c_i}{\partial x_j})$$

PROC NLP provides three ways to compute derivatives:

- It computes analytical first- and second-order derivatives of the objective function f with respect to the n variables x_j.
- It computes first- and second-order finite difference approximations to the derivatives. For more information, see the section "Finite-Difference Approximations of Derivatives" on page 439.
- The user supplies formulas for analytical or numerical first- and second-order derivatives of the objective function in the GRADIENT, JACOBIAN, CRPJAC, and HESSIAN statements. The JACNLC statement can be used to specify the derivatives for the nonlinear constraints.

Optimization Algorithms

There are three groups of optimization techniques available in PROC NLP. A particular optimizer can be selected with the TECH=$name$ option in the PROC NLP statement.

Algorithm	TECH=
Linear Complementary Problem	LICOMP
Quadratic Active Set Technique	QUADAS
Trust-Region Method	TRUREG
Newton-Raphson Method With Line-Search	NEWRAP
Newton-Raphson Method With Ridging	NRRIDG
Quasi-Newton Methods (DBFGS, DDFP, BFGS, DFP)	QUANEW
Double-Dogleg Method (DBFGS, DDFP)	DBLDOG
Conjugate Gradient Methods (PB, FR, PR, CD)	CONGRA
Nelder-Mead Simplex Method	NMSIMP
Levenberg-Marquardt Method	LEVMAR
Hybrid Quasi-Newton Methods (DBFGS, DDFP)	HYQUAN

Since no single optimization technique is invariably superior to others, PROC NLP provides a variety of optimization techniques that work well in various circumstances. However, it possible to devise problems for which none of the techniques in PROC

NLP can find the correct solution. Moreover, nonlinear optimization can be computationally expensive in terms of time and memory, so care must be taken when matching an algorithm to a problem.

All optimization techniques in PROC NLP use $O(n^2)$ memory except the conjugate gradient methods, which use only $O(n)$ memory and are designed to optimize problems with many variables. Since the techniques are iterative, they require the repeated computation of

- the function value (optimization criterion)
- the gradient vector (first-order partial derivatives)
- for some techniques, the (approximate) Hessian matrix (second-order partial derivatives)
- values of linear and nonlinear constraints
- the first-order partial derivatives (Jacobian) of nonlinear constraints

However, since each of the optimizers requires different derivatives and supports different types of constraints, some computational efficiencies can be gained. The following table shows, for each optimization technique, which derivatives are needed (FOD: first-order derivatives; SOD: second-order derivatives) and what kind of constraints (BC: boundary constraints; LIC: linear constraints; NLC: nonlinear constraints) are supported.

Algorithm	FOD	SOD	BC	LIC	NLC
LICOMP	-	-	x	x	-
QUADAS	-	-	x	x	-
TRUREG	x	x	x	x	-
NEWRAP	x	x	x	x	-
NRRIDG	x	x	x	x	-
QUANEW	x	-	x	x	x
DBLDOG	x	-	x	x	-
CONGRA	x	-	x	x	-
NMSIMP	-	-	x	x	x
LEVMAR	x	-	x	x	-
HYQUAN	x	-	x	x	-

Preparation for Using Optimization Algorithms

It is rare that a problem is submitted to an optimization algorithm "as is." By making a few changes in your problem, you can reduce its complexity, that would increase the chance of convergence and save execution time.

- Whenever possible, use linear functions instead of nonlinear functions. PROC NLP will reward you with faster and more accurate solutions.
- Most optimization algorithms are based on quadratic approximations to nonlinear functions. You should try to avoid the use of funcions that cannot be properly approximated by quadratic functions. Try to avoid the use of rational functions.

For example,the constraint

$$\frac{sin(x)}{x+1} > 0$$

should be replaced by the equivalent constraint

$$sin(x)(x+1) > 0$$

and the constraint

$$\frac{sin(x)}{x+1} = 1$$

should be replaced by the equivalent constraint

$$sin(x) - (x+1) = 0$$

- Try to avoid the use of exponential functions, if possible.
- If you can reduce the complexity of your function by the addition of a small number of variables, that may help the algorithm avoid stationary points.
- Provide the best starting point you can. A good starting point leads to better quadratic approximations and faster convergence.

Choosing an Optimization Algorithm

The factors that go into choosing a particular optimizer for a particular problem are complex and may involve trial and error. The following should be taken into account: First, the structure of the problem has to be considered: Is it quadratic? least-squares? Does it have linear or nonlinear constraints? Next, it is important to consider the type of derivatives of the objective function and the constraints that are needed and whether these are analytically tractable or not. This section provides some guidelines for making the right choices.

For many optimization problems, computing the gradient takes more computer time than computing the function value, and computing the Hessian sometimes takes *much* more computer time and memory than computing the gradient, especially when there are many decision variables. Optimization techniques that do not use the Hessian usually require more iterations than techniques that do use Hessian approximations (such as finite differences or BFGS update) and so are often slower. Techniques that do not use Hessians at all tend to be slow and less reliable.

The derivative compiler is not efficient in the computation second-order derivatives. For large problems, memory and computer time can be saved by programming your own derivatives using the GRADIENT, JACOBIAN, CRPJAC, HESSIAN, and JAC-NLC statements. If you are not able or willing to specify first- and second-order derivatives of the objective function, you can rely on finite difference gradients and Hessian update formulas. This combination is frequently used and works very well for small and medium size problems. For large size problems, you are advised not to use an optimization technique that requires the computation of second derivatives.

The following provides some guidance for matching an algorithm to a particular problem.

- Quadratic Programming
  - **QUADAS**
  - **LINCOMP**
- General Nonlinear Optimization
  - Nonlinear Constraints
    - **Small Problems: NMSIMP**
      Not suitable for highly nonlinear problems or for problems with $n > 20$.
    - **Medium Problems: QUANEW**
  - Only Linear Constraints
    - **Small Problems: TRUREG (NEWRAP, NRRIDG)**
      ($n \leq 40$) where the Hessian matrix is not expensive to compute. Sometimes NRRIDG can be faster than TRUREG, but TRUREG can be more stable. NRRIDG needs only one matrix with $n(n+1)/2$ double words; TRUREG and NEWRAP need two such matrices.
    - **Medium Problems: QUANEW (DBLDOG)**
      ($n \leq 200$) where the objective function and the gradient are much faster to evaluate than the Hessian. QUANEW and DBLDOG in general need more iterations than TRUREG, NRRIDG, and NEWRAP, but each iteration can be much faster. QUANEW and DBLDOG need only the gradient to update an approximate Hessian. QUANEW and DBLDOG need slightly less memory than TRUREG or NEWRAP (essentially one matrix with $n(n+1)/2$ double words).
    - **Large Problems: CONGRA**
      ($n > 200$) where the objective function and the gradient can be computed much faster than the Hessian and where too much memory is needed to store the (approximate) Hessian. CONGRA in general needs more iterations than QUANEW or DBLDOG, but each iteration can be much faster. Since CONGRA needs only a factor of n double-word memory, many large applications of PROC NLP can be solved only by CONGRA.
    - **No Derivatives: NMSIMP**
      ($n \leq 20$) where derivatives are not continuous or are very difficult to compute.
- Least-Squares Minimization
  - **Small Problems: LEVMAR (HYQUAN)**
    ($n \leq 60$) where the crossproduct Jacobian matrix is easy and inexpensive to compute. In general, LEVMAR is more reliable, but there are problems with high residuals where HYQUAN can be faster than LEVMAR.

– **Medium Problems: QUANEW (DBLDOG)**
($n \leq 200$) where the objective function and the gradient are much faster to evaluate than the crossproduct Jacobian. QUANEW and DBLDOG in general need more iterations than LEVMAR or HYQUAN, but each iteration can be much faster.
– **Large Problems: CONGRA**

– **No Derivatives: NMSIMP**

Quadratic Programming Method

The QUADAS and LICOMP algorithms can be used to minimize or maximize a quadratic objective function,

$$f(x) = \frac{1}{2}x^T Gx + g^T x + c, \quad \text{with} \quad G^T = G$$

with linear or boundary constraints

$$Ax \geq b \quad \text{or} \quad l_j \leq x_j \leq u_j$$

where $x = (x_1, \ldots, x_n)^T$, $g = (g_1, \ldots, g_n)^T$, G is an $n \times n$ symmetric matrix, A is an $m \times n$ matrix of general linear constraints, and $b = (b_1, \ldots, b_m)^T$. The value of c modifies only the value of the objective function, not its derivatives, and the location of the optimizer x^* does not depend on the value of the constant term c. For QUADAS or LICOMP, the objective function must be specified using the MINQUAD or MAX QUAD statement or using an INQUAD= data set. In this case, derivatives do not need to be specified. because the gradient vector

$$\nabla f(x) = Gx + g$$

and the $n \times n$ Hessian matrix

$$\nabla^2 f(x) = G$$

are easily obtained from the data input.

Simple boundary and general linear constraints can be specified using the BOUNDS or LINCON statement or an INQUAD= or INEST= data set.

General Quadratic Programming (QUADAS)

The QUADAS algorithm is an active set method that iteratively updates the QT decomposition of the matrix A_k of active linear constraints and the Cholesky factor of the projected Hessian $Z^T GZ$ simultaneously. The update of active boundary and linear constraints is done separately; refer to Gill et al. (1984). Here Q is an $n_{free} \times n_{free}$ orthogonal matrix composed of vectors spanning the null space Z of A_k in its first $n_{free} - n_{alc}$ columns and range space Y in its last n_{alc} columns; T is an $n_{alc} \times n_{alc}$ triangular matrix of special form, $t_{ij} = 0$ for $i < n - j$, where n_{free} is the

number of free parameters (n minus the number of active boundary constraints), and n_{alc} is the number of active linear constraints. The Cholesky factor of the projected Hessian matrix $Z_k^T G Z_k$ and the QT decomposition are updated simultaneously when the active set changes.

Linear Complementarity (LICOMP)

The LICOMP technique solves a quadratic problem as a linear complementarity problem. It can be used only if G is positive (negative) semi-definite for minimization (maximization) and if the parameters are restricted to be positive.

This technique finds a point that meets the Karush-Kuhn-Tucker conditions by solving the linear complementary problem

$$w = Mz + q$$

with constraints

$$w^T z \geq 0, \quad w \geq 0, \quad z \geq 0$$

where

$$z = \begin{bmatrix} x \\ \lambda \end{bmatrix} \quad M = \begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \quad q = \begin{bmatrix} g \\ -b \end{bmatrix}$$

Only the LCEPSILON= option can be used to specify a tolerance used in computations.

General Nonlinear Optimization

Trust-Region Optimization (TRUREG)

The trust-region method uses the gradient $g(x^{(k)})$ and Hessian matrix $G(x^{(k)})$ and thus requires that the objective function $f(x)$ 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 using Dennis, Gay, & Welsch (1981), Gay (1983), and Moré & Sorensen (1983) .

The trust region method performs well for small to medium-sized problems and does not require many function, gradient, and Hessian calls. If the computation of the Hessian matrix is computationally expensive, use the UPDATE= option for update formulas (that gradually build the second-order information in the Hessian). For larger problems, the conjugate gradient algorithm may be more appropriate.

Newton-Raphson Optimization With Line-Search (NEWRAP)

The NEWRAP technique uses the gradient $g(x^{(k)})$ and Hessian matrix $G(x^{(k)})$ and thus 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 may 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 done 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 done 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 $g(x^{(k)})$ and Hessian matrix $G(x^{(k)})$ and thus 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. If this algorithm is used for least-squares problems, it performs a ridged Gauss-Newton minimization.

The NRRIDG method performs well for small to medium-sized problems and 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 may be more efficient.

Since NRRIDG uses an orthogonal decomposition of the approximate Hessian, each iteration of NRRIDG can be slower than that of NEWRAP, that works with Cholesky decomposition. However, usually NRRIDG needs fewer iterations than NEWRAP.

Quasi-Newton Optimization (QUANEW)

The (dual) quasi-Newton method uses the gradient $g(x^{(k)})$ and does not need to compute second-order derivatives since they are approximated. It works well for medium to moderately large optimization problems where the objective function and the gradient are much faster to compute than the Hessian, but in general it requires more iterations than the techniques TRUREG, NEWRAP, and NRRIDG, which compute second-order derivatives.

The QUANEW algorithm depends on whether or not there are nonlinear constraints.

Unconstrained or Linearly Constrained Problems

If there are no nonlinear constraints, QUANEW is either

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

depending upon the value of the UPDATE= options. For problems with general linear inequality constraints, the dual quasi-Newton methods can be more efficient than the original ones.

Four update formulas can be specified with the *UPDATE*= option:

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

In each iteration, a line-search is done 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 those cases, either the update is skipped or the iterations are restarted with an identity matrix resulting in the steepest descent or ascent search direction. Line-search algorithms other than the default one can be specified with the LIS= option.

Nonlinearly Constrained Problems

The algorithm used for nonlinearly constrained quasi-Newton optimization is an efficient modification of Powell's (1978, 1982) *Variable Metric Constrained WatchDog* (VMCWD) algorithm. A similar but older algorithm (VF02AD) is part of the Harwell library. Both VMCWD and VF02AD use Fletcher's VE02AD algorithm (part of the Harwell library) for positive definite quadratic programming. The PROC NLP QUANEW implementation uses a quadratic programming subroutine that updates and downdates the approximation of the Cholesky factor when the active set changes. The nonlinear QUANEW algorithm is not a feasible point algorithm, and the value of the objective function need not decrease (minimization) or increase (maximization) monotonically. Instead, the algorithm tries to reduce a linear combination of the objective function and constraint violations, called the *merit function*.

The following are similarities and differences between this algorithm and the VMCWD algorithm:

- A modification of this algorithm can be performed by specifying VERSION=1, that replaces the update of the Lagrange vector μ with the original update of Powell (1978) that is used in VF02AD. This can be helpful for some applications with linearly dependent active constraints.
- If the VERSION option is not specified or if VERSION=2 is specified, the evaluation of the Lagrange vector μ is performed in the same way as Powell (1982) describes.
- Instead of updating an approximate Hessian matrix, this algorithm uses the dual BFGS (or DFP) update that updates the Cholesky factor of an approximate

Hessian. If the condition of the updated matrix gets too bad, a restart is done with a positive diagonal matrix. At the end of the first iteration after each restart, the Cholesky factor is scaled.

- The Cholesky factor is loaded into the quadratic programming subroutine, automatically ensuring positive definiteness of the problem. During the quadratic programming step, the Cholesky factor of the projected Hessian matrix $Z_k^T G Z_k$ and the QT decomposition are updated simultaneously when the active set changes. Refer to Gill dt al. (1984) for more information.
- The line-search strategy is very similar to that of Powell (1982). However, this algorithm does not call for derivatives during the line-search, so the algorithm generally needs fewer derivative calls than function calls. VMCWD always requires the same number of derivative and function calls. Sometimes Powell's line-search method uses steps that are too long. In these cases, use the INSTEP= option to restrict the step length α.
- The watchdog strategy is similar to that of Powell (1982); however, it doesn't return automatically after a fixed number of iterations to a former better point. A return here is further delayed if the observed function reduction is close to the expected function reduction of the quadratic model.
- The Powell termination criterion still is used (as FTOL2) but the QUANEW implementation uses two additional termination criteria (GTOL and ABSGTOL).

The nonlinear QUANEW algorithm needs the Jacobian matrix of the first-order derivatives (constraints normals) of the constraints $CJ(x)$.

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

- UPDATE=DBFGS performs the dual BFGS update of the Cholesky factor of the Hessian matrix. This is the default.
- UPDATE=DDFP performs the dual DFP update of the Cholesky factor of the Hessian matrix.

This algorithm uses its own line-search technique. All options and parameters (except the INSTEP= option) controlling the line-search in the other algorithms do not apply here. In several applications, large steps in the first iterations were troublesome. You can use the INSTEP= option to impose an upper bound for the step size α during the first five iterations. You may also use the INHESSIAN[=r] option to specify a different starting approximation for the Hessian. Choosing simply the INHESSIAN option will use the Cholesky factor of a (possibly ridged) finite difference approximation of the Hessian to initialize the quasi-Newton update process. The values of the LCSINGULAR=, LCEPSILON=, and LCDEACT= options, which control the processing of linear and boundary constraints, are valid only for the quadratic programming subroutine used in each iteration of the nonlinear constraints QUANEW algorithm.

Double Dogleg Optimization (DBLDOG)

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

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

The step is requested to remain within a prespecified trust region radius, refer to Fletcher (1987, p. 107). Thus, the DBLDOG subroutine uses the dual quasi-Newton update but does not perform a line-search. Two update formulas can be specified with the UPDATE= option:

DBFGS performs the dual BFGS (Broyden, Fletcher, Goldfarb, & Shanno) update of the Cholesky factor of the Hessian matrix. This is the default.

DDFP performs the dual DFP (Davidon, Fletcher, & Powell) update of the Cholesky factor of the Hessian matrix.

The double dogleg optimization technique works well for medium 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 & Mei (1979) and Gay (1983) but is extended for dealing with boundary and linear constraints. DBLDOG generally needs more iterations than the techniques TRUREG, NEWRAP, or NRRIDG that need second-order derivatives, but each of the DBLDOG iterations is computationally cheap. Furthermore, DBLDOG needs only gradient calls for the update of the Cholesky factor of an approximate Hessian.

Conjugate Gradient Optimization (CONGRA)

Second-order derivatives are not used by CONGRA. The CONGRA algorithm can be expensive in function and gradient calls but needs only $O(n)$ memory for unconstrained optimization. In general, many iterations are needed to obtain a precise solution, but each of the CONGRA iterations is computationally cheap. Four different update formulas for generating the conjugate directions can be specified 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 value is UPDATE=PB, since it behaved best in most test examples. You are advised to avoid the option UPDATE=CD, that behaved worst in most test examples.

The CONGRA subroutine should be used for optimization problems with large n. For the unconstrained or boundary constrained case, CONGRA needs only $O(n)$ bytes of working memory, whereas all other optimization methods require order $O(n^2)$ bytes of working memory. During n successive iterations, uninterrupted by restarts or changes in the working set, the conjugate gradient algorithm computes a cycle of n conjugate search directions. In each iteration, a line-search is done 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 requires a large number of function evaluations. It is unlikely to give accurate results for $n \gg 40$.

Depending on the kind of constraints, one of the following Nelder-Mead simplex algorithms is used:

- unconstrained or only boundary constrained problems

  The original Nelder-Mead simplex algorithm is implemented and extended to boundary constraints. This algorithm does not compute the objective for infeasible points. This algorithm is automatically invoked if the LINCON or NLINCON statement is not specified.
- general linearly constrained or nonlinearly constrained problems

  A slightly modified version of Powell's (1992) COBYLA (Constrained Optimization BY Linear Approximations) implementation is used. This algorithm is automatically invoked if either the LINCON or the NLINCON statement is specified.

The original Nelder-Mead algorithm cannot be used for general linear or nonlinear constraints but can be faster for the unconstrained or boundary constrained case. The original Nelder-Mead algorithm changes the shape of the simplex adapting the nonlinearities of the objective function which contributes to an increased speed of convergence. The two NMSIMP subroutines use special sets of termination criteria. For more details, refer to the section "Termination Criteria" on page 443.

Powell's COBYLA Algorithm (COBYLA)

Powell's COBYLA algorithm is a sequential trust-region algorithm (originally with a monotonically decreasing radius ρ of a spheric trust region) that tries to maintain a regular-shaped simplex over the iterations. A small modification was made to the original algorithm, that permits an increase of the trust-region radius ρ in special situations. A sequence of iterations is performed with a constant trust-region radius ρ until the computed objective function reduction is much less than the predicted reduc-

tion. Then, the trust-region radius ρ is reduced. The trust-region radius is increased only if the computed function reduction is relatively close to the predicted reduction and the simplex is well-shaped. The start radius ρ_{beg} and the final radius ρ_{end} can be specified using ρ_{beg}=*INSTEP* and ρ_{end}=*ABSXTOL*. The convergence to small values of ρ_{end} (high precision) may take many calls of the function and constraint modules and may result in numerical problems. There are two main reasons for the slow convergence of the COBYLA algorithm:

- Only linear approximations of the objective and constraint functions are used locally.
- Maintaining the regular-shaped simplex and not adapting its shape to nonlinearities yields very small simplexes for highly nonlinear functions (for example, fourth-order polynomials).

Nonlinear Least-Squares Optimization

Levenberg-Marquardt Least-Squares Method (LEVMAR)

The Levenberg-Marquardt method is a modification of the trust-region method for nonlinear least-squares problems and is implemented as in Moré (1978).

This is the recommended algorithm for small- to medium-sized least-squares problems. Large least-squares problems can be transformed into minimization problems, which can be processed with conjugate gradient or (dual) quasi-Newton techniques. In each iteration, LEVMAR solves a quadratically constrained quadratic minimization problem that restricts the step to stay at the surface of or inside an n dimensional elliptical (or spherical) trust region. In each iteration, LEVMAR uses the crossproduct Jacobian matrix $\mathbf{J}^T\mathbf{J}$ as an approximate Hessian matrix.

Hybrid Quasi-Newton Least-Squares Methods (HYQUAN)

In each iteration of one of the Fletcher and Xu (1987) (refer also to AlBaali and Fletcher 1985, 1986) hybrid quasi-Newton methods, a criterion is used to decide whether a Gauss-Newton or a dual quasi-Newton search direction is appropriate. The VERSION= option can be used to choose one of three criteria (HY1, HY2, HY3) proposed by Fletcher and Xu (1987). The default is VERSION=2; that is, HY2. In each iteration, HYQUAN computes the crossproduct Jacobian (used for the Gauss-Newton step), updates the Cholesky factor of an approximate Hessian (used for the quasi-Newton step), and does a line-search to compute an approximate minimum along the search direction. The default line-search technique used by HYQUAN is especially designed for least-squares problems (refer to Lindström and Wedin 1984 and AlBaali and Fletcher, 1986). Using the LIS= option you can choose a different line-search algorithm than the default one.

Two update formulas can be specified with the UPDATE= option:

DBFGS performs the dual BFGS (Broyden, Fletcher, Goldfarb, and Shanno) update of the Cholesky factor of the Hessian matrix. This is the default.

DDFP performs the dual DFP (Davidon, Fletcher, and Powell) update of the Cholesky factor of the Hessian matrix.

The HYQUAN subroutine needs about the same amount of working memory as the LEVMAR algorithm. In most applications, LEVMAR seems to be superior to HYQUAN, and using HYQUAN is recommended only when problems are experienced with the performance of LEVMAR.

Finite-Difference Approximations of Derivatives

The FD= and FDHESSIAN= options specify the use of finite difference approximations of the derivatives. The FD= option specifies that all derivatives are approximated using function evaluations, and the FDHESSIAN= option specifies that second-order derivatives are approximated using gradient evalutions.

Computing derivatives by finite-difference approximations can be very time consuming, especially for second-order derivatives based only on values of the objective function (FD= option). If analytical derivatives are difficult to obtain (for example, if a function is computed by an iterative process), you might consider one of the optimization techniques that uses first-order derivatives only (TECH=QUANEW, TECH=DBLDOG, or TECH=CONGRA).

Forward Difference Approximations

The forward-difference derivative approximations consume less computer time but are usually not as precise as those using central-difference formulas.

- First-order derivatives: n additional function calls are needed:

$$g_i = \frac{\partial f}{\partial x_i} = \frac{f(x + h_i e_i) - f(x)}{h_i}$$

- Second-order derivatives based on function calls only (Dennis and Schnabel 1983, p. 80, 104): for dense Hessian, $n + n^2/2$ additional function calls are needed:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j}$$

- Second-order derivatives based on gradient calls (Dennis and Schnabel, 1983, p. 103): n additional gradient calls are needed:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{g_i(x + h_j e_j) - g_i(x)}{2h_j} + \frac{g_j(x + h_i e_i) - g_j(x)}{2h_i}$$

Central Difference Approximations

- First-order derivatives: $2n$ additional function calls are needed:

$$g_i = \frac{\partial f}{\partial x_i} = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i}$$

- Second-order derivatives based on function calls only (Abramowitz and Stegun 1972, p. 884): for dense Hessian, $2n + 4n^2/2$ additional function calls are needed:

$$\begin{aligned}
\frac{\partial^2 f}{\partial x_i^2} &= \frac{-f(x+2h_ie_i) + 16f(x+h_ie_i) - 30f(x) + 16f(x-h_ie_i) - f(x-2h_ie_i)}{12h_i^2} \\
\frac{\partial^2 f}{\partial x_i \partial x_j} &= \frac{f(x+h_ie_i+h_je_j) - f(x+h_ie_i-h_je_j) - f(x-h_ie_i+h_je_j) + f(x-h_ie_i-h_je_j)}{4h_ih_j}
\end{aligned}$$

- Second-order derivatives based on gradient: $2n$ additional gradient calls are needed:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{g_i(x+h_je_j) - g_i(x-h_je_j)}{4h_j} + \frac{g_j(x+h_ie_i) - g_j(x-h_ie_i)}{4h_i}$$

The FDIGITS= and CDIGITS= options can be used for specifying the number of accurate digits in the evaluation of objective function and nonlinear constraints. These specifications are helpful in determining an appropriate interval size h to be used in the finite-difference formulas.

The FDINT= option specifies whether the finite difference intervals h should be computed using an algorithm of Gill, Murray, Saunders, and Wright (1983) or based only on the information of the FDIGITS= and CDIGITS= options. For FDINT=OBJ, the interval h is based on the behavior of the objective function; for FDINT=CON, the interval h is based on the behavior of the nonlinear constraints functions; and for FDINT=ALL, the interval h is based on both, the behavior of the objective function and the nonlinear constraints functions. Note that the algorithm of Gill, Murray, Saunders, and Wright (1983) to compute the finite difference intervals h_j can be very expensive in the number of function calls. If the FDINT= option is specified, it is currently performed twice, the first time before the optimization process starts and the second time after the optimization terminates.

If FDINT= is not specified, the step sizes h_j, $j = 1, \ldots, n$, are defined as follows:

- for the forward-difference approximation of first-order derivatives using function calls and second-order derivatives using gradient calls: $h_j = \sqrt[2]{\eta_j}(1. + |x_j|)$,
- for the forward-difference approximation of second-order derivatives that use only function calls and all central-difference formulas: $h_j = \sqrt[3]{\eta_j}(1. + |x_j|)$.

where η is defined using the FDIGITS= option:

- If the number of accurate digits is specified with FDIGITS=r, η is set to 10^{-r}.
- If FDIGITS= is not specified, η is set to the machine precision ϵ.

For FDINT=OBJ and FDINT=ALL, the FDIGITS= specification is used in computing the forward and central finite-difference intervals.

If the problem has nonlinear constraints and the FD[=] option is specified, the first-order formulas are used to compute finite difference approximations of the Jacobian matrix $JC(x)$. You can use the CDIGITS= option to specify the number of accurate digits in the constraint evaluations to define the step sizes h_j, $j = 1, \ldots, n$. For FDINT=CON and FDINT=ALL, the CDIGITS= specification is used in computing the forward and central finite-difference intervals.

Note: If you are not able to specify analytic derivatives and the finite-difference approximations provided by PROC NLP are not good enough to solve your problem, you may program better finite-difference approximations using the GRADIENT, JACOBIAN, CRPJAC, or HESSIAN statement and the program statements.

Hessian and CRP Jacobian Scaling

The rows and columns of the Hessian and crossproduct Jacobian matrix can be scaled when using the trust-region, Newton-Raphson, Double Dogleg, and Levenberg-Marquardt optimization techniques. Each element $G_{i,j}$, $i, j = 1, \ldots, n$, is divided by the scaling factor $d_i * d_j$, where the scaling vector $d = (d_1, \ldots, d_n)$ is iteratively updated in a way specified by the HESCAL=i option, as follows:

$i = 0$: No scaling is done (equivalent to $d_i = 1$).

$i \neq 0$: First iteration and each restart iteration sets:

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

$i = 1$: refer to Moré (1978):

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

$i = 2$: refer to Dennis, Gay, and Welsch (1981):

$$d_i^{(k+1)} = \max(.6 * d_i^{(k)}, \sqrt{\max(|G_{i,i}^{(k)}|, \epsilon)})$$

$i = 3$: d_i is reset in each iteration:

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

where ϵ is the relative machine precision or, equivalently, the largest double precision value that when added to 1 results in 1.

Testing the Gradient Specification

There are three main ways to check the correctness of derivative specifications.

- Specify the FD[=] or FDHESSIAN[=] option in the PROC NLP statement to compute finite difference approximations of first- and second-order derivatives. In many applications, the finite difference approximations are computed with high precision and do not differ too much from the derivatives that are computed by specified formulas.
- Specify the GRADCHECK[=DETAIL] or GC[=DETAIL] option in the PROC NLP statement to compute and display a test vector and a test matrix of the gradient values at the start point $x^{(0)}$ by the method of Wolfe (1982). If you do not specify the GRADCHECK option, a fast derivative test identical to the GRADCHECK= FAST specification is done by default.
- If the default analytical derivative compiler is used or if derivatives are specified using the GRADIENT or JACOBIAN statement, the gradient or Jacobian computed at the initial point $x^{(0)}$ is tested by default using finite difference approximations. In some examples, the relative test can show significant differences between the two forms of derivatives and result in a warning message indicating that the specified derivatives could be wrong, even if they are correct. This happens especially in cases where the magnitude of the gradient at the starting point $x^{(0)}$ is small.

The algorithm of Wolfe (1982) is used to check whether the gradient $g(x)$ specified by a GRADIENT (or indirectly by a JACOBIAN) statement is appropriate for the objective function $f(x)$ specified by the program statements.

Using function and gradient evaluations in the neighborhood of the starting point $x^{(0)}$, second derivatives are approximated by finite difference formulas. Forward differences of gradient values are used to approximate the Hessian element G_{jk},

$$G_{jk} \approx H_{jk} = \frac{g_j(x + \delta e_k) - g_j(x)}{\delta}$$

where δ is a small step size and $e_k = (0, \ldots, 0, 1, 0, \ldots, 0)^T$ is the unit vector along the kth coordinate axis. The test vector S, with

$$s_j = H_{jj} - \frac{2}{\delta}\{\frac{f(x + \delta e_j) - f(x)}{\delta} - g_j(x)\}$$

contains the differences between two sets of finite difference approximations for the diagonal elements of the Hessian matrix

$$G_{jj} = \partial^2 f(x^{(0)})/\partial x_j^2 \quad , \quad j = 1, \ldots, n$$

The test matrix ΔH contains the absolute differences of symmetric elements in the approximate Hessian $|H_{jk} - H_{kj}|$, $j, k = 1, \ldots, n$, generated by forward differences of the gradient elements.

If the specification of the first derivatives is correct, the elements of the test vector and test matrix should be relatively small. The location of large elements in the test matrix points to erroneous coordinates in the gradient specification. For very large optimization problems, this algorithm can be too expensive in terms of computer time and memory.

Termination Criteria

All optimization techniques stop iterating at $x^{(k)}$ if at least one of a set of termination criteria is satisfied. PROC NLP also terminates if the point $x^{(k)}$ is fully constrained by n linearly independent active linear or boundary constraints, and all Lagrange multiplier estimates of active inequality constraints are greater than a small negative tolerance.

Since the Nelder-Mead simplex algorithm does not use derivatives, no termination criterion is available based on the gradient of the objective function. Powell's COBYLA algorithm uses only one more termination criterion. COBYLA is a trust-region algorithm that sequentially reduces the radius ρ of a spheric trust region beginning from a start radius ρ_{beg} = *INSTEP* to the final radius ρ_{end} = *ABSXTOL*. The default value is $\rho_{end} = 1e - 4$. The convergence to small values of ρ_{end} (high precision) may take many calls of the function and constraint modules and may result in numerical problems.

In some applications, the small default value of the ABSGCONV= criterion is too difficult to satisfy for some of the optimization techniques. This occurs most often when finite difference approximations of derivatives are used.

The default setting for the GCONV= option sometimes leads to early termination far from the location of the optimum. This is especially true for the special form of this criterion used in the CONGRA optimization.

The QUANEW algorithms for nonlinearly constrained optimization does not monotonically reduce either the value of the objective function or some kind of merit function which combines objective and constraint functions. Furthermore, the algorithm uses the watchdog technique with backtracking (Chamberlain et.al. 1982). Therefore, no termination criteria were implemented that are based on the values (x or f) of successive iterations. In addition to the criteria used by all optimization techniques, three more termination criteria are currently available, and are based on satisfying the Karush-Kuhn-Tucker conditions. For more information, refer to the section "Criteria for Optimality" on page 424, that requires that the gradient of the Lagrange function is zero at the optimal point (x^*, λ^*):

$$\nabla_x L(x^*, \lambda^*) = 0$$

Active Set Methods

The parameter vector $x \in \mathcal{R}^n$ may be subject to a set of m linear equality and inequality constraints:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad , \quad i = 1, \ldots, m_e$$

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad , \quad i = m_e + 1, \ldots, m$$

The coefficients a_{ij} and right-hand sides b_i of the equality and inequality constraints are collected in the $m \times n$ matrix A and the m vector b.

The m linear constraints define a feasible region $\mathcal{G}$ in $\mathcal{R}^n$ that must contain the point x^* that minimizes the problem. If the feasible region $\mathcal{G}$ is empty, no solution to the optimization problem exists.

All optimization techniques in PROC NLP (except those processing nonlinear constraints) are *active set methods*. The iteration starts with a feasible point $x^{(0)}$, which either is provided by the user or can be computed by the Schittkowski and Stoer (1979) algorithm implemented in PROC NLP. The algorithm then moves from one feasible point $x^{(k-1)}$ to a better feasible point $x^{(k)}$ along a feasible search direction $s^{(k)}$,

$$x^{(k)} = x^{(k-1)} + \alpha^{(k)}s^{(k)} \qquad \alpha^{(k)} > 0.$$

Theoretically, the path of points $x^{(k)}$ never leaves the feasible region $\mathcal{G}$ of the optimization problem, but it can hit its boundaries. The active set $\mathcal{A}^{(k)}$ of point $x^{(k)}$ is defined as the index set of all linear equality constraints and those inequality constraints that are satisfied at $x^{(k)}$. If no constraint is active $x^{(k)}$, the point is located in the interior of $\mathcal{G}$, and the active set $\mathcal{A}^{(k)} = \emptyset$ is empty. If the point $x^{(k)}$ in iteration k hits the boundary of inequality constraint i, this constraint i becomes active and is added to $\mathcal{A}^{(k)}$. Each equality constraint and each active inequality constraint reduces the dimension (degrees of freedom) of the optimization problem.

In practice, the active constraints can be satisfied only with finite precision. The LCEPSILON=r option specifies the range for active and violated linear constraints. If the point $x^{(k)}$ satisfies the condition

$$|\sum_{j=1}^{n} a_{ij}x_j^{(k)} - b_i| \leq t$$

where $t = r * (|b_i| + 1)$, 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. Due to rounding errors in computing the projected search direction, error can be accumulated so that an iterate $x^{(k)}$ steps out of the feasible region. In those cases, PROC NLP may try to pull the iterate $x^{(k)}$ into the feasible region. However, in some cases the algorithm needs to increase the feasible region by increasing

the LCEPSILON=r value. If this happens it is indicated by a message displayed in the log output.

If you cannot expect an improvement in the value of the objective function by moving from an active constraint back into the interior of the feasible region, you use this inequality constraint as an equality constraint in the next iteration. That means the active set $\mathcal{A}^{(k+1)}$ still contains the constraint i. Otherwise you release the active inequality constraint and increase the dimension of the optimization problem in the next iteration.

A serious numerical problem can arise when some of the active constraints become (nearly) linearly dependent. Linearly dependent equality constraints are removed before entering the optimization. You can use the LCSINGULAR= option to specify a criterion r used in the update of the QR decomposition that decides whether an active constraint is linearly dependent relative to a set of other active constraints.

If the final parameter set x^* is subjected to n_{act} linear equality or active inequality constraints, the QR decomposition of the $n \times n_{act}$ matrix $\hat{A}^T$ of the linear constraints is computed by $\hat{A}^T = QR$, where Q is an $n \times n$ orthogonal matrix and R is an $n \times n_{act}$ upper triangular matrix. The n columns of matrix Q can be separated into two matrices, $Q = [Y, Z]$, where Y contains the first n_{act} orthogonal columns of Q and Z the last $n - n_{act}$ orthogonal columns of Q. The $n \times (n - n_{act})$ column-orthogonal matrix Z is also called the nullspace matrix of the active linear constraints $\hat{A}^T$. The $n - n_{act}$ columns of the $n \times (n - n_{act})$ matrix Z form a basis orthogonal to the rows of the $n_{act} \times n$ matrix $\hat{A}$.

At the end of the iteration process, the PROC NLP can display the *projected gradient* g_Z,

$$g_Z = Z^T g$$

In the case of boundary constrained optimization, the elements of the projected gradient correspond to the gradient elements of the free parameters. A necessary condition for x^* to be a local minimum of the optimization problem is

$$g_Z(x^*) = Z^T g(x^*) = 0$$

The symmetric $n_{act} \times n_{act}$ matrix G_Z,

$$G_Z = Z^T G Z$$

is called a *projected Hessian matrix*. A second-order necessary condition for x^* to be a local minimizer requires that the projected Hessian matrix is positive semidefinite. If available, the projected gradient and projected Hessian matrix can be displayed and written in an OUTEST= or OUTVAR= data set.

Those elements of the n_{act} vector of first-order estimates of *Lagrange multipliers*,

$$\lambda = (\hat{A}\hat{A}^T)^{-1}\hat{A}ZZ^Tg,$$

which correspond to active inequality constraints indicate whether an improvement of the objective function can be obtained by releasing this active constraint. For minimization (maximization), a significant negative (positive) Lagrange multiplier indicates that a possible reduction (increase) of the objective function can be obtained by releasing this active linear constraint. The LCDEACT=r option can be used to specify a threshold r for the Lagrange multiplier that decides whether an active inequality constraint remains active or can be deactivated. The Lagrange multipliers are displayed (and written in an OUTEST= or OUTVAR= data set) only if linear constraints are active at the solution x^*. (In the case of boundary-constrained optimization, the Lagrange multipliers for active lower (upper) constraints are the negative (positive) gradient elements corresponding to the active parameters.)

Feasible Starting Point

Two algorithms are used to obtain a feasible starting point.

- When only boundary constraints are specified:
  - If the parameter x_j, $1 \leq j \leq n$, violates a two-sided boundary constraint (or an equality constraint) $l_j \leq x_j \leq u_j$, the parameter is given a new value inside the feasible interval, as follows:

$$\begin{aligned} x_j &= l_j, && \text{if } u_j \leq l_j \\ x_j &= l_j + \tfrac{1}{2}(u_j - l_j), && \text{if } u_j - l_j < 4 \\ x_j &= l_j + \tfrac{1}{10}(u_j - l_j), && \text{if } u_j - l_j \geq 4 \end{aligned}$$

  - If the parameter x_j, $j = 1, \ldots, n$, violates a one-sided boundary constraint $l_j \leq x_j$ or $x_j \leq u_j$, the parameter is given a new value near the violated boundary, as follows:

$$\begin{aligned} x_j &= l_j + \max(1, \tfrac{1}{10}l_j), && \text{if } l_j \text{ violated} \\ x_j &= u_j - \max(1, \tfrac{1}{10}u_j), && \text{if } u_j \text{ violated} \end{aligned}$$

- When general linear constraints are specified, a feasible point is computed by the algorithm of Schittkowski and Stoer (1979), that may be quite far from a user-specified infeasible point.

Line-Search Methods

In each iteration k, the (dual) quasi-Newton, hybrid quasi-Newton, conjugate gradient, and Newton-Raphson minimization techniques use iterative line-search algorithms that try to optimize a linear, quadratic, or cubic approximation of f along a feasible descent search direction $s^{(k)}$

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)}, \quad \alpha^{(k)} > 0$$

by computing an approximately optimal scalar $\alpha^{(k)}$.

Therefore, a line-search algorithm is an iterative process that optimizes a nonlinear function $f = f(\alpha)$ of one parameter (α) within each iteration k of the optimization technique, which itself tries to optimize a linear or quadratic approximation of the nonlinear objective function $f = f(x)$ of n parameters x. Since the outside iteration process is based only on the approximation of the objective function, the inside iteration of the line-search algorithm does not have to be perfect. Usually, the choice of α significantly reduces (in a minimization) the objective function. Criteria often used for termination of line-search algorithms are the Goldstein conditions (refer to Fletcher 1987).

Various line-search algorithms can be selected by using the LIS= option. The line-search method LIS=2 seems to be superior when function evaluation consumes significantly less computation time than gradient evaluation. Therefore, LIS=2 is the default value for Newton-Raphson, (dual) quasi-Newton, and conjugate gradient optimizations.

A special default line-search algorithm for TECH= HYQUAN is useful only for least-squares problems and cannot be chosen by the LIS= option. This method uses three columns of the $m \times n$ Jacobian matrix, which can for large m require more memory than using the algorithms designated by LIS=1 through LIS=8.

The line-search methods LIS=2 and LIS=3 can be modified to exact line-search by using the LSPRECISION= option (specifying the σ parameter in Fletcher, 1987). The line-search methods LIS=1, LIS=2, and LIS=3 satisfy the left-hand side and right-hand side Goldstein conditions (refer to Fletcher 1987). When derivatives are available, the line-search methods LIS=6, LIS=7,and LIS=8 try to satisfy the right-hand side Goldstein condition; if derivatives are not available, these line-search algorithms use only function calls.

Restricting the Step Length

Almost all line-search algorithms use iterative extrapolation techniques which can easily lead them to (feasible) points where the objective function f is no longer defined (For example, resulting in indefinite matrices for ML estimation) or difficult to compute (For example, resulting in floating point overflows). Therefore, PROC NLP provides options restricting the step length α or trust region radius Δ, especially during the first main iterations.

The inner product $g^T s$ of the gradient g and the search direction s is the slope of $f(\alpha) = f(x + \alpha s)$ along the search direction s. The default starting value $\alpha^{(0)} =$

$\alpha^{(k,0)}$ in each line-search algorithm ($\min_{\alpha>0} f(x+\alpha s)$) during the main iteration k is computed in three steps:

1. The first step uses either the difference $df = |f^{(k)} - f^{(k-1)}|$ of the function values during the last two consecutive iterations or the final stepsize value α^- of the last iteration $k-1$ to compute a first value of $\alpha_1^{(0)}$.
   - Not using the DAMPSTEP option:
     $$\alpha_1^{(0)} = \begin{cases} step & \text{if } 0.1 \le step \le 10 \\ 10 & \text{if } step > 10 \\ 0.1 & \text{if } step < 0.1 \end{cases}$$
     with
     $$step = \begin{cases} df/|g^T s| & \text{if } |g^T s| \ge \epsilon \max(100df, 1) \\ 1 & \text{otherwise} \end{cases}$$
     This value of $\alpha_1^{(0)}$ can be too large and lead to a difficult or impossible function evaluation, especially for highly nonlinear functions such as the EXP function.
   - Using the DAMPSTEP[=r] option:
     $$\alpha_1^{(0)} = \min(1, r\alpha^-)$$
     The initial value for the new step length can be no larger than r times the final step length α^- of the former iteration. The default value is $r = 2$.
2. During the first five iterations, the second step enables you to reduce $\alpha_1^{(0)}$ to a smaller starting value $\alpha_2^{(0)}$ using the INSTEP=r option:
   $$\alpha_2^{(0)} = \min(\alpha_1^{(0)}, r)$$
   After more than five iterations, $\alpha_2^{(0)}$ is set to $\alpha_1^{(0)}$.
3. The third step can further reduce the step length by
   $$\alpha_3^{(0)} = \min(\alpha_2^{(0)}, \min(10, u))$$
   where u is the maximum length of a step inside the feasible region.

The INSTEP=r option lets you specify a smaller or larger radius Δ of the trust region used in the first iteration of the trust-region, double dogleg, and the Levenberg-Marquardt algorithm. The default initial trust region radius $\Delta^{(0)}$ is the length of the scaled gradient (Moré 1978). This step corresponds to the default radius factor of $r = 1$. In most practical applications of the TRUREG, DBLDOG, and LEVMAR algorithms, this choice is successful. However, for bad initial values and highly nonlinear objective functions (such as the EXP function), the default start radius can result in arithmetic overflows. If this happens, you may try decreasing values of INSTEP=r, $0 < r < 1$, until the iteration starts successfully. A small factor r also affects the trust region radius $\Delta^{(k+1)}$ of the next steps because the radius is changed in each iteration by a factor $0 < c \le 4$, depending on the ratio ρ expressing the goodness of quadratic function approximation. Reducing the radius Δ corresponds to increasing the ridge parameter λ, producing smaller steps directed more closely toward the (negative) gradient direction.

Computational Problems

First Iteration Overflows

If you use bad initial values for the parameters, the computation of the value of the objective function (and its derivatives) can lead to arithmetic overflows in the first iteration. The line-search algorithms that work with cubic extrapolation are especially sensitive to arithmetic overflows. If an overflow occurs with an optimization technique that uses line-search, you can use the INSTEP= option to reduce the length of the first trial step during the line-search of the first five iterations or use the DAMPSTEP or MAX STEP option to restrict the step length of the initial α in subsequent iterations. If an arithmetic overflow occurs in gthe first iteration of the trust-region, double dogleg, or Levenberg-Marquardt algorithm, you can use the INSTEP= option to reduce the default trust region radius of the first iteration. You can also change the minimization technique or the line-search method. If none of these methods helps, consider the following actions:

- scale the parameters
- provide better initial values
- use boundary constraints to avoid the region where overflows may happen
- change the algorithm (specified in program statements) which computes the objective function

Problems in Evaluating the Objective Function

The starting point $x^{(0)}$ must be a point that can be evaluated by all the functions involved in your problem. However, during optimization the optimizer may iterate to a point $x^{(k)}$ where the objective function or nonlinear constraint functions and their derivatives cannot be evaluated. If you can identify the problematic region, you can prevent the algorithm from reaching it by adding another constraint to the problem. Another possiblity is a modification of the objective function, that will, as a result, get a large, undesired function value. As a result, the optimization algorithm reduces the step length and stays closer to the point that has been evaluated successfully in the previous iteration. For more information, refer to the section “Missing Values in Program Statements” on page 465.

Problems with Quasi-Newton Methods for Nonlinear Constraints

The sequential quadratic programming algorithm in QUANEW, that is used for solving nonlinearly constrained problems, can have problems updating the Lagrange multiplier vector μ. This results usually in very high values of the Lagrange function and in *watchdog* restarts indicated in the iteration history. If this happens, there are three actions you can try:

- By default, the Lagrange vector μ is evaluated in the same way as Powell (1982) describes. This corresponds to VERSION=2. By specifying VERSION=1, a modification of this algorithm replaces the update of the Lagrange vector μ with the original update of Powell (1978), that is used in VF02AD.
- You can use the INSTEP= option to impose an upper bound for the step size α during the first five iterations.

- You can use the INHESSIAN[=r] option to specify a different starting approximation for the Hessian. Choosing simply the INHESSIAN option will use the Cholesky factor of a (possibly ridged) finite difference approximation of the Hessian to initialize the quasi-Newton update process.

Other Convergence Difficulties

There are a number of things to try if the optimizer fails to converge.

- Check the derivative specification:
  If derivatives are specified by using the GRADIENT, HESSIAN, JACOBIAN, CRPJAC, or JACNLC statement, you can compare the specified derivatives with those computed by finite-difference approximations (specifying the FD and FDHESSIAN option). Use the GRADCHECK option to check if the gradient g is correct. For more information, refer to the section "Testing the Gradient Specification" on page 442.
- Forward-difference derivatives specified with the FD[=] or FDHESSIAN[=] option may not be precise enough to satisfy strong gradient termination criteria. You may need to specify the more expensive central-difference formulas or use analytical derivatives. The finite difference intervals may be too small or too big and the finite difference derivatives may be erroneous. You can specify the FDINT= option to compute better finite difference intervals.
- Change the optimization technique:
  For example, if you use the default TECH=LEVMAR, you can
  - change to TECH=QUANEW or to TECH=NRRIDG
  - run some iterations with TECH= CONGRA, write the results in an OUTEST= or OUTVAR= data set, and use them as initial values specified by an INEST= or INVAR= data set in a second run with a different TECH= technique
- Change or modify the update technique and the line-search algorithm:
  This method applies only to TECH=QUANEW, TECH=HYQUAN, or TECH= CONGRA. For example, if you use the default update formula and the default line-search algorithm, you can
  - change the update formula with the UPDATE= option
  - change the line-search algorithm with the LIS= option
  - specify a more precise line-search with the LSPRECISION= option, if you use LIS=2 or LIS=3
- Change the initial values by using a grid search specification to obtain a set of good feasible starting values.

Convergence to Stationary Point

The (projected) gradient at a stationary point is zero and that translates into a zero step size. The stopping criteria are satisfied.

There are two ways to avoid this situation:

- Use the PARMS statement to specify a grid of feasible starting points.

- Use the OPTCHECK[=*r*] option to avoid terminating at the stationary point.

The signs of the eigenvalues of the (reduced) Hessian matrix contain information regarding a stationary point.

- If all eigenvalues are positive, the Hessian matrix is positive definite and the point is a minimum point.
- If some of the eigenvalues are positive and all remaining eigenvalues are zero, the Hessian matrix is positive semidefinite and the point is a minimum or saddle point.
- If all eigenvalues are negative, the Hessian matrix is negative definite and the point is a maximum point.
- If some of the eigenvalues are negative and all remaining eigenvalues are zero, the Hessian matrix is negative semidefinite and the point is a maximum or saddle point.
- If all eigenvalues are zero, the point can be a minimum, maximum, or saddle point.

Precision of Solution

In some applications, PROC NLP may result in parameter estimates that are not precise enough. Usually this means that the procedure terminated too early at a point too far from the optimal point. The termination criteria define the size of the termination region around the optimal point. Any point inside this region can be accepted for terminating the optimization process. The default values of the termination criteria are set to satisfy a reasonable compromise between the computational effort (computer time) and the precision of the computed estimates for the most common applications. However, there are a number of circumstances where the default values of the termination criteria specify a region that is either too large or is too small. If the termination region is too large, then it can contain points with low precision. In such cases, you should inspect your log or list output to find the message stating which termination criterion terminated the optimization process. In many applications, you can obtain a solution with higher precision by simply using the old parameter estimates as starting values in a subsequent run where you specify a smaller value for the termination criterion that was satisfied at the former run.

If the termination region is too small, the optimization process may take longer to find a point inside such a region or cannot even find such a point due to rounding errors in function values and derivatives. This can easily happen in applications where finite difference approximations of derivatives are used and the GCONV and ABSGCONV termination criteria are too small to respect rounding errors in the gradient values.

Covariance Matrix

The COV= option must be specified to compute an approximate covariance matrix for the parameter estimates under asymptotic theory for least-squares, maximum-likelihood, or Bayesian estimation, with or without corrections for degrees of freedom as specified by the VARDEF= option.

Two groups of six different forms of covariance matrices (and therefore approximate standard errors) can be computed corresponding to the following two situations:

- The LSQ statement is specified, which means that least-squares estimates are being computed,

$$\min f(x) = \sum_{i=1}^{m} f_i^2(x)$$

- The MIN or MAX statement is specified, which means that maximum-likelihood or Bayesian estimates are being computed,

$$\text{opt } f(x) = \sum_{i=1}^{m} f_i(x)$$

  where opt is either min or max.

In either case, the following matrices are used:

$$G = \nabla^2 f(x)$$

$$\mathbf{J}(f) = (\nabla f_1, \ldots, \nabla f_m) = (\frac{\partial f_i}{\partial x_j})$$

$$\mathbf{JJ}(f) = \mathbf{J}(f)^T \mathbf{J}(f)$$

$$\mathbf{V} = \mathbf{J}(f)^T Diag(f_i^2) \mathbf{J}(f)$$

$$\mathbf{W} = \mathbf{J}(f)^T Diag(f_i^\dagger) \mathbf{J}(f)$$

where

$$f_i^\dagger = \begin{cases} 0 & \text{if } f_i = 0 \\ 1/f_i & \text{otherwise} \end{cases}$$

For unconstrained minimization, or when none of the final parameter estimates is subjected to linear equality or active inequality constraints, the formulas of the six types of covariance matrices areas follows.

COV		MIN or MAX Statement	LSQ Statement
1	M	$\frac{\text{\_NOBS\_}}{d} G^{-1} JJ(f) G^{-1}$	$\frac{\text{\_NOBS\_}}{d} G^{-1} V G^{-1}$
2	H	$\frac{\text{\_NOBS\_}}{d} G^{-1}$	$\sigma^2 G^{-1}$
3	J	$\frac{1}{d} W^{-1}$	$\sigma^2 JJ(f)^{-1}$
4	B	$\frac{1}{d} G^{-1} W G^{-1}$	$\sigma^2 G^{-1} JJ(f) G^{-1}$
5	E	$\frac{\text{\_NOBS\_}}{d} JJ(f)^{-1}$	$\frac{1}{d} V^{-1}$
6	U	$\frac{\text{\_NOBS\_}}{d} W^{-1} JJ(f) W^{-1}$	$\frac{\text{\_NOBS\_}}{d} JJ(f)^{-1} V JJ(f)^{-1}$

The value of d depends on the VARDEF= option and on the value of the _NOBS_ variable:

$$d = \begin{cases} \max(1, \text{\_NOBS\_} - \text{\_DF\_}) & \text{for VARDEF= DF} \\ \text{\_NOBS\_} & \text{for VARDEF= N} \end{cases}$$

where _DF_ is either set in the program statements or set by default to n (the number of parameters) and _NOBS_ is either set in the program statements or set by default to nobs $*$ mfun; nobs is the number of observations in the data set and mfun is the number of functions listed in the LSQ, MIN, or MAX statement.

The value σ^2 depends on the specifications of the SIGSQ= options and on the value of d:

$$\sigma^2 = \begin{cases} sq * \_NOBS\_/d & \text{if SIGSQ=}sq \text{ is specified} \\ 2 * f(x^*)/d & \text{if SIGSQ= is not specified} \end{cases}$$

where $f(x^*)$ is the value of the objective function at the optimal parameter estimates x^*.

The two groups of formulas distinguish between two situations:

- For least-squares estimates, the error variance can be estimated from the objective function value and is used in three of the six different forms of covariance matrices. If you have an independent estimate of the error variance, you can specify it with the SIGSQ= option.
- For maximum-likelihood or Bayesian estimates, the objective function should be the logarithm of the likelihood or of the posterior density when using the MAX statement.

For minimization, the inversion of the matrices in these formulas is done so that negative eigenvalues are considered zero, resulting always in a positive semidefinite covariance matrix.

In small samples, estimates of the covariance matrix based on asymptotic theory are often too small and should be used with caution.

If the final parameter estimates are subjected to $n_{act} > 0$ linear equality or active linear inequality constraints, the formulas of the covariance matrices are modified similar to Gallant (1987) and Cramer (1986, p. 38) and additionally generalized for applications with singular matrices. In the constrained case, the value of d used in the scalar factor σ^2 is defined by

$$d = \begin{cases} \max(1, \text{\_NOBS\_} - \text{\_DF\_} + n_{act}) & \text{for VARDEF= DF} \\ \text{\_NOBS\_} & \text{for VARDEF= N} \end{cases}$$

where n_{act} is the number of active constraints, and _NOBS_ is set as in the unconstrained case.

For minimization, the covariance matrix should be positive definite; for maximization it should be negative definite. There are several options available to check for a rank deficiency of the covariance matrix:

- The ASINGULAR=, MSINGULAR=, and VSINGULAR= options can be used to set three singularity criteria for the inversion of the matrix A needed

to compute the covariance matrix, when A is either the Hessian or one of the crossproduct Jacobian matrices. The singularity criterion used for the inversion is

$$|d_{j,j}| \leq \max(\text{ASING}, \text{VSING} * |A_{j,j}|, \text{MSING} * max(|A_{1,1}|, \ldots, |A_{n,n}|))$$

where $d_{j,j}$ is the diagonal pivot of the matrix A, and ASING, VSING and MSING are the specified values of the ASINGULAR=, VSINGULAR=, and MSINGULAR= options. The default values are

- ASING: the square root of the smallest positive double precision value
- MSING: $1e-12$ if the SINGULAR= option is not specified and $\max(10.*\epsilon, 1e-4*\text{SINGULAR})$ otherwise, where ϵ is the machine precision
- VSING: $1e-8$ if the SINGULAR= option is not specified and the value of SINGULAR otherwise

Note: In many cases, a normalized matrix $D^{-1}AD^{-1}$ is decomposed and the singularity criteria are modified correspondingly.

- If the matrix A is found singular in the first step, a generalized inverse is computed. Depending on the G4= option, either a generalized inverse satisfying all four Moore-Penrose conditions is computed or a generalized inverse satisfying only two Moore-Penrose conditions in general. If the number of parameters n of the application is less than or equal to G4=i, a G4 inverse is computed; otherwise only a G2 inverse is computed. The G4 inverse is computed by (the computationally very expensive but numerically stable) eigenvalue decomposition, the G2 inverse is computed by Gauss transformation. The G4 inverse is computed using the eigenvalue decomposition $A = Z\Lambda Z^T$, where Z is the orthogonal matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues, $\Lambda = diag(\lambda_1, ..., \lambda_n)$. If the PEIGVAL option is specified, the eigenvalues λ_i are displayed. The G4 inverse of A is set to

  $$A^- = Z\Lambda^- Z^T$$

  where the diagonal matrix $\Lambda^- = diag(\lambda_1^-, ..., \lambda_n^-)$ is defined using the COVSING= option

  $$\lambda_i^- = \begin{cases} 1/\lambda_i & \text{if } |\lambda_i| > COVSING \\ 0 & \text{if } |\lambda_i| \leq COVSING \end{cases}$$

  If the COVSING= option is not specified, the nr smallest eigenvalues are set to zero, where nr is the number of rank deficiencies found in the first step.

For optimization techniques that do not use second-order derivatives, the covariance matrix is usually computed using finite difference approximations of the derivatives. By specifying TECH= NONE, any of the covariance matrices can be computed using analytical derivatives. The covariance matrix specified by the COV= option can be displayed (using the PCOV option) and is written to the OUTEST= or OUTVAR= data set.

Input and Output Data Sets

DATA= Input Data Set

The DATA= data set is used only to specify an objective function f, that is a combination of m other functions f_i. For each function f_i, $i = 1, \ldots, m$, listed in a MAX, MIN, or LSQ statement, each observation l, $l = 1, \ldots, nobs$, in the DATA= data set defines a specific function f_{il} that is evaluated by substituting the values of the variables of this observation into the program statements. If the MAX or MIN statement is used, the $m * nobs$ specific functions f_{il} are added to a single objective function f. If the LSQ statement is used, the sum-of-squares f of the $m * nobs$ specific functions f_{il} is minimized. The NOMISS option causes observations with missing values to be skipped.

INEST= Input Data Set

The INEST= (or INVAR=, or ESTDATA=) input data set can be used to specify the initial values of the parameters defined in a PARMS statement as well as boundary constraints and the more general linear constraints which could be imposed on these parameters. This form of input is similar to the dense format input used in PROC LP.

The variables of the INEST= data set are

- a character variable _TYPE_ that indicates the type of the observation
- n numeric variables with the parameter names used in the PARMS statement
- the BY variables that are used in a DATA= input data set
- a numeric variable _RHS_ (right-hand side) (needed only if linear constraints are used)
- additional variables with names corresponding to constants used in the program statements

The content of the _TYPE_ variable defines the meaning of the observation of the INEST= data set. PROC NLP recognizes the following _TYPE_ values:

- PARMS, which specifies initial values for parameters. Additional variables can contain the values of constants that are referred to in program statements. The values of the constants in the PARMS observation initialize the constants in the program statements.
- UPPERBD | UB, which as specifies upper bounds. Missing values indicate that no upper bound is specified for the parameter.
- LOWERBD | LB, which specifies lower bounds. Missing values indicate that no lower bound is specified for the parameter.
- LE | <= | <, which specifies linear constraint $\sum_j a_{ij}x_j \leq b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing values indicate zeros.
- GE | >= | >, which specifies linear constraint $\sum_j a_{ij}x_j \geq b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing value indicates zeros.

- EQ | =, which specifies linear constraint $\sum_j a_{ij}x_j = b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing value indicates zeros.

The constraints specified in an INEST= data set are added to the constraints specified in BOUNDS and LINCON statements. You can use an OUTEST= data set as an INEST= data set in a subsequent run of PROC NLP. However, be aware that the OUTEST= data set also contains the boundary and general linear constraints specified in the former run of PROC NLP. When you are using this OUTEST= data set without changes as an INEST= data set, PROC NLP adds the constraints from the data set to the constraints specified by a BOUNDS and LINCON statement. Although PROC NLP automatically eliminates multiple identical constraints you should avoid specifying the same constraint a second time.

INQUAD= Input Data Set

Two types of INQUAD= data sets can be used to specify the objective function of a quadratic programming problem for TECH=QUADAS or TECH=LICOMP,

$$f(x) = \frac{1}{2}x^T Gx + g^T x + c, \quad \text{with} \quad G^T = G$$

The *dense* INQUAD= data set must contain all numerical values of the symmetric matrix G, vector g, and the value of the scalar c. Using the *sparse* INQUAD= data set allows to specify only the nonzero positions in matrix G and vector g. Those locations that are not set by the *sparse* INQUAD= data set are assumed to be zero.

Dense INQUAD= Data Set

A dense INQUAD= data set must contain two character variables _TYPE_ and _NAME_ and at least n numeric variables whose names are the parameter names. The _TYPE_ variable takes the following values:

- QUAD lists the n values of the row of the G matrix that is defined by the parameter name used in the _NAME_ variable.
- LINEAR lists the n values of the g vector.
- CONST sets the the value of the scalar c and cannot contain different numerical values; however, it could contain up to $n - 1$ missing values.
- PARMS specifies initial values for parameters.
- UPPERBD | UB specifies upper bounds. Missing value indicates that no upper bound is specified.
- LOWERBD | LB specifies lower bounds. The use of a missing value indicates that no lower bound.
- LE | <= | < specifies linear constraint $\sum_j a_{ij}x_j \leq b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing values indicate zeros.
- GE | >= | > specifies linear constraint $\sum_j a_{ij}x_j \geq b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing values indicate zeros.

- EQ | = specifies linear constraint $\sum_j a_{ij}x_j = b_i$. The n parameter values contain the coefficients a_{ij}, and the _RHS_ variable contains the right-hand side b_i. Missing values indicate zeros.

Constraints specified in a dense INQUAD= data set are added to the constraints specified in BOUNDS and LINCON statements.

Sparse INQUAD= Data Set

A sparse INQUAD= data set must contain three character variables _TYPE_, _ROW_, and _COL_ and one numeric variable _VALUE_. The _TYPE_ variable can assume three values:

- QUAD specifies that the _ROW_ and _COL_ variables define the row and column location of the value in the G matrix.
- LINEAR specifies that the _ROW_ variable defines the row location of the value in the g vector. The _COL_ variable is not used.

Using both the MODEL= option and the INCLUDE program statement with the same model file will include the file twice (erroneous in most cases).

OUT= Output Data Set

The OUT= data set contains those variables of a DATA= input data set that are referred to in the program statements and additionally variables computed by the program statements for the objective function. Specifying the NOMISS option enables you to skip observations with missing values in variables used in the program statements. The OUT= data set can also contain first- and second-order derivatives of these variables if the OUTDER= option is specified. The variables and derivatives are the final parameter estimates x^* or (for TECH=NONE) the initial value x^0.

The variables of the OUT= data set are:

- the BY variables and all other variables that are used in a DATA= input data set and referred to in the program code
- a variable _OBS_ containing the number of observations read from a DATA= input data set where the counting is restarted with the start of each BY group. If there is no DATA= input data set, then _OBS_=1
- a character variable _TYPE_ naming the type of the observation
- the parameter variables listed in the PARMS statement
- the function variables listed in the the MIN, MAX, or LSQ statement
- all other variables computed in the program statements
- the character variable _WRT_ (if OUTDER=1) containing the *with respect to* variable for which the first-order derivatives are written in the function variables
- the two character variables _WRT1_ and _WRT2_(if OUTDER=2) containing the two *with respect to* variables for which the first- and second-order derivatives are written in the function variables

OUTEST= Output Data Set

The OUTEST= or OUTVAR= output data set saves the optimization solutions of the use of the OUTEST= or OUTVAR= data set

- to save the values of the objective function on grid points to examine, for example, surface plots using PROC G3D (use the OUTGRID option)
- to avoid any costly computation of analytical (first- or second-order) derivatives during optimization when they only needed upon termination. In this case a two-step approach is recommended:
  1. In a first execution, the optimization is done; that is, optimal parameter estimates are computed, and the results are saved in an OUTEST= data set.
  2. In a subsequent execution, the optimal parameter estimates in the former OUTEST= data set are read in an INEST= data set and used with TECH=NONE to compute further results, such as analytical second-order derivatives or some kind of covariance matrix.
- to restart the procedure using parameter estimates as initial values
- to split a timeconsuming optimization problem into a series of smaller problems using intermediate results as initial values in a subsequent runs. (Refer to the MAXTIME=, MAXIT=, and MAXFU= options to trigger stopping in the section "PROC NLP Statement" on page 386)
- to write the value of the objective function, the parameter estimates, the time in seconds starting at the beginning of the optimization process and (if available) the gradient to the OUTEST= data set during the iterations. After the PROC NLP run is completed, the convergence progress can be inspected by graphically displaying the iterative information. (Refer to the OUTITER option in the section "PROC NLP Statement" on page 386)

The variables of the OUTEST= data set are

- the BY variables that are used in a DATA= input data set
- a character variable _TECH_ naming the optimization technique used
- a character variable _TYPE_ specifying the type of the observation
- a character variable _NAME_ naming the observation. For a linear constraint, the _NAME_ variable indicates whether the constraint is active at the solution. For the initial observations, the _NAME_ variable indicates if the number in the _RHS_ variable corresponds to the number of positive, negative, or zero eigenvalues
- n numeric variables with the parameter names used in the PARMS statement. These variables contain a point x of the parameter space, lower or upper boundary constraints, or the coefficients of linear constraints
- a numeric variable _RHS_ (right-hand side) that is used for the right-hand side value b_i of a linear constraints or for the value $f = f(x)$ of the objective function at a point x of the parameter space

- a numeric variable _ITER_, that is zero for initial values, equal to the iteration number for the OUTITER output, and missing for the result output

The _TYPE_ variable identifies how to interpret the observation. If _TYPE_ is:

- PARMS then parameter named variables contain the coordinates of the resulting point x^*. The _RHS_ variable contains $f(x^*)$.
- INITIAL then parameter named variables contain the feasible starting point $x^{(0)}$. The _RHS_ variable contains $f(x^{(0)})$.
- GRIDPNT then (if the OUTGRID option is specified) parameter named variables contain the coordinates of any point $x^{(k)}$ used in the grid search. The _RHS_ variable contains $f(x^{(k)})$.
- GRAD then parameter named variables contain the gradient at the initial or final estimates.
- STDERR then parameter named variables contain the approximate standard errors (square roots of the diagonal elements of the covariance matrix) if the COV= option is specified.
- _NOBS_ then (if the COV= options is specified) all parameter variables contain the value of _NOBS_ used in computing the σ^2 value in the formula of the covariance matrix.
- UPPERBD | UB then (if there are boundary constraints) the parameter variables contain the upper bounds.
- LOWERBD | LB then (if there are boundary constraints) the parameter variables contain the lower bounds.
- NACTBC then all parameter variables contain the number n_{abc} of active boundary constraints at the solution $x^{(*)}$.
- ACTBC then (if there are active boundary constraints) three observation indicate which of the parameters is actively constrained, as follows:

  NAME=GE the active lower bounds

  NAME=LE the active upper bounds

  NAME=EQ the active equality constraints

- NACTLC then all parameter variables contain the number n_{alc} of active linear constraints that are recognized as linear independent.
- NLDACTLC then all parameter variables contain the number of active linear constraints that are recognized as linearly dependent.
- LE then (if there are linear constraints) the observation contains the ith linear constraint $\sum_j a_{ij}x_j \leq b_i$. The parameter variables contain the coefficients a_{ij}, $j = 1, \ldots, n$, and the _RHS_ variable contains b_i. If the constraint i is active at the solution x^*, then _NAME_= 'ACTLC' or 'LDACTLC'.
- GE then (if there are linear constraints) the observation contains the ith linear constraint $\sum_j a_{ij}x_j \geq b_i$. The parameter variables contain the coefficients a_{ij}, $j = 1, \ldots, n$, and the _RHS_ variable contains b_i. If the constraint i is active at the solution x^*, then _NAME_= 'ACTLC' or 'LDACTLC'.

- EQ then (if there are linear constraints) the observation contains the ith linear constraint $\sum_j a_{ij}x_j = b_i$. The parameter variables contain the coefficients a_{ij}, $j = 1, \ldots, n$, the _RHS_ variable contains b_i, and _NAME_= 'ACTLC' or 'LDACTLC'.
- LAGRANGE then (if at least one of the linear constraints is an equality constraint or an inequality constraint that is active) the observation contains the vector of Lagrange multipliers. The Lagrange multipliers of active boundary constraints are listed first followed by those of active linear constraints and those of active nonlinear constraints. Lagrange multipliers are only available for the set of linearly independent active constraints.
- PROJGRAD then (if there are linear constraints) the observation contains the $n - n_{act}$ values of the projected gradient $g_Z = Z'g$ in the variables corresponding to the first $n - n_{act}$ parameters.
- JACOBIAN (then if the PJAC or OUTJAC option is specified) the m observations contain the m rows of the $m \times n$ Jacobian matrix The _RHS_ variable contains the row number l, $l = 1, \ldots, m$.
- HESSIAN then the first n observations contain the n rows of the (symmetric) Hessian matrix. The _RHS_ variable contains the row number j, $j = 1, \ldots, n$, and the _NAME_ variable contains the corresponding parameter name.
- PROJHESS then the first $n - n_{act}$ observations contain the $n - n_{act}$ rows of the projected Hessian matrix $Z^T G Z$. The _RHS_ variable contains the row number j, $j = 1, \ldots, n - n_{act}$, and the _NAME_ variable is blank.
- CRPJAC then the first n observations contain the n rows of the (symmetric) crossproduct Jacobian matrix at the solution. The _RHS_ variable contains the row number j, $j = 1, \ldots, n$, and the _NAME_ variable contains the corresponding parameter name.
- PROJCRPJ then the first $n - n_{act}$ observations contain the $n - n_{act}$ rows of the projected crossproduct Jacobian matrix $Z^T(J^T J)Z$. The _RHS_ variable contains the row number j, $j = 1, \ldots, n - n_{act}$, and the _NAME_ variable is blank.
- COV1, COV2, COV3, COV4, COV5, or COV6 then (depending on the COV= option) the first n observations contain the n rows of the (symmetric) covariance matrix of the parameter estimates. The _RHS_ variable contains the row number j, $j = 1, \ldots, n$, and the _NAME_ variable contains the corresponding parameter name.
- DETERMIN then contains the determinant $det = a*10^b$ of the matrix specified by the value of the _NAME_ variable where the value of the first variable in the PARMS statement and b is in _RHS_.
- NEIGPOS, NEIGNEG, or NEIGZER then the _RHS_ variable contains the number of positive, negative, and zero eigenvalues of the matrix specified by the value of the _NAME_ variable.
- COVRANK then the _RHS_ variable contains the rank of the covariance matrix.

- SIGSQ then the _RHS_ variable contains the scalar factor of the covariance matrix.
- _TIME_ then (if the OUTITER option is specified) the _RHS_ variable contains the number of seconds passed since the start of the optimization.
- TERMINAT then if optimization terminated at a point satisfying one of the termination criteria, an abbreviation of the corresponding criteria is given to the _NAME_ variable. Otherwise _NAME_='PROBLEMS'.

If for some reason the procedure does not terminate successfully (for example, no feasible initial values can be computed or the function value or derivatives at the starting point cannot be computed), the OUTEST= data set may contain only part of the observations (usually only the PARMS and GRAD observation).

Note: Generally you can use an OUTEST= or OUTVAR= data set as an INEST= or INVAR= data set in a further run of PROC NLP. However, be aware that the OUTEST= or OUTVAR= data set also contains the boundary and general linear constraints specified in the former run of PROC NLP. When you are using this OUTEST= data set without changes as an INEST= data set, PROC NLP adds the constraints from the data set to the constraints specified by a BOUNDS or LINCON statement. Although PROC NLP automatically eliminates multiple identical constraints you should avoid specifying the same constraint a second time.

Output of Profiles

The following observations are written to the OUTEST= data set only when the PROFILE statement or CLPARM option is specified

TYPE	_NAME_	_RHS_	Meaning of Observation
PLC_LOW	parname	y value	coordinates of lower CL for α
PLC_UPP	parname	y value	coordinates of upper CL for α
WALD_CL	LOWER	y value	lower Wald CL for α in _ALPHA_
WALD_CL	UPPER	y value	upper Wald CL for α in _ALPHA_
PL_CL	LOWER	y value	lower PL CL for α in _ALPHA_
PL_CL	UPPER	y value	upper PL CL for α in _ALPHA_
PROFILE	L(THETA)	missing	y value corresponding to x in following _NAME_=THETA
PROFILE	THETA	missing	x value corresponding to y in previous _NAME_=L(THETA)

Assume that the PROFILE statement specifies n_p parameters and n_α confidence levels. For CLPARM, $n_p = n$ and $n_\alpha = 4$.

- _TYPE_=PLC_LOW and _TYPE_=PLC_UPP:
  If CLPARM= option or the PROFILE statement with the OUTTABLE option is specified, then the complete set θ of parameter estimates (rather than only the confidence limit $x = \theta_j$) is written to the OUTEST= data set for each side of the confidence interval. This output may be helpful for further analyses on how small changes in $x = \theta_j$ affect the changes in the other $\theta_i, i \neq j$. The

ALPHA variable contains the corresponding value of α. There should be no more than $2n_\alpha n_p$ observations. If the confidence limit cannot be computed, the corresponding observation is not available.

- _TYPE_=WALD_CL:
  If CLPARM=WALD, CLPARM=BOTH, or the PROFILE statement with α values is specified, then the Wald confidence limits are written to the OUTEST= data set for each of the default or specified values of α. The _ALPHA_ variable contains the corresponding value of α. There should be $2n_\alpha$ observations.
- _TYPE_=PL_CL:
  If CLPARM=PL, CLPARM=BOTH, or the PROFILE statement with α values is specified, then the PL confidence limits are written to the OUTEST= data set for each of the default or specified value of α. The _ALPHA_ variable contains the corresponding values of α. There should be $2n_\alpha$ observations; some observations may have missing values.
- _TYPE_=PROFILE:
  If CLPARM=PL, CLPARM=BOTH, or the PROFILE statement with or without α values is specified, then a set of (x, y) point coordinates in two adjacent observations with _NAME_=L(THETA) (y value) and _NAME_=THETA (x value) is written to the OUTEST= data set. The _RHS_ and _ALPHA_ variables are not used (are set to missing). The number of observations depends on the difficulty of the optimization problems.

OUTMODEL= Output Data Set

The program statements for objective functions, nonlinear constraints, and derivatives can be saved into an OUTMODEL= output data set. This data set can be used in an INCLUDE program statement or as a MODEL= input data set in subsequent calls of PROC NLP. The OUTMODEL= option is similar to the option used in PROC MODEL in SAS/ETS software.

Storing Programs in Model Files

Models can be saved to and recalled from SAS catalog files. SAS catalogs are special files which can store many kinds of data structures as separate units in one SAS file. Each separate unit is called an entry, and each entry has an entry type that identifies its structure to the SAS system.

In general, to save a model, use the OUTMODEL=*name* option in the PROC NLP statement, where *name* is specified as *libref.catalog.entry*, *libref.entry*, or *entry*. The *libref*, *catalog*, and *entry* names must be valid SAS names no more than 8 characters long. The *catalog* name is restricted to 7 characters on the CMS operating system. If not given, the *catalog* name defaults to MODELS, and the *libref* defaults to WORK. The entry type is always MODEL. Thus, OUTMODEL=X writes the model to the file WORK.MODELS.X.MODEL.

The MODEL= option is used to read in a model. A list of model files can be specified in the MODEL= option, and a range of names with numeric suffixes can be given, as in MODEL=(MODEL1-MODEL10). When more than one model file is given, the list must be placed in parentheses, as in MODEL=(A B C), except in the case of a

single name. If more than one model file is specified, the files are combined in the order listed in the MODEL= option.

When the MODEL= option is specified in the PROC NLP statement and model definition statements are also given later in the PROC NLP step, the model files are read in first, in the order listed, and the model program specified in the PROC NLP step is appended after the model program read from the MODEL= files.

The INCLUDE statement can be used to append model code to the current model code. The contents of the model files are inserted into the current model at the position where the INCLUDE statement appears.

Note that the following statements are not part of the program code that is written to an OUTMODEL= data set: MIN, MAX, LSQ, MINQUAD, MAXQUAD, PARMS, BOUNDS, BY, CRPJAC, GRADIENT, HESSIAN, JACNLC, JACOBIAN, LABEL, LINCON, MATRIX, NLINCON.

Displayed Output

Procedure Initialization

After the procedure has processed the problem, it displays summary information about the problem and the options that you have selected. It may also display a list of linearly dependent constraints and other information about the constraints and parameters.

Optimization Start

At the start of optimization the procedure displays

- the number of constraints that are active at the starting point, or more precisely, the number of constraints that are currently members of the working set. If this number is followed by a plus sign, there are more active constraints, of which at least one is temporarily released from the working set due to negative Lagrange multipliers
- the value of the objective function at the starting point
- if the (projected) gradient is available, the value of the largest absolute (projected) gradient element
- for the TRUREG and LEVMAR subroutines, the initial radius of the trust region around the starting point

Iteration History

In general, the iteration history consists of one line of output containing the most important information for each iteration. The iteration-extensive Nelder-Mead simplex method, however, displays only one line for several internal iterations. This technique skips the output for some iterations because

- some of the termination tests (size and standard deviation) are rather time-consuming compared to the simplex operations and are only done every five simplex operation.

- the resulting history output is smaller

The _LIST_ variable (refer to the "Program Statements" section) also enables you to display the parameter estimates $x^{(k)}$ and the gradient $g^{(k)}$ in all or some selected iterations k.

The iteration history always includes the following (the words in parentheses indicate the column header output):

- the iteration number (iter)
- the number of iteration restarts (nrest)
- the number of function calls (nfun)
- the number of active constraints (act)
- the value of the optimization criterion (optcrit)
- the difference between adjacent function values (difcrit)
- the maximum of the absolute (projected) gradient components (maxgrad)

An apostrophe trailing the number of active constraints indicates that at least one of the active constraints was released from the active set due to a significant Lagrange multiplier.

The optimization history is displayed by default because it is important to check for possible convergence problems.

Optimization Termination

The output of the optimization history ends with a short output of information concerning the optimization result:

- the number of constraints that are active at the final point, or more precisely, the number of constraints that are currently members of the working set When this number is followed by a plus sign, it indicates that there are more active constraints of which at least one is temporarily released from the working set due to negative Lagrange multipliers.
- the value of the objective function at the final point
- if the (projected) gradient is available, the value of the largest absolute (projected) gradient element
- other information that is specific for the optimization technique

The NOPRINT option suppresses all output to the list file and only error's, warning's, and note's are displayed to the log file. The PALL option sets a large group of some of the commonly used specific displaying options, the PSHORT option suppresses some, and the PSUM (or PSUMMARY) option suppresses almost all of the default output. The following table summarizes the correspondence between the general and the specific print options

Output Options	PALL	default	PSHORT	PSUM	
	y	y	y	y	summary of optimization
	y	y	y	n	parameter estimates
	y	y	y	n	gradient of objective func
PHISTORY	y	y	y	n	iteration history
PINIT	y	y	n	n	setting of initial values
	y	y	n	n	listing of constraints
PGRID	y	n	n	n	results of grid search
PNLCJAC	y	n	n	n	Jacobian nonlin. constr.
PFUNCTION	y	n	n	n	values of functions
PEIGVAL	y	n	n	n	eigenvalue distribution
PCRPJAC	y	n	n	n	crossproduct Jacobian
PHESSIAN	y	n	n	n	Hessian matrix
PSTDERR	y	n	n	n	approx. standard errors
PCOV	y	n	n	n	covariance matrices
PJACOBI	n	n	n	n	Jacobian
LIST	n	n	n	n	model program, variables
LISTCODE	n	n	n	n	compiled model program

Missing Values

Missing Values in Program Statements

There is one very important reason for using missing values in program statements specifying the values of the objective functions and derivatives: it may not be possible to evaluate the program statements for a particular point x. For example, the extrapolation formula of one of the line-search algorithms may generate large x values for which the exp function cannot be evaluated without floating point overflow. The compiler of the program statements may check for such situations automatically, but it would be safer if you check the feasibility of your program statements. In some cases, the specification of boundary or linear constraints for parameters can avoid such situations. In many other cases, you can indicate that x is a *bad* point simply by returning a missing value for the objective function. In such cases the optimization algorithms in PROC NLP shorten the step size α or reduce the trust-region radius so that the next point will be closer to the point that was already successfully evaluated at the last iteration. Note that the starting point $x^{(0)}$ must be a point for which the program statements can be evaluated.

Missing Values in Input Data Sets

Observations with missing values in the DATA= data set for variables used in the objective function can lead to a missing value of the objective function implying that the corresponding BY group of data is not processed. The NOMISS option can be used to skip those observations of the DATA= data set for which relevant variables have missing values. Relevant are such variables that are referred to in program statements.

There can be different reasons to include observations with missing values in the INEST= data set. The value of the _RHS_ variable is not used in some cases and

can be missing. Missing values for the variables corresponding to parameters in the _TYPE_ = data set are as follows

- PARMS observations cause those parameters to have initial values assigned by the PARMS statement or by the RANDOM= or INITIAL= option.
- UPPERBD or LOWERBD observations cause those parameters to be unconstrained by upper or lower bounds.
- LE, GE, or EQ observations cause those parameters to have zero values in the constraint.

In general, missing values are treated as zeros.

Computational Resources

Since nonlinear optimization is an iterative process that depends on many factors, it is difficult to estimate how much computer time is necessary to compute an optimal solution satisfying one of the termination criteria. The MAXTIME=, MAXITER=, and MAXFU= options can be used to restrict the amount of CPU time, the number of iterations, and the number of function calls in a single run of PROC NLP.

In each iteration k, the NRRIDG and LEVMAR techniques use symmetric Householder transformations to decompose the $n \times n$ Hessian (crossproduct Jacobian) matrix G

$$G = V'TV \quad , \quad V\text{: orthogonal}, \quad T\text{: tridiagonal}$$

to compute the (Newton) search direction s

$$s^{(k)} = -G^{(k)-1}g^{(k)} \quad , \quad k = 1, 2, 3, \ldots .$$

The QUADAS, TRUREG, NEWRAP, and HYQUAN techniques use the Cholesky decomposition to solve the same linear system while computing the search direction. The QUANEW, DBLDOG, CONGRA, and NMSIMP techniques do not need to invert or decompose a Hessian or crossproduct Jacobian matrix and thus require less computational resources then the first group of techniques.

The larger the problem, the more time is spent computing function values and derivatives. Therefore, many researchers compare optimization techniques by counting and comparing the respective numbers of function, gradient, and Hessian (crossproduct Jacobian) evaluations. You can save computer time and memory by specifying derivatives (using the GRADIENT, JACOBIAN, CRPJAC, or HESSIAN statement) since you will typically produce a more efficient representation than the internal derivative compiler.

Finite difference approximations of the derivatives are expensive since they require additional function or gradient calls.

- Forward-difference formulas:
  - First-order derivatives: n additional function calls are needed.
  - Second-order derivatives based on function calls only: for a dense Hessian, $n + n^2/2$ additional function calls are needed.
  - Second-order derivatives based on gradient calls: n additional gradient calls are needed.
- Central-difference formulas:
  - First-order derivatives: $2n$ additional function calls are needed.
  - Second-order derivatives based on function calls only: for a dense Hessian, $2n + 2n^2$ additional function calls are needed.
  - Second-order derivatives based on gradient: $2n$ additional gradient calls are needed.

Many applications need considerably more time for computing second-order derivatives (Hessian matrix) than for first-order derivatives (gradient). In such cases, a (dual) quasi-Newton or conjugate gradient technique is recommended, that does not require second-order derivatives.

The following table shows for each optimization technique which derivatives are needed (FOD: first-order derivatives; SOD: second-order derivatives), what kind of constraints are supported (BC: boundary constraints; LIC: linear constraints), and the minimal memory (number of double floating point numbers) required. For various reasons, there are additionally about $7n + m$ double floating point numbers needed.

Quadratic Programming	FOD	SOD	BC	LIC	Memory
LICOMP	-	-	x	x	$18n + 3nn$
QUADAS	-	-	x	x	$1n + 2nn/2$
General Optimization	FOD	SOD	BC	LIC	Memory
TRUREG	x	x	x	x	$4n + 2nn/2$
NEWRAP	x	x	x	x	$2n + 2nn/2$
NRRIDG	x	x	x	x	$6n + nn/2$
QUANEW	x	-	x	x	$1n + nn/2$
DBLDOG	x	-	x	x	$7n + nn/2$
CONGRA	x	-	x	x	$3n$
NMSIMP	-	-	x	x	$4n + nn$
Least-Squares	FOD	SOD	BC	LIC	Memory
LEVMAR	x	-	x	x	$6n + nn/2$
HYQUAN	x	-	x	x	$2n + nn/2 + 3m$

Notes:

- Here, n denotes the number of parameters, nn the squared number of parameters, and $nn/2 := n(n + 1)/2$.

- The value of m is the product of the number of functions specified in the MIN, MAX, or LSQ statement and the maximum number of observation in each BY group of a DATA= input data set. The following table also contains the number v of variables in the DATA= data set that are used in the program statements.
- For a diagonal Hessian matrix, the $nn/2$ term in QUADAS, TRUREG, NEWRAP, and NRRIDG is replaced by n.
- If the TRUREG, NRRIDG, or NEWRAP method is used to minimize a least-squares problem, the second derivatives are replaced by the crossproduct Jacobian matrix.
- The memory needed by the TECH=NONE specification depends on the output specifications (typically, it needs $3n + nn/2$ double floating point numbers and an additional mn if the Jacobian matrix is required).

The total amount of memory needed to run an optimization technique consists of the technique-specific memory listed in the preceeding table, plus additional blocks of memory as shown in the following table.

	double	int	long	8byte
Basic Requirement	7n+m	n	3n	n+m
DATA= data set	v	-	-	v
JACOBIAN	m(n+2)	-	-	-
CRPJAC statement	nn/2	-	-	-
HESSIAN statement	nn/2	-	-	-
COV= statement	(2*)nn/2 + n	-	-	-
Scaling vector	n	-	-	-
BOUNDS statement	2n	n	-	-
Bounds in INEST=	2n	-	-	-
LINCON and TRUREG	c(n+1)+nn+ nn/2+4n	3c	-	-
LINCON and other	c(n+1)+nn+2nn/2+4n	3c	-	-

Notes:

- For TECH=LICOMP, the total amount of memory needed for the linear or boundary constrained case is $18(n + c) + 3(n + c)(n + c)$, where c is the number of constraints.
- The amount of memory needed to specify derivatives with a GRADIENT, JACOBIAN, CRPJAC, or HESSIAN statement (shown in this table) is small compared to that needed for using the internal function compiler to compute the derivatives. This is especially so for second-order derivatives.
- If the CONGRA technique is used, specifying the GRADCHECK [=DETAIL] option requires an additional $nn/2$ double floating point numbers to store the finite difference Hessian matrix.

Examples

Example 5.1. Using the DATA= Option

This example illustrates the use of the DATA= option. The Bard function (refer to More et al. 1981) is a least-squares problem with $n = 3$ parameters and $m = 15$ functions f_k:

$$f(x) = \frac{1}{2} \sum_{k=1}^{15} f_k^2(x), \quad x = (x_1, x_2, x_3),$$

where

$$\min f_k(x) = y_k - (x_1 + \frac{u_k}{v_k x_2 + w_k x_3})$$

with $u_k = k$, $v_k = 16 - k$, $w_k = \min(u_k, v_k)$, and

$$y_k = .14, .18, .22, .25, .29, .32, .35, .39, .37, .58, .73, .96, 1.34, 2.10, 4.39$$

The minimum function value $f(x^*) = 4.107e - 3$ is at the point $(0.08, 1.13, 2.34)$. The starting point $x^0 = (1, 1, 1)$ is used.

The following is the naive way of specifying the objective function.

```
proc nlp tech=levmar;
   lsq y1-y15;
   parms x1-x3 = 1;
   tmp1 = 15 * x2 + min(1,15) * x3;
   y1 = 0.14 - (x1 + 1 / tmp1);
   tmp1 = 14 * x2 + min(2,14) * x3;
   y2 = 0.18 - (x1 + 2 / tmp1);
   tmp1 = 13 * x2 + min(3,13) * x3;
   y3 = 0.22 - (x1 + 3 / tmp1);
   tmp1 = 12 * x2 + min(4,12) * x3;
   y4 = 0.25 - (x1 + 4 / tmp1);
   tmp1 = 11 * x2 + min(5,11) * x3;
   y5 = 0.29 - (x1 + 5 / tmp1);
   tmp1 = 10 * x2 + min(6,10) * x3;
   y6 = 0.32 - (x1 + 6 / tmp1);
   tmp1 = 9 * x2 + min(7,9) * x3;
   y7 = 0.35 - (x1 + 7 / tmp1);
   tmp1 = 8 * x2 + min(8,8) * x3;
   y8 = 0.39 - (x1 + 8 / tmp1);
   tmp1 = 7 * x2 + min(9,7) * x3;
   y9 = 0.37 - (x1 + 9 / tmp1);
   tmp1 = 6 * x2 + min(10,6) * x3;
   y10 = 0.58 - (x1 + 10 / tmp1);
   tmp1 = 5 * x2 + min(11,5) * x3;
   y11 = 0.73 - (x1 + 11 / tmp1);
```

```
   tmp1 = 4 * x2 + min(12,4) * x3;
   y12 = 0.96 - (x1 + 12 / tmp1);
   tmp1 = 3 * x2 + min(13,3) * x3;
   y13 = 1.34 - (x1 + 13 / tmp1);
   tmp1 = 2 * x2 + min(14,2) * x3;
   y14 = 2.10 - (x1 + 14 / tmp1);
   tmp1 = 1 * x2 + min(15,1) * x3;
   y15 = 4.39 - (x1 + 15 / tmp1);
run;
```

A more economical way to program this problem uses the DATA= option to input the 16 terms in $f(x)$.

```
   data bard;
      input r @@;
         w1 = 16. - _n_;
         w2 = min(_n_ , 16. - _n_);
         datalines;
   .14  .18  .22  .25  .29  .32  .35  .39
   .37  .58  .73  .96 1.34 2.10 4.39
   ;
   proc nlp data=bard tech=levmar;
      lsq y;
      parms x1-x3 = 1.;
      y = r - (x1 + _obs_ / (w1 * x2 + w2 * x3));
   end;
```

Another way you can specify the objective function uses the ARRAY statement and an explicit do loop, as in the following code.

```
   proc nlp tech=levmar;15] .14  .18  .22  .25  .29  .32  .35  .39  .37  .58
                  .73  .96 1.34 2.10 4.39 ;15] y1-y15;
      lsq y1-y15;
      parms x1-x3 = 1.;
      do i = 1 to 15;
        w1 = 16. - i;
       w2 = min(i , w1);
       w3 = w1 * x2 + w2 *i] = (x1 + i];
      end;
   run;
```

Example 5.2. Using the INQUAD= Option

This example illustrates the INQUAD= option for specifying a quadratic programming problem:

$$\min f(x) = \frac{1}{2}x^T G x + g^T x + c, \quad \text{with} \quad G^T = G,$$

Suppose that $c = -100$, $G = diag(.4, 4)$ and that $2 \leq x_1 \leq 50$, $-50 \leq x_2 \leq 50$, and $10 \leq 10x_1 - x_2$.

You specify the constant c and the Hessian G in the data set QUAD1. Notice that the _TYPE_ variable contains the keywords that identify how the procedure should interpret the observations.

```
data quad1;
   input _type_ $ _name_ $ x1 x2;
   datalines;
const   .   -100  -100
quad   x1    0.4     0
quad   x2      0     4
;
```

You specify the QUAD1 data set with the INQUAD= option. Notice that the names of the variables in the QUAD1 data set and the _NAME_ variable match the names of the parameters in the PARMS statement.

```
proc nlp inquad=quad1 all;
   min ;
   parms x1 x2 = -1;
   bounds  2 <= x1 <= 50,
         -50 <= x2 <= 50;
   lincon 10 <= 10 * x1 - x2;
run;
```

Alternatively, you can use a sparse format for specifying the c and G matrices eliminating the zeros. You use the special variables _ROW_, _COL_, and _VALUE_ to give the nonzero row and column names and value.

```
data quad2;
   input _type_ $ _row_ $ _col_ $ _value_;
   datalines;
const  .      .    -100
quad   x1    x1     0.4
quad   x2    x2       4
;
```

You can also include the constraints in the QUAD data set. Notice how the _TYPE_ variable contains keywords that identify how the procedure is to interpret the values in each observation.

```
data quad3;
   input _type_ $ _name_ $ x1   x2   _rhs_;
   datalines;
const      .     -100  -100   .
quad       x1    0.02     0   .
quad       x2    0.00     2   .
parms      .       -1    -1   .
lowerbd    .        2   -50   .
upperbd    .       50    50   .
ge         .       10    -1  10
```

```
proc nlp inquad=quad3;
   min ;
   parms x1 x2;
run;
```

Example 5.3. Using the INEST=Option

This example illustrates the use of the INEST= option for specifying a starting point and linear constraints. You name a data set with the INEST= option. The format of this data set is similar to the format of the QUAD= data set described in the previous example.

Consider the Hock and Schittkowski (1981) Problem # 24.

$$\min f(x) = \frac{((x_1 - 3)^2 - 9)x_2^3}{27\sqrt{3}}$$

subject to:

$$\begin{aligned} 0 &\leq x_1, x_2 \\ 0 &\leq .57735x_1 - x_2 \\ 0 &\leq x_1 + 1.732x_2 \\ 6 &\geq x_1 + 1.732x_2 \end{aligned}$$

with minimum function value $f(x^*) = -1$ at $x^* = (3, \sqrt{3})$. The feasible starting point is $x^0 = (1, .5)$.

You can specify this model in PROC NLP as follows:

```
proc nlp tech=trureg outest=res;
   min y;
   parms x1 = 1,
         x2 = .5;
   bounds  0 <= x1-x2;
   lincon  .57735 * x1 -         x2 >=  0,
                  x1 + 1.732 * x2 >=  0,
                 -x1 - 1.732 * x2 >= -6;
   y = (((x1 - 3)**2 - 9.) * x2**3) / (27 * sqrt(3));
run;
```

Note that none of the data for this model are in a data set. Alternatively, you can save the starting point $(1, .5)$ and the linear constraints in a data set. Notice that the _TYPE_ variable contains keywords that identify how the procedure is to interpret each of the observations and that the parameters in the problems X1 and X2 are variables in the data set. The observation with _TYPE_=LOWERBD gives the lower bounds on the parameters. The observation with _TYPE_=GE gives the coefficients for the first constraint. Similarly, the subsequent observations contain specifications for the other constraints. Also notice that the special variable _RHS_ contains the right-hand-side values.

```
data betts1(type=est);
   input _type_ $ x1 x2 _rhs_;
   datalines;
     parms    1        .5       .
     lowerbd  0        0        .
     ge       .57735   -1       .
     ge       1        1.732    .
     le       1        1.732    6
  ;
```

Now you can solve this problem with the following code. Notice that you specify the objective function and the parameters.

```
proc nlp inest=betts1 tech=trureg;
   min y;
   parms x1 x2;
   y = (((x1 - 3)**2 - 9) * x2**3) / (27 * sqrt(3));
run;
```

You can even include any constants used in the program statements in the INEST= data set. In the following code the variables A, B, C, and D contain some of the constants used in calculating the objective function Y.

```
data betts2(type=est);
   input _type_ $ x1    x2   _rhs_    a   b   c   d;
   datalines;
     parms    1        .5       .     3   9  27   3
     lowerbd  0        0        .     .   .   .   .
     ge       .57735   -1       0     .   .   .   .
     ge       1        1.732    0     .   .   .   .
     le       1        1.732    6     .   .   .   .
```

Notice that in the program statement for calculating Y, the constants are replaced by the A, B, C, and D variables.

```
proc nlp inest=betts2 tech=trureg;
   min y;
   parms x1 x2;
   y = (((x1 - a)**2 - b) * x2**3) / (c * sqrt(d));
run;
```

Example 5.4. Restarting an Optimization

This example shows how you can restart an optimization problem using the OUTEST=, INEST=, OUTMODEL=, and MODEL= options and how to save output into an OUT= data set. The least-squares solution of the Rosenbrock function using the trust-region method is used.

The following code solves the problem and saves the model in the MODEL data set and the solution in the EST and in OUT1 data sets.

```
proc nlp tech=trureg outmodel=model outest=est out=out1;
   lsq y1 y2;
   parms x1 = -1.2 ,
         x2 =  1.;
   y1 = 10. * (x2 - x1 * x1);
   y2 = 1. - x1;

proc print data=out1;
run;
```

The final parameter estimates $x^* = (1, 1)$ and the values of the functions $f_1 = Y1$ and $f_2 = Y2$ are written into an OUT= data set. Since OUTDER=0 is the default, the OUT= data set does not contain the Jacobian matrix.

Output 5.4.1. Solution in an OUT= Data Set

Obs	_OBS_	_TYPE_	y1	y2	x2	x1
1	1		0	-2.2204E-16	1	1

Next, the procedure reads the optimal parameter estimates from the INEST=EST data set and the model from the MODEL data set. It does not do any optimization (TECH=NONE) but it saves the Jacobian matrix to the OUT=OUT2 data set because of the option OUTDER=1. It also diplays the Jacobian matrix because of the option PJAC.

```
proc nlp tech=none model=model inest=est out=out2 outder=1 pjac;
   lsq y1 y2;
   parms x1 x2;
run;

proc print data=out2; run;
```

Output 5.4.2 displays the Jacobian matrix,

Output 5.4.2. Jacobian Matrix Output

PROC NLP: Least Squares Minimization

Jacobian Matrix

x1	x2
-20	10
-1	0

Output 5.4.3 shows the contents of the OUT2 data set, which also contains the Jacobian matrix.

Output 5.4.3. Jacobian Matrix in an OUT= Data Set

Obs	_OBS_	_TYPE_	y1	y2	_WRT_	x2	x1
1	1		0	-0		1	1
2	1	ANALYTIC	10	0	x2	1	1
3	1	ANALYTIC	-20	-1	x1	1	1

Example 5.5. Approximate Standard Errors

The NLP procedure provides a variety of ways for estimating parameters in nonlinear statistical models and for obtaining approximate standard errors and covariance matrices for the estimators. These methods are illustrated by estimating the mean of a random sample from a normal distribution with mean μ and standard deviation σ. The simplicity of the example makes it easy to compare the results of different methods in NLP with the usual estimator, the sample mean.

The following data is used:

```
data x; input x @@; datalines;
1 3 4 5 7
;
```

The standard error of the mean, computed with $n - 1$ degrees of freedom, is 1. The usual maximum-likelihood approximation to the standard error of the mean, using a variance divisor of n rather than $n - 1$, is 0.8944272.

The sample mean is a least-squares estimator, so it can be computed using an LSQ statement. Moreover, since this model is linear, the Hessian matrix and crossproduct Jacobian matrix are identical, and all three versions of the COV= option yield the same variance and standard error of the mean. Note that COV=j means that the crossproduct Jacobian is used. This is chosen because it requires the least computation.

```
proc nlp data=x cov=j pstderr pshort;
   lsq resid;
   parms mean=0;
   resid=x-mean;
run;
```

The results are the same as the usual estimates.

Output 5.5.1. Parameter Estimates

PROC NLP: Least Squares Minimization

Optimization Results
Parameter Estimates

N	Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
1	mean	4.000000	1.000000	4.000000	0.016130

Optimization Results
Parameter Estimates

Gradient Objective Function
8.881784E-15

Value of Objective Function = 10

PROC NLP can also compute maximum-likelihood estimates of μ and σ. In this case it is convenient to minimize the negative log likelihood. To get correct standard errors for maximum-likelihood estimators, the SIGSQ=1 option is required. The following program shows COV=1 but the output that follows has COV=2 and COV=3.

```
proc nlp data=x cov=1 sigsq=1 pstderr phes pcov pshort;
   min nloglik;
   parms mean=0, sigma=1;
   bounds 1e-12 < sigma;
   nloglik=.5*((x-mean)/sigma)**2 + log(sigma);
run;
```

The variance divisor is n instead of $n - 1$, so the standard error of the mean is 0.8944272 instead of 1. The standard error of the mean is the same with all six types of covariance matrix, but the standard error of the standard deviation varies. The sampling distribution of the standard deviation depends on the higher moments of the population distribution, so different methods of estimation can produce markedly different estimates of the standard error of the standard deviation.

Output 5.5.2 shows the output when COV=1.

Output 5.5.2. Solution for COV=1

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                      Approx                          Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.894427        4.472136        0.006566
2 sigma             2.000000        0.458258        4.364358        0.007260

                          Optimization Results
                           Parameter Estimates
                                 Gradient
                               Objective
                                Function

                             1.331492E-10
                             -5.606415E-9

                Value of Objective Function = 5.9657359028


                             Hessian Matrix

                                   mean             sigma

                mean       1.2500000028      -1.33149E-10
                sigma      -1.33149E-10       2.500000014

                        Determinant = 3.1250000245

                   Matrix has Only Positive Eigenvalues


                    Covariance Matrix 1: M = (NOBS/d)
                           inv(G) JJ(f) inv(G)

                                   mean             sigma

                mean                0.8      1.906775E-11
                sigma      1.906775E-11      0.2099999991

                             Factor sigm = 1


                        Determinant = 0.1679999993

                   Matrix has Only Positive Eigenvalues


                             Determinant = 1

                   Matrix has Only Positive Eigenvalues
```

Output 5.5.3 shows the output when COV=2.

Output 5.5.3. Solution for COV=2

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                      Approx                                Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.894427        4.472136        0.006566
2 sigma             2.000000        0.632456        3.162278        0.025031

                          Optimization Results
                           Parameter Estimates
                                 Gradient
                               Objective
                                Function

                             1.331492E-10
                             -5.606415E-9

                Value of Objective Function = 5.9657359028


                             Hessian Matrix

                                   mean             sigma

                mean       1.2500000028      -1.33149E-10
                sigma      -1.33149E-10       2.500000014

                        Determinant = 3.1250000245

                   Matrix has Only Positive Eigenvalues


                Covariance Matrix 2: H = (NOBS/d) inv(G)

                                   mean             sigma

                mean       0.7999999982      4.260769E-11
                sigma      4.260769E-11      0.3999999978

                             Factor sigm = 1


                        Determinant = 0.3199999975

                   Matrix has Only Positive Eigenvalues


                             Determinant = 1

                   Matrix has Only Positive Eigenvalues
```

Output 5.5.4 shows the output when COV=3.

Output 5.5.4. Solution for COV=3

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                      Approx                          Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.509136        7.856442        0.000537
2 sigma             2.000000        0.419936        4.762634        0.005048

                          Optimization Results
                           Parameter Estimates
                                 Gradient
                               Objective
                                Function

                             1.338402E-10
                             -5.940302E-9

                Value of Objective Function = 5.9657359028


                            Hessian Matrix

                                  mean             sigma

                mean       1.2500000028      -1.33149E-10
                sigma      -1.33149E-10       2.500000014

                        Determinant = 3.1250000245

                   Matrix has Only Positive Eigenvalues


                  Covariance Matrix 3: J = (1/d) inv(W)

                                  mean             sigma

                mean       0.2592197879      1.091093E-11
                sigma      1.091093E-11      0.1763460041

                            Factor sigm = 0.2


                        Determinant = 0.0457123738

                   Matrix has Only Positive Eigenvalues


                              Determinant = 1

                   Matrix has Only Positive Eigenvalues
```

Under normality, the maximum-likelihood estimators of μ and σ are independent, as indicated by the diagonal Hessian matrix in the previous example. Hence, the maximum-likelihood estimate of μ can be obtained by using any fixed value for σ, such as 1. However, if the fixed value of σ differs from the actual maximum-likelihood estimate (in this case 2), the model is misspecified and the standard errors obtained with COV=2 or COV=3 are incorrect. It is therefore necessary to use COV=1, that yields consistent estimates of the standard errors under a variety of forms of misspecification of the error distribution.

```
proc nlp data=x cov=1 sigsq=1 pstderr pcov pshort;
   min sqresid;
   parms mean=0;
   sqresid=.5*(x-mean)**2;
run;
```

This formulation produces the same standard error of the mean, 0.8944272 (see Output 5.5.5).

Output 5.5.5. Solution for FIXED σ and COV=1

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                      Approx                            Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.894427        4.472136        0.006566

                         Optimization Results
                          Parameter Estimates
                                Gradient
                               Objective
                                Function

                                      0

                   Value of Objective Function = 10


                            Covariance Matrix
                         1: M = (NOBS/d) inv(G)
                              JJ(f) inv(G)

                                              mean

                         mean                  0.8

                            Factor sigm = 1
```

The maximum-likelihood formulation with fixed σ is actually a least-squares problem. The objective function, parameter estimates, and Hessian matrix are the same as those in the first example in this section using the LSQ statement. However, the Jacobian matrix is different, each row being multiplied by twice the residual. To treat this formulation as a least-squares problem, the SIGSQ=1 option can be omitted. But since the Jacobian is not the same as in the formulation using the LSQ statement, the COV=1 | M and COV=3 | J options, that use the Jacobian, do not yield correct standard errors. The correct standard error is obtained with COV=2 | H, that uses only the Hessian matrix:

```
proc nlp data=x cov=2 pstderr pcov pshort;
   min sqresid;
   parms mean=0;
   sqresid=.5*(x-mean)**2;
run;
```

The results are the same as in the first example.

Output 5.5.6. Solution for Fixed σ and COV=2

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                      Approx                          Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 mean              4.000000        0.500000        8.000000        0.001324

                         Optimization Results
                          Parameter Estimates
                                 Gradient
                               Objective
                                Function

                                       0

                     Value of Objective Function = 10


                         Covariance Matrix 2:
                         H = (NOBS/d) inv(G)

                                        mean

                      mean               0.25

                        Factor sigm = 1.25
```

In summary, to obtain appropriate standard errors for least-squares estimates, you can use the LSQ statement with any of the COV= options, or you can use the MIN statement with COV=2. To obtain appropriate standard errors for maximum-likelihood estimates, you can use the MIN statement with the negative log likelihood or the MAX statement with the log likelihood, and in either case you can use any of the COV= options provided that you specify SIGSQ=1. You can also use a log-likelihood function with a misspecified scale parameter provided that you use SIGSQ=1 and COV=1. For nonlinear models, all of these methods yield approximations based on asymptotic theory, and should therefore be interpreted cautiously.

Example 5.6. Maximum Likelihood Weibull Estimation

Two-Parameter Weibull Estimation

The following data are taken from Lawless (1982, p.193) and represent the number of days it took rats painted with a carcinogen to develop carcinoma. The last 2 observations are censored data from a group of 19 rats:

```
title 'Lawless (1982): 2-Parameter Weibull MLE';
data pike;
   input days cens @@;
   datalines;
143  0  164  0  188  0  188  0
190  0  192  0  206  0  209  0
213  0  216  0  220  0  227  0
230  0  234  0  246  0  265  0
304  0  216  1  244  1
;
```

Suppose that you want to show how to compute the maximum likelihood estimates of the scale parameter σ (α in Lawless), the shape parameter c (β in Lawless), and the location parameter θ (μ in Lawless). The observed likelihood function of the three-parameter Weibull transformation (Lawless 1982, p.191) is

$$L(\theta,\sigma,c) = \frac{c^m}{\sigma^m} \prod_{i \in D} (\frac{t_i - \theta}{\sigma})^{c-1} \prod_{i=1}^{p} \exp(-(\frac{t_i - \theta}{\sigma})^c)$$

and the log likelihood is

$$l(\theta,\sigma,c) = m \log c - mc \log \sigma + (c-1) \sum_{i \in D} \log(t_i - \theta) - \sum_{i=1}^{p} (\frac{t_i - \theta}{\sigma})^c$$

The log likelihood function can only be evaluated for $\sigma > 0$, $c > 0$, and $\theta < \min_i t_i$. In the estimation process, you must enforce these conditions using lower and upper boundary constraints. The three-parameter Weibull estimation can be numerically difficult, and it usually pays off to provide good initial estimates. Therefore, you first estimate σ and c of the two-parameter Weibull distribution for constant $\theta = 0$. You then use the optimal parameters $\hat{\sigma}$ and $\hat{c}$ as starting values for the three-parameter Weibull estimation.

Although the use of an INEST= data set is not really necessary for this simple example, it illustrates how it is used to specify starting values and lower boundary constraints:

```
data par1(type=est);
   keep _type_ sig c theta;
        _type_='parms'; sig = .5;
        c = .5;    theta = 0;   output;
   _type_='lb';    sig = 1.0e-6;
        c = 1.0e-6; theta = .;  output;
```

The following PROC NLP call specifies the maximization of the log likelihood function for the two-parameter Weibull estimation for constant $\theta = 0$:

```
proc nlp data=pike tech=tr inest=par1 outest=opar1
     outmodel=model cov=2 vardef=n pcov phes;
   max logf;
   parms sig c;
   profile sig c / alpha = .9 to .1 by -.1 .09 to .01 by -.01;

   x_th = days - theta;
   s    = - (x_th / sig)**c;
   if cens=0 then s + log(c) - c*log(sig) + (c-1)*log(x_th);
   logf = s;
run;
```

After a few iterations you obtain the following solution.

Output 5.6.1. Optimization Results

```
                    PROC NLP: Nonlinear Maximization

                          Optimization Results
                           Parameter Estimates
                                      Approx                          Approx
N Parameter         Estimate         Std Err         t Value        Pr > |t|

1 sig             234.318611        9.645908       24.292021    9.050475E-16
2 c                 6.083147        1.068229        5.694611     0.000017269

                          Optimization Results
                           Parameter Estimates
                                  Gradient
                                 Objective
                                  Function

                               1.3372182E-9
                               -7.859277E-9

                Value of Objective Function = -88.23273515
```

Since the gradient has only small elements and the Hessian is negative definite (has only negative eigenvalues), the solution defines an isolated maximum point.

Output 5.6.2. Hessian Matrix at x^*

```
                    PROC NLP: Nonlinear Maximization

                Value of Objective Function = -88.23273515


                             Hessian Matrix

                                   sig                 c

                sig       -0.011457556       0.0257527577
                c          0.0257527577      -0.934221388

                       Determinant = 0.0100406894

                  Matrix has Only Negative Eigenvalues
```

The square roots of the diagonal elements of the approximate covariance matrix of parameter estimates are the approximate standard errors (ASE's).

Output 5.6.3. Covariance Matrix

```
                Covariance Matrix 2:
                 H = (NOBS/d) inv(G)

                          sig                 c

          sig      93.043549863      2.5648395794
          c        2.5648395794       1.141112488

                     Factor sigm = 1


                Determinant = 99.594754608

              Matrix has 2 Positive Eigenvalue(s)
```

The following confidence limits correspond to the α values in the PROFILE statement.

Output 5.6.4. Confidence Limits

PROC NLP: Nonlinear Maximization

Wald and PL Confidence Limits

N	Parameter	Estimate	Alpha	Profile Likelihood Confidence Limits	
1	sig	234.318611	0.900000	233.111324	235.532695
1	sig	.	0.800000	231.886549	236.772876
1	sig	.	0.700000	230.623280	238.063824
1	sig	.	0.600000	229.292797	239.436639
1	sig	.	0.500000	227.855829	240.935290
1	sig	.	0.400000	226.251597	242.629201
1	sig	.	0.300000	224.372260	244.643392
1	sig	.	0.200000	221.984557	247.278423
1	sig	.	0.100000	218.390824	251.394102
1	sig	.	0.090000	217.884162	251.987489
1	sig	.	0.080000	217.326988	252.645278
1	sig	.	0.070000	216.708814	253.383546
1	sig	.	0.060000	216.008815	254.228034
1	sig	.	0.050000	215.199301	255.215496
1	sig	.	0.040000	214.230116	256.411041
1	sig	.	0.030000	213.020874	257.935686
1	sig	.	0.020000	211.369067	260.066128
1	sig	.	0.010000	208.671091	263.687174
2	c	6.083147	0.900000	5.950029	6.217752
2	c	.	0.800000	5.815559	6.355576
2	c	.	0.700000	5.677909	6.499187
2	c	.	0.600000	5.534275	6.651789
2	c	.	0.500000	5.380952	6.817880
2	c	.	0.400000	5.212344	7.004485
2	c	.	0.300000	5.018784	7.225733
2	c	.	0.200000	4.776379	7.506166
2	c	.	0.100000	4.431310	7.931669
2	c	.	0.090000	4.382687	7.991457
2	c	.	0.080000	4.327815	8.056628
2	c	.	0.070000	4.270773	8.129238
2	c	.	0.060000	4.207130	8.211221
2	c	.	0.050000	4.134675	8.306218
2	c	.	0.040000	4.049531	8.418782
2	c	.	0.030000	3.945037	8.559677
2	c	.	0.020000	3.805759	8.749130
2	c	.	0.010000	3.588814	9.056751

Three-Parameter Weibull Estimation

You now prepare for the three-parameter Weibull estimation by using PROC UNIVARIATE to obtain the smallest data value for the upper boundary constraint for θ. For this small problem, you can do this much more simple by just using a value slightly smaller than the minimum data value 143.

```
/* Calculate upper bound for theta parameter */
proc univariate data=pike noprint;
   var days;
   output out=stats n=nobs min=minx range=range;

data stats;
   set stats;
   keep _type_ theta;

   /* 1. write parms observation */
```

```
   theta = minx - .1 * range;
   if theta < 0 then theat = 0
   _type_ = 'parms'; output;

   /* 2. write ub observation */
   theta = minx * (1 - 1e-4);
   _type_ = 'ub'; output;
```

The data set PAR2 specifies the starting values and the lower and upper bounds for the three-parameter Weibull problem:

```
proc sort data=opar1;
   by _type_;

data par2(type=est);
   merge opar1(drop=theta) stats;
   by _type_;
   keep _type_ sig c theta;
   if _type_ in ('parms' 'lowerbd' 'ub');
```

The following PROC NLP call uses the MODEL= input data set containing the log likelihood function that was saved during the two-parameter Weibull estimation:

```
proc nlp data=pike tech=tr inest=par2 outest=opar2
      model=model cov=2 vardef=n pcov phes;
      max logf;
   parms sig c theta;
   profile sig c theta / alpha = .5 .1 .05 .01;
run;
```

After a few iterations, you obtain the following solution.

Output 5.6.5. Optimization Results

PROC NLP: Nonlinear Maximization

Optimization Results
Parameter Estimates

N	Parameter	Estimate	Approx Std Err	t Value	Approx Pr > |t|
1	sig	108.382670	32.573286	3.327348	0.003540
2	c	2.711475	1.058756	2.561000	0.019108
3	theta	122.026001	28.692328	4.252914	0.000430

Optimization Results
Parameter Estimates

Gradient Objective Function
-4.598334E-9
-0.000000714
-9.916609E-8

Value of Objective Function = -87.32424712

From inspecting the first- and second-order derivatives at the optimal solution, you can verify that you obtained an isolated maximum point.

Output 5.6.6. Hessian Matrix

PROC NLP: Nonlinear Maximization

Value of Objective Function = -87.32424712

Hessian Matrix

	sig	c	theta
sig	-0.010639974	0.0453887849	-0.010033749
c	0.0453887849	-4.078687936	-0.083026332
theta	-0.010033749	-0.083026332	-0.014752091

Determinant = 0.0000502116

Matrix has Only Negative Eigenvalues

The square roots of the diagonal elements of the approximate covariance matrix of parameter estimates are the approximate standard errors.

Output 5.6.7. Covariance Matrix

Covariance Matrix 2: H = (NOBS/d) inv(G)

	sig	c	theta
sig	1061.0122895	29.925873264	-890.0802845
c	29.925873264	1.1209598699	-26.66315923
theta	-890.0802845	-26.66315923	823.24374725

Factor sigm = 1

Determinant = 19915.386855

Matrix has 3 Positive Eigenvalue(s)

The difference between the Wald and profile CL's for parameter PHI2 are remarkable, especially for the upper 95% and 99% limits.

Output 5.6.8. Confidence Limits

PROC NLP: Nonlinear Maximization

Wald and PL Confidence Limits

N	Parameter	Estimate	Alpha	Profile Likelihood Confidence Limits	
1	sig	108.382730	0.500000	91.811562	141.564605
1	sig	.	0.100000	76.502373	.
1	sig	.	0.050000	72.215845	.
1	sig	.	0.010000	64.262384	.
2	c	2.711477	0.500000	2.139297	3.704052
2	c	.	0.100000	1.574162	9.250072
2	c	.	0.050000	1.424853	19.516224
2	c	.	0.010000	1.163096	19.540738
3	theta	122.025944	0.500000	91.027145	135.095454
3	theta	.	0.100000	.	141.833769
3	theta	.	0.050000	.	142.512603
3	theta	.	0.010000	.	142.967407

Wald and PL Confidence Limits

Wald Confidence Limits

86.412311	130.353149
54.804268	161.961192
44.540054	172.225405
24.479232	192.286228
1.997355	3.425599
0.969973	4.452981
0.636347	4.786607
-0.015706	5.438660
102.673191	141.378698
74.831088	142.985700
65.789804	142.985700
48.119128	142.985700

Example 5.7. Simple Pooling Problem

The following optimization problem is discussed in Haverly (1978) and in Liebman et al. (1986, pp.127-128). Two liquid chemicals, X and Y, are produced by the pooling and blending of three input liquid chemicals, A, B, and C. You know the sulfur impurity amounts of the input chemicals, and you have to respect upper limits of the sulfur impurity amounts of the output chemicals. The sulfur concentrations and the prices of the input and output chemicals are

- Chemical A: Concentration = 3%, Price= $ 6
- Chemical B: Concentration = 1%, Price= $ 16
- Chemical C: Concentration = 2%, Price= $ 10
- Chemical X: Concentration $\leq$ 2.5%, Price= $ 9
- Chemical Y: Concentration $\leq$ 1.5%, Price= $ 15

The problem is complicated by the fact that the two input chemicals A and B are available only as a mixture (they are either shipped together or stored together). Be-

cause the amounts of A and B are unknown, the sulfur concentration of the mixture is also unknown.

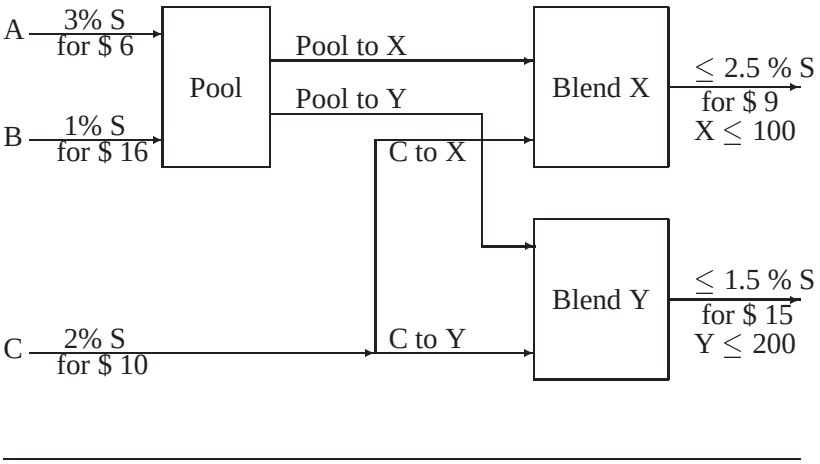

Figure 13: Pooling of Liquid Chemicals

You know which customers will buy no more than 100 units of X and 200 units of Y. The problem is determining how to operate the pooling and blending of the chemicals to maximize the profit. The objective function for the profit is

$$\begin{aligned} profit &= cost(x) * amount(x) + cost(y) * amount(y) \\ &- cost(a) * amount(a) - cost(b) * amount(b) - cost(c) * amount(c) \end{aligned}$$

There are three groups of constraints:

1. The first group of constraint functions is the mass balance restrictions illustrated by the graph. These are four linear equality constraints:
   - $amount(a) + amount(b) = pool\_to\_x + pool\_to\_y$
   - $pool\_to\_x + c\_to\_x = amount(x)$
   - $pool\_to\_y + c\_to\_y = amount(y)$
   - $amount(c) = c\_to\_x + c\_to\_y$
2. You introduce a new variable, $pool\_s$, that represents the sulfur concentration of the pool. Using $pool\_s$ and the sulfur concentration of C (2%), you obtain two nonlinear inequality constraints for the sulfur concentrations of X and Y, one linear equality constraint for the sulfur balance, and lower and upper boundary restrictions for $pool\_s$:
   - $pool\_s * pool\_to\_x + 2 * c\_to\_x \leq 2.5 * amount(x)$
   - $pool\_s * pool\_to\_y + 2 * c\_to\_y \leq 1.5 * amount(y)$

- $3 * amount(a) + 1 * amount(b) = pool\_s * (amount(a) + amount(b))$
- $1 \leq pool\_s \leq 3$

3. The last group assembles the remaining boundary constraints. First, you do not want to produce more than you can sell; and finally, all variables must be nonnegative:
   - $amount(x) \leq 100, amount(y) \leq 200$
   - $amount(a), amount(b), amount(c), amount(x), amount(y) \geq 0$
   - $pool\_to\_x, pool\_to\_y, c\_to\_x, c\_to\_y \geq 0$

There exist several local optima to this problem that can be found by specifying different starting points. Using the starting point $amount(a)$, $amount(b)$, $amount(c)$, $amount(x)$, $amount(y)$, $pool\_to\_x$, $pool\_to\_y$, c_to_x, c_to_y, $pool\_s$, PROC NLP finds a solution with $profit = 400$:

```
proc nlp all;
   parms amountx amounty amounta amountb amountc
         pooltox pooltoy ctox ctoy pools = 1;
   bounds 0 <= amountx amounty amounta amountb amountc,
                amountx <= 100,
                amounty <= 200,
          0 <= pooltox pooltoy ctox ctoy,
          1 <= pools <= 3;
   lincon amounta + amountb = pooltox + pooltoy,
          pooltox + ctox = amountx,
          pooltoy + ctoy = amounty,
          ctox + ctoy    = amountc;
   nlincon nlc1-nlc2 >= 0.,
           nlc3 = 0.;
   max f;
   costa = 6; costb = 16; costc = 10;
   costx = 9; costy = 15;
   f = costx * amountx + costy * amounty
       - costa * amounta - costb * amountb - costc * amountc;
   nlc1 = 2.5 * amountx - pools * pooltox - 2. * ctox;
   nlc2 = 1.5 * amounty - pools * pooltoy - 2. * ctoy;
   nlc3 = 3 * amounta + amountb - pools * (amounta + amountb);
run;
```

The specified starting point was not feasible with respect to the linear equality constraints; therefore, a starting point is generated that satisfies linear and boundary constraints.

Output 5.7.1. Starting Estimates

PROC NLP: Nonlinear Maximization

Optimization Start
Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function	Gradient Lagrange Function	Lower Bound Constraint	Upper Bound Constraint
1	amountx	1.363636	9.000000	-0.843698	0	100.000000
2	amounty	1.363636	15.000000	-0.111882	0	200.000000
3	amounta	0.818182	-6.000000	-0.430733	0	.
4	amountb	0.818182	-16.000000	-0.542615	0	.
5	amountc	1.090909	-10.000000	0.017768	0	.
6	pooltox	0.818182	0	-0.669628	0	.
7	pooltoy	0.818182	0	-0.303720	0	.
8	ctox	0.545455	0	-0.174070	0	.
9	ctoy	0.545455	0	0.191838	0	.
10	pools	2.000000	0	0.068372	1.000000	3.000000

Value of Objective Function = 3.8181818182

Value of Lagrange Function = -2.866739915

The starting point satisfies the four equality constraints.

Output 5.7.2. Linear Constraints

PROC NLP: Nonlinear Maximization

Linear Constraints

```
1 -3.331E-16 : ACT       0  ==   +    1.0000 * amounta  +    1.0000 * amountb
   -    1.0000 * pooltox  -    1.0000 * pooltoy
2 1.1102E-16 : ACT       0  ==   -    1.0000 * amountx  +    1.0000 * pooltox
   +    1.0000 * ctox
3 1.1102E-16 : ACT       0  ==   -    1.0000 * amounty  +    1.0000 * pooltoy
   +    1.0000 * ctoy
4 1.1102E-16 : ACT       0  ==   -    1.0000 * amountc  +    1.0000 * ctox
   +    1.0000 * ctoy
```

Output 5.7.3. Nonlinear Constraints

PROC NLP: Nonlinear Maximization

Values of Nonlinear Constraints

	Constraint	Value	Residual	Lagrange Multiplier		
[5]	nlc3	0	0	4.9441	Active	NLEC
[6]	nlc1_G	0.6818	0.6818	.		
[7]	nlc2_G	-0.6818	-0.6818	-9.8046	Violat.	NLIC

This following table shows the settings of some important PROC NLP options.

Output 5.7.4. Options

PROC NLP: Nonlinear Maximization

Minimum Iterations	0
Maximum Iterations	200
Maximum Function Calls	500
Iterations Reducing Constraint Violation	20
ABSGCONV Gradient Criterion	0.00001
GCONV Gradient Criterion	1E-8
ABSFCONV Function Criterion	0
FCONV Function Criterion	2.220446E-16
FCONV2 Function Criterion	1E-6
FSIZE Parameter	0
ABSXCONV Parameter Change Criterion	0
XCONV Parameter Change Criterion	0
XSIZE Parameter	0
ABSCONV Function Criterion	1.340781E154
Line Search Method	2
Starting Alpha for Line Search	1
Line Search Precision LSPRECISION	0.4
DAMPSTEP Parameter for Line Search	.
FD Derivatives: Accurate Digits in Obj.F	15.653559775
FD Derivatives: Accurate Digits in NLCon	15.653559775
Singularity Tolerance (SINGULAR)	1E-8
Constraint Precision (LCEPS)	1E-8
Linearly Dependent Constraints (LCSING)	1E-8
Releasing Active Constraints (LCDEACT)	.

The iteration history does not show any problems.

Output 5.7.5. Optimization History

PROC NLP: Nonlinear Maximization

Dual Quasi-Newton Optimization

Modified VMCWD Algorithm of Powell (1978, 1982)
Dual Quasi-Newton Optimization
Modified VMCWD Algorithm of Powell (1978, 1982)

Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
Lagrange Multiplier Update of Powell(1982)

Iter	Restarts	Function Calls	Objective Function	Maximum Constraint Violation	Predicted Function Reduction	Step Size	Maximum Gradient Element of the Lagrange Function
1	0	19	-1.42400	0.00962	6.9131	1.000	0.783
2′	0	20	2.77026	0.0166	5.3770	1.000	2.629
3	0	21	7.08706	0.1409	7.1965	1.000	9.452
4′	0	22	11.41264	0.0583	15.5769	1.000	23.390
5′	0	23	24.84613	8.88E-16	496.1	1.000	147.6
6	0	24	378.22825	147.4	3316.7	1.000	840.4
7′	0	25	307.56810	50.9339	607.9	1.000	27.143
8′	0	26	347.24468	1.8329	21.9883	1.000	28.482
9′	0	27	349.49255	0.00915	7.1833	1.000	28.289
10′	0	28	356.58341	0.1083	50.2566	1.000	27.479
11′	0	29	388.70731	2.4280	24.7996	1.000	21.114
12′	0	30	389.30118	0.0157	10.0475	1.000	18.647
13′	0	31	399.19240	0.7997	11.1862	1.000	0.416
14′	0	32	400.00000	0.0128	0.1533	1.000	0.00087
15′	0	33	400.00000	7.38E-11	2.44E-10	1.000	365E-12

Optimization Results

Iterations	15	Function Calls	34
Gradient Calls	18	Active Constraints	10
Objective Function	400	Maximum Constraint Violation	7.381118E-11
Maximum Projected Gradient	0	Value Lagrange Function	-400
Maximum Gradient of the Lagran Func	1.065814E-14	Slope of Search Direction	-2.43574E-10

FCONV2 convergence criterion satisfied.

The optimal solution shows that to obtain the maximum profit of $ 400, you need only to produce the maximum 200 units of blending Y and no units of blending X

Output 5.7.6. Optimization Solution

```
                          Optimization Results
                           Parameter Estimates
                              Gradient      Gradient  Active
                             Objective      Lagrange  Bound
 N Parameter       Estimate     Function      Function  Constraint

 1 amountx     -1.40474E-11     9.000000             0   Lower BC
 2 amounty       200.000000    15.000000             0   Upper BC
 3 amounta     1.027701E-16    -6.000000             0   Lower BC
 4 amountb       100.000000   -16.000000  -1.77636E-15
 5 amountc       100.000000   -10.000000  1.776357E-15
 6 pooltox     7.024003E-12             0             0   Lower BC
 7 pooltoy       100.000000             0  -1.06581E-14
 8 ctox        -2.10714E-11             0  5.329071E-15   Lower BC      LinDep
 9 ctoy          100.000000             0  1.776357E-15
10 pools           1.000000             0             0   Lower BC      LinDep

                    Value of Objective Function = 400


                    Value of Lagrange Function = 400



                           Determinant = 0

                    Matrix has 10 Zero Eigenvalue(s)
```

The linear and nonlinear constraints are satisfied at the solution.

Output 5.7.7. Linear and Nonlinear Constraints at the Solution

```
                 Linear Constraints Evaluated at Solution

 1 ACT          0  =         0  +    1.0000 * amounta  +    1.0000 * amountb
   -    1.0000 * pooltox  -    1.0000 * pooltoy
 2 ACT -4.481E-17  =         0  -    1.0000 * amountx  +    1.0000 * pooltox
   +    1.0000 * ctox
 3 ACT          0  =         0  -    1.0000 * amounty  +    1.0000 * pooltoy
   +    1.0000 * ctoy
 4 ACT          0  =         0  -    1.0000 * amountc  +    1.0000 * ctox
   +    1.0000 * ctoy


                    Values of Nonlinear Constraints

                                       Lagrange
   Constraint           Value Residual Multiplier

 [   5 ]    nlc3             0        0     6.0000 Active  NLEC
 [   6 ]    nlc1_G    4.04E-16 4.04E-16          . Active  NLIC     LinDep
 [   7 ]    nlc2_G    -284E-16 -284E-16    -6.0000 Active  NLIC


              Linearly Dependent Active Boundary Constraints

                    Parameter      N    Kind

                    ctox           8      Lower BC
                    pools         10      Lower BC


       Linearly Dependent Gradients of Active Nonlinear Constraints

                         Parameter           N

                         nlc3                6
```

The same problem can be specified in many different ways. For example, the following specification uses an INEST= data set containing the values of the starting point and of the constants COST, COSTB, COSTC, COSTX, COSTY, CA, CB, CC, and CD:

```
data init1(type=est);
input _type_ $ amountx amounty amounta amountb amountc
               pooltox pooltoy ctox ctoy pools
               _rhs_ costa costb costc costx costy
                     ca cb cc cd;
   datalines;
parms  1 1 1 1 1 1 1 1 1
       . 6 16 10 9 15  2.5 1.5 2. 3.
;

proc nlp inest=init1 all;
   parms amountx amounty amounta amountb amountc
         pooltox pooltoy ctox ctoy pools;
   bounds 0 <= amountx amounty amounta amountb amountc,
             amountx <= 100,
             amounty <= 200,
          0 <= pooltox pooltoy ctox ctoy,
          1 <= pools <= 3;
   lincon amounta + amountb = pooltox + pooltoy,
          pooltox + ctox = amountx,
          pooltoy + ctoy = amounty,
          ctox + ctoy    = amountc;
   nlincon nlc1-nlc2 >= 0.,
           nlc3 = 0.;
   max f;
   f = costx * amountx + costy * amounty
       - costa * amounta - costb * amountb - costc * amountc;
   nlc1 = ca * amountx - pools * pooltox - cc * ctox;
   nlc2 = cb * amounty - pools * pooltoy - cc * ctoy;
   nlc3 = cd * amounta + amountb - pools * (amounta + amountb);
run;
```

The third specification uses an INEST= data set containing the boundary and linear constraints in addition to the values of the starting point and of the constants. This specification also writes the model specification into an OUTMOD= data set:

```
data init2(type=est);
   input _type_ $ amountx amounty amounta amountb amountc
                  pooltox pooltoy ctox ctoy pools
                  _rhs_ costa costb costc costx costy;
   datalines;
parms      1   1  1  1  1    1   1   1  1  1
           .   6 16 10  9   15 2.5 1.5  2  3
lowerbd    0   0  0  0  0    0   0   0  0  1
           .   .  .  .  .    .   .   .  .  .
upperbd  100 200  .  .  .    .   .   .  .  3
           .   .  .  .  .    .   .   .  .  .
```

```
eq          .   .  1  1  .   -1  -1   .  .  .
            0   .  .  .  .    .   .   .  .  .
eq          1   .  .  .  .   -1   .  -1  .  .
            0   .  .  .  .    .   .   .  .  .
eq          .   1  .  .  .    .  -1   . -1  .
            0   .  .  .  .    .   .   .  .  .
eq          .   .  .  .  1    .   .  -1 -1  .
            0   .  .  .  .    .   .   .  .  .
;

proc nlp inest=init2 outmod=model all;
   parms amountx amounty amounta amountb amountc
         pooltox pooltoy ctox ctoy pools;
   nlincon nlc1-nlc2 >= 0.,
           nlc3 = 0.;
   max f;
   f = costx * amountx + costy * amounty
       - costa * amounta - costb * amountb - costc * amountc;
   nlc1 = 2.5 * amountx - pools * pooltox - 2. * ctox;
   nlc2 = 1.5 * amounty - pools * pooltoy - 2. * ctoy;
   nlc3 = 3 * amounta + amountb - pools * (amounta + amountb);
run;
```

The fourth specification not only reads the INEST=INIT2 data set, it also uses the model specification from the MODEL data set that was generated in the last specification. The PROC NLP call now contains only the defining variable statements:

```
proc nlp inest=init2 model=model all;
   parms amountx amounty amounta amountb amountc
         pooltox pooltoy ctox ctoy pools;
   nlincon nlc1-nlc2 >= 0.,
           nlc3 = 0.;
   max f;
run;
```

All four specifications start with the same starting point $amount(a)$, $amount(b)$, $amount(c)$, $amount(x)$, $amount(y)$, $pool\_to\_x$, $pool\_to\_y$, c_to_x, c_to_y, $pool\_s$ and generate the same results. However, there exist several local optima to this problem, as is pointed out in Liebman et al. (1986, p.130).

```
proc nlp inest=init2 model=model all;
   parms amountx amounty amounta amountb amountc
         pooltox pooltoy ctox ctoy = 0,
         pools = 2;
   nlincon nlc1-nlc2 >= 0.,
           nlc3 = 0.;
   max f;
run;
```

This starting point is accepted as a local solution with $profit = 0$, which, however, minimizes the profit.

Example 5.8. Chemical Equilibrium

The following example is used in many test libraries for nonlinear programming and was taken originally from Bracken and McCormick (1968).

The problem is to determine the composition of a mixture of various chemicals satisfying its chemical equilibrium state. The second law of thermodynamics implies that a mixture of chemicals satisfies its chemical equilibrium state (at a constant temperature and pressure) when the free energy of the mixture is reduced to a minimum. Therefore the composition of the chemicals satisfying its chemical equilibrium state can be found by minimizing the function of the free energy of the mixture.

Notation:

- m number of chemical elements in the mixture
- n number of compounds in the mixture
- x_j number of moles for compound j, $j = 1, \ldots, n$
- $s = \sum_{i=1}^{n} x_j$ total number of moles in the mixture
- a_{ij} number of atoms of element i in a molecule of compound j
- b_i the atomic weight of element i in the mixture

Constraints for the Mixture:

- The number of moles cannot be negative,

$$x_j > 0, \quad j = 1, \ldots, n$$

- There are m mass balance relationships,

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m$$

Objective Function: Total Free Energy of Mixture

$$f(x) = \sum_{j=1}^{n} x_j [c_j + ln(\frac{x_j}{s})]$$

with

$$c_j = (\frac{F^0}{RT})_j + lnP$$

where $\frac{F^0}{RT}$ is the model standard free energy function for the jth compound (that is found in tables) and P is the total pressure in atmospheres.

Minimization Problem:

Determine the parameters x_j that minimize the objective function $f(x)$ subject to the nonnegativity and linear balance constraints.

Numeric Example:

Determine the equilibrium composition of compound $\frac{1}{2}N_2H_4 + \frac{1}{2}O_2$ at temperature $T = 3500^oK$ and pressure $P = 750psi$.

				a_{ij}		
				i=1	i=2	i=3
j	Compound	$(F^0/RT)_j$	c_j	H	N	O
1	H	-10.021	-6.089	1		
2	H_2	-21.096	-17.164	2		
3	H_2O	-37.986	-34.054	2		1
4	N	-9.846	-5.914		1	
5	N_2	-28.653	-24.721		2	
6	NH	-18.918	-14.986	1	1	
7	NO	-28.032	-24.100		1	1
8	O	-14.640	-10.708			1
9	O_2	-30.594	-26.662			2
10	OH	-26.111	-22.179	1		1

Example Specification:

```
proc nlp tech=tr pall;
   array c[10] -6.089 -17.164 -34.054  -5.914 -24.721
               -14.986 -24.100 -10.708 -26.662 -22.179;
   array x[10] x1-x10;
   min y;
   parms x1-x10 = .1;
   bounds  1.e-6 <= x1-x10;
   lincon 2. = x1 + 2. * x2 + 2. * x3 + x6 + x10,
          1. = x4 + 2. * x5 + x6 + x7,
          1. = x3 + x7  + x8 + 2. * x9 + x10;
   s = x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10;
   y = 0.;
   do j = 1 to 10;
   y = y + x[j] * (c[j] + log(x[j] / s));
   end;
run;
```

Displayed Output:

The iteration history does not show any problems.

Output 5.8.1. Iteration History

```
                        Trust Region Optimization

                        Without Parameter Scaling


                                                                 Max Abs            Trust
          Rest     Func       Act    Objective  Obj Fun Gradient           Region
 Iter     arts    Calls       Con     Function   Change  Element  Lambda  Radius

    1        0        2         3'   -47.33412   2.2790   6.0765   2.456   1.000
    2        0        3         3'   -47.70043   0.3663   8.5592   0.908   0.418
    3        0        4         3    -47.73074   0.0303   6.4942       0   0.359
    4        0        5         3    -47.73275  0.00201   4.7606       0   0.118
    5        0        6         3    -47.73554  0.00279   3.2125       0  0.0168
    6        0        7         3    -47.74223  0.00669   1.9552   110.6 0.00271
    7        0        8         3    -47.75048  0.00825   1.1157   102.9 0.00563
    8        0        9         3    -47.75876  0.00828   0.4165   3.787  0.0116
    9        0       10         3    -47.76101  0.00224   0.0716       0  0.0121
   10        0       11         3    -47.76109 0.000083  0.00238       0  0.0111
   11        0       12         3    -47.76109 9.609E-8 2.733E-6       0 0.00248

                           Optimization Results

Iterations                           11  Function Calls                     13
Hessian Calls                        12  Active Constraints                  3
Objective Function         -47.76109086  Max Abs Gradient Element   1.8637499E-6
Lambda                                0  Actual Over Pred Change             0
Radius                     0.0024776027

GCONV convergence criterion satisfied.
```

The output lists the optimal parameters with the gradient.

Output 5.8.2. Optimization Results

```
                    PROC NLP: Nonlinear Minimization

                          Optimization Results
                           Parameter Estimates
                                                  Gradient
                                                 Objective
                N Parameter         Estimate       Function

                1 x1                0.040668      -9.785055
                2 x2                0.147730     -19.570110
                3 x3                0.783153     -34.792170
                4 x4                0.001414     -12.968921
                5 x5                0.485247     -25.937841
                6 x6                0.000693     -22.753976
                7 x7                0.027399     -28.190984
                8 x8                0.017947     -15.222060
                9 x9                0.037314     -30.444120
               10 x10               0.096871     -25.007115

               Value of Objective Function = -47.76109086
```

The three equality constraints are satisfied at the solution.

Output 5.8.3. Linear Constraints at Solution

```
                    PROC NLP: Nonlinear Minimization

                Linear Constraints Evaluated at Solution

 1 ACT -3.608E-16  =    2.0000  -    1.0000 * x1   -    2.0000 * x2   -
     2.0000 * x3   -    1.0000 * x6   -    1.0000 * x10
 2 ACT 2.2204E-16  =    1.0000  -    1.0000 * x4   -    2.0000 * x5   -
     1.0000 * x6   -    1.0000 * x7
 3 ACT -1.943E-16  =    1.0000  -    1.0000 * x3   -    1.0000 * x7   -
     1.0000 * x8   -    2.0000 * x9   -    1.0000 * x10
```

The Lagrange multipliers and the projected gradient are displayed also.

Output 5.8.4. Lagrange Multipliers

```
                    PROC NLP: Nonlinear Minimization

                    First Order Lagrange Multipliers


                                                  Lagrange
                       Active Constraint         Multiplier

                  Linear EC      [1]              9.785055
                  Linear EC      [2]             12.968921
                  Linear EC      [3]             15.222060
```

The elements of the projected gradient must be small to satisfy a necessary first-order optimality condition.

Output 5.8.5. Projected Gradient

```
                    PROC NLP: Nonlinear Minimization

                          Projected Gradient

                              Free    Projected
                         Dimension     Gradient

                                 1 4.5770108E-9
                                 2 6.868355E-10
                                 3 -7.283013E-9
                                 4 -0.000001864
                                 5 -0.000001434
                                 6 -0.000001361
                                 7 -0.000000294
```

The projected Hessian matrix is positive definite satisfying the second-order optimality condition.

Output 5.8.6. Projected Hessian Matrix

Projected Hessian Matrix

	X1	X2	X3	X4
X1	20.903196985	-0.122067474	2.6480263467	3.3439156526
X2	-0.122067474	565.97299938	106.54631863	-83.7084843
X3	2.6480263467	106.54631863	1052.3567179	-115.230587
X4	3.3439156526	-83.7084843	-115.230587	37.529977667
X5	-1.373829641	-37.43971036	182.89278895	-4.621642366
X6	-1.491808185	-36.20703737	175.97949593	-4.574152161
X7	1.1462413516	-16.635529	-57.04158208	10.306551561

Projected Hessian Matrix

	X5	X6	X7
X1	-1.373829641	-1.491808185	1.1462413516
X2	-37.43971036	-36.20703737	-16.635529
X3	182.89278895	175.97949593	-57.04158208
X4	-4.621642366	-4.574152161	10.306551561
X5	79.326057844	22.960487404	-12.69831637
X6	22.960487404	66.669897023	-8.121228758
X7	-12.69831637	-8.121228758	14.690478023

The following PROC NLP call uses a specified analytical gradient and the Hessian matrix is computed by finite difference approximations based on the analytic gradient:

```
proc nlp tech=tr fdhessian all;
   array c[10] -6.089 -17.164 -34.054  -5.914 -24.721
               -14.986 -24.100 -10.708 -26.662 -22.179;
   array x[10] x1-x10;
   array g[10] g1-g10;
   min y;
   parms x1-x10 = .1;
   bounds  1.e-6 <= x1-x10;
   lincon 2. = x1 + 2. * x2 + 2. * x3 + x6 + x10,
          1. = x4 + 2. * x5 + x6 + x7,
          1. = x3 + x7  + x8 + 2. * x9 + x10;
   s = x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10;
   y = 0.;
   do j = 1 to 10;
      y = y + x[j] * (c[j] + log(x[j] / s));
      g[j] = c[j] + log(x[j] / s);
   end;
run;
```

The results are almost identical to those of the former run.

Example 5.9. Minimize Total Delay in a Network

The following example is taken from the user's guide of GINO (Liebman et al. 1986). A simple network of five roads (arcs) can be illustrated by the path diagram:

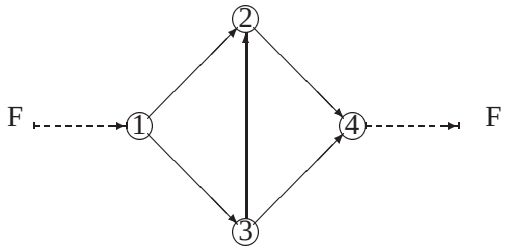

Figure 12: Simple Road Network

The five roads connect four intersections illustrated by numbered nodes. Each minute F vehicles enter and leave the network. arc_{ij} refers to the road from intersection i to intersection j, and the parameter x_{ij} refers to the flow from i to j. The law that traffic flowing into each intersection j must also flow out is described by the linear equality constraint

$$\sum_i x_{ij} = \sum_i x_{ji} \quad , \quad j = 1, \ldots, n$$

In general, roads also have an upper capacity, that is the number of vehicles which can be handled per minute. The upper limits c_{ij} can be enforced by boundary constraints

$$0 \leq x_{ij} \leq c_{ij} \quad , \quad i, j = 1, \ldots, n$$

Finding the maximum flow through a network is equivalent to solving a simple linear optimization problem, and for large problems, PROC LP or PROC NETFLOW can be used. The objective function is

$$\max f = x_{24} + x_{34}$$

and the constraints are

$$0 \leq x_{12}, x_{32}, x_{34} \leq 10$$

$$0 \leq x_{13}, x_{24} \leq 30$$

$$x_{13} = x_{32} + x_{34}$$

$$x_{12} + x_{32} = x_{24}$$

$$x_{12} + x_{13} = x_{24} + x_{34}$$

The three linear equality constraints are linearly dependent. One of them is deleted automatically by the PROC NLP subroutines. Even though the default technique is used for this small example any optimization subroutine can be used.

```
proc nlp all initial=.5;
   max y;
   parms x12 x13 x32 x24 x34;
   bounds x12 <= 10,
          x13 <= 30,
          x32 <= 10,
          x24 <= 30,
          x34 <= 10;
   /* what flows into an intersection must flow out */
   lincon x13 = x32 + x34,
          x12 + x32 = x24,
          x24 + x34 = x12 + x13;
   y = x24 + x34 + 0*x12 + 0*x13 + 0*x32;
run;
```

The optimal solution follows.

Output 5.9.1. Iteration History

Newton-Raphson Ridge Optimization

Without Parameter Scaling

Iter	Rest arts	Func Calls	Act Con	Objective Function	Obj Fun Change	Max Abs Gradient Element	Ridge	Actual Over Pred Change
1*	0	2	4	20.25000	19.2500	0.5774	0.0313	0.860
2*	0	3	5	30.00000	9.7500	0	0.0313	1.683

Optimization Results

Iterations	2	Function Calls	4
Hessian Calls	3	Active Constraints	5
Objective Function	30	Max Abs Gradient Element	0
Ridge	0	Actual Over Pred Change	1.6834532374

All parameters are actively constrained. Optimization cannot proceed.

Output 5.9.2. Optimization Results

PROC NLP: Nonlinear Maximization

Optimization Results
Parameter Estimates

N	Parameter	Estimate	Gradient Objective Function	Active Bound Constraint
1	x12	10.000000	0	Upper BC
2	x13	20.000000	0	
3	x32	10.000000	0	Upper BC
4	x24	20.000000	1.000000	
5	x34	10.000000	1.000000	Upper BC

Value of Objective Function = 30

Finding a traffic pattern that minimizes the total delay to move F vehicles per minute from node 1 to node 4 introduces nonlinearities that, in turn, demand nonlinear optimization techniques. As traffic volume increases, speed decreases. Let t_{ij} be the travel time on arc_{ij} and assume that the following formulas describe the travel time as decreasing functions of the amount of traffic:

$$t_{12} = 5 + 0.1x_{12}/(1 - x_{12}/10)$$

$$t_{13} = x_{13}/(1 - x_{13}/30)$$

$$t_{32} = 1 + x_{32}/(1 - x_{32}/10)$$

$$t_{24} = x_{24}/(1 - x_{24}/30)$$

$$t_{34} = 5 + .1 * x_{34}/(1 - x_{34}/10)$$

These formulas use the road capacities (upper bounds), assuming $F = 5$ vehicles per minute have to be moved through the network. The objective function is now

$$\min f = t_{12}x_{12} + t_{13}x_{13} + t_{32}x_{32} + t_{24}x_{24} + t_{34}x_{34}$$

and the constraints are.

$$0 \leq x_{12}, x_{32}, x_{34} \leq 10$$

$$0 \leq x_{13}, x_{24} \leq 30$$

$$x_{13} = x_{32} + x_{34}$$

$$x_{12} + x_{32} = x_{24}$$

$$x_{24} + x_{34} = F = 5$$

Again, just for variety, the default algorithm is used:

```
proc nlp all initial=.5;
   min y;
   parms x12 x13 x32 x24 x34;
   bounds x12 x13 x32 x24 x34 >= 0;
   lincon x13 = x32 + x34,  /* flow in = flow out */
          x12 + x32 = x24,
```

```
        x24 + x34 = 5;    /* = f = desired flow */
   t12 = 5 + .1 * x12 / (1 - x12 / 10);
   t13 = x13 / (1 - x13 / 30);
   t32 = 1 + x32 / (1 - x32 / 10);
   t24 = x24 / (1 - x24 / 30);
   t34 = 5 + .1 * x34 / (1 - x34 / 10);
   y = t12*x12 + t13*x13 + t32*x32 + t24*x24 + t34*x34;
run;
```

The optimal solution follows.

Output 5.9.3. Iteration History

```
                  Newton-Raphson Ridge Optimization

                      Without Parameter Scaling


                                                                    Actual
                                                 Max Abs             Over
         Rest    Func       Act    Objective  Obj Fun Gradient            Pred
 Iter     arts   Calls       Con     Function   Change  Element  Ridge   Change

    1        0       2         4     40.30303   0.3433 4.44E-16      0    0.508

                          Optimization Results

Iterations                           1  Function Calls                     3
Hessian Calls                        2  Active Constraints                 4
Objective Function         40.303030303  Max Abs Gradient Element   4.440892E-16
Ridge                                0  Actual Over Pred Change    0.5083585587

ABSGCONV convergence criterion satisfied.
```

Output 5.9.4. Opimization Results

```
                  PROC NLP: Nonlinear Minimization

                       Optimization Results
                        Parameter Estimates
                                          Gradient    Active
                                         Objective    Bound
          N Parameter        Estimate       Function    Constraint

          1 x12               2.500000       5.777778
          2 x13               2.500000       5.702479
          3 x32           2.775558E-17       1.000000     Lower BC
          4 x24               2.500000       5.702479
          5 x34               2.500000       5.777778

                 Value of Objective Function = 40.303030303
```

The active constraints and corresponding Lagrange multiplier estimates (costs) are as follows.

Output 5.9.5. Linear Constraints at Solution

```
                  PROC NLP: Nonlinear Minimization

               Linear Constraints Evaluated at Solution

 1 ACT          0  =         0  +    1.0000 * x13  -    1.0000 * x32  -
     1.0000 * x34
 2 ACT 4.4409E-16  =         0  +    1.0000 * x12  +    1.0000 * x32  -
     1.0000 * x24
 3 ACT          0  =   -5.0000  +    1.0000 * x24  +    1.0000 * x34
```

Output 5.9.6. Lagrange Multipliers at Solution

```
                    First Order Lagrange Multipliers


                                                      Lagrange
                       Active Constraint           Multiplier

                  Lower BC       x32                 0.924702
                  Linear EC      [1]                 5.702479
                  Linear EC      [2]                 5.777778
                  Linear EC      [3]                11.480257
```

The projected gradient is very small, satisfying the first-order optimality criterion.

Output 5.9.7. Projected Gradient at Solution

```
                          Projected Gradient

                             Free    Projected
                        Dimension     Gradient

                                1 4.440892E-16
```

The projected Hessian matrix is positive definite, satisfying the second-order optimality criterion:

Output 5.9.8. Projected Hessian at Solution

```
                          Projected Hessian
                                Matrix

                                             X1

                         X1       1.535309013
```

References

Abramowitz, M. and Stegun, I.A. (1972), *Handbook of Mathematical Functions*, New-York: Dover Publications, Inc.

Al-Baali, M. and Fletcher, R. (1985), "Variational Methods for Nonlinear Least-Squares" *J. Oper. Res. Soc.*, 36, 405-421.

Al-Baali, M. and Fletcher, R. (1986), "An Efficient Line-Search for Nonlinear Least-Squares", *J. Optimiz. Theory Appl.*, 48, 359-377.

Bard, Y. (1974), *Nonlinear Parameter Estimation*, New-York: Academic Press.

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

Betts, J. T. (1977), "An Accelerated Multiplier Method for Nonlinear Programming", *Journal of Optimization Theory and Applications*, 21, 137-174.

Bracken, J. and McCormick, G.P. (1968), *Selected Applications of Nonlinear Programming*, New York: John Wiley and Sons, Inc.

Chamberlain, R.M.; Powell, M.J.D.; Lemarechal, C.; and Pedersen, H.C. (1982), "The Watchdog Technique for Forcing Convergence in Algorithms for Constrained Optimization", *Mathematical Programming*, 16, 1-17.

Cramer, J. S. (1986), *Econometric Applications of Maximum Likelihood Methods*, Cambridge, England: Cambridge University Press.

Dennis, J.E., Gay, D.M. and Welsch, R.E. (1981), "An Adaptive Nonlinear Least-Squares Algorithm", *ACM Trans. Math. Software*, 7, 348-368.

Dennis, J.E. and Mei, H.H.W. (1979), "Two New Unconstrained Optimization Algorithms that use Function and Gradient Values", *J. Optim. Theory Appl.*, 28, 453-482.

Dennis, J.E. and Schnabel, R.B. (1983), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, New Jersey: Prentice-Hall.

Fletcher, R. (1987), *Practical Methods of Optimization.* second edition, Chichester: John Wiley and Sons.

Fletcher, R. and Powell, M.J.D. (1963), "A Rapidly Convergent Descent Method for Minimization", *Comput.J.*, 6, 163-168.

Fletcher, R. and Xu, C. (1987), "Hybrid Methods for Nonlinear Least-Squares", *J. Numerical Analysis*, 7, 371-389.

Gallant, A.R. (1987), *Nonlinear Statistical Models*, New York: John Wiley and Sons.

Gay, D.M. (1983), "Subroutines for Unconstrained Minimization", *ACM Trans. Math. Software*, 9, 503-524.

George, J.A. and Liu, J.W. (1981), *Computer Solution of Large Sparse Positive Definite Systems*, New Jersey: Prentice-Hall.

Gill, E.P.; Murray, W.; and Wright, M.H. (1981), *Practical Optimization*, London: Academic Press.

Gill, E.P.; Murray, W.; Saunders; M.A. and Wright, M.H. (1983), "Computing Forward-Difference Intervals for Numerical Optimization," *SIAM J. Sci. Stat. Comput.*, 4, 310-321.

Gill, E.P.; Murray, W.; Saunders, M.A.; and Wright, M.H. (1984), "Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints", *ACM Trans. Math. Software*, 10, 282-298.

Goldfeld, S.M.; Quandt, R.E.; and Trotter, H.F. (1966), "Maximisation by Quadratic Hill-Climbing", *Econometrica*, 34, 541-551.

Hambleton, R.K.; Swaminathan, H.; and Rogers, H.J. (1991), *Fundamentals of Item Response Theory*, Newbury Park, CA: Sage Publications.

Hartmann, W. (1992), *Nonlinear Optimization in IML*, Releases 6.08, 6.09, 6.10; Technical Report, Cary, N.C.: SAS Institute Inc.

Hartmann, W. (1992), *Applications of Nonlinear Optimization with PROC NLP and SAS/IML Software*, Technical Report, Cary, N.C.: SAS Institute Inc.

Haverly, C.A. (1978), "Studies of the Behavior of Recursion for the Pooling Problem," *SIGMAP Bulletin*, Association for Computing Machinery.

Hock, W. and Schittkowski, K. (1981), *s for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems 187, Berlin-Heidelberg-New York: Springer Verlag.

Jennrich, R.I. and Sampson, P.F. (1968), "Application of Stepwise Regression to Nonlinear Estimation", *Technometrics*, 10, 63-72.

Lawless, J.F. (1982), "Statistical Methods and Methods for Lifetime Data", New York: John Wiley and Sons, Inc.

Liebman, J.; Lasdon, L.; Schrage, L.; and Waren, A. (1986), *Modeling and Optimization with GINO*, California: The Scientific Press.

Lindström, P. and Wedin, P.A. (1984), "A New Line-Search Algorithm for Nonlinear Least-Squares Problems", *Math. Prog.*, 29, 268-296.

Moré, J.J. (1978), "The Levenberg-Marquardt Algorithm: Implementation and Theory", in: G.A. Watson (ed.), *Lecture Notes in Mathematics 630*, Berlin-Heidelberg-New York: Springer Verlag, 105-116.

Moré, J.J., Garbow, B.S. and Hillstrom, K.E. (1981), "Testing Unconstrained Optimization Software", *ACM Trans. Math. Software*, 7, 17-41.

Moré, J.J. and Sorensen, D.C. (1983), "Computing a Trust-Region Step", *SIAM J. Sci. Stat. Comput.*, 4, 553-572.

Moré, J.J. and Sorensen, D.C. (1983), "Computing a Trust-Region Step", *SIAM J. Sci. Stat. Comput.*, 4, 553-572.

Moré, J.J. and Wright, S.J. (1993), *Optimization Software Guide*, Philadelphia: SIAM.

Murtagh, B.A. and Saunders, M.A. (1983), *MINOS 5.0 User's Guide*; Technical Report SOL 83-20, Stanford University.

Nelder, J.A. and Mead, R. (1965), "A Simplex Method for Function Minimization", *Comput. J.*, 7, 308-313.

Polak, E. (1971), *Computational Methods in Optimization*, New York - San Francisco - London: 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, M.J.D. (1982a), "Extensions to subroutine VF02AD", in *Systems Modeling and Optimization, Lecture Notes in Control and Information Sciences* 38, eds. 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.

Powell, J.M.D. (1992), "A Direct Search Optimization Method that Models the Objective and Constraint Functions by Linear Interpolation", *DAMTP/NA5*, Cambridge, England.

Rosenbrock, H.H. (1960), "An Automatic Method for Finding the Greatest or Least Value of a Function", *Comput. J.*, 3, 175-184.

Schittkowski, K. (1980), *Nonlinear Programming Codes - Information, Tests, Performance* Lecture Notes in Economics and Mathematical Systems 183, Berlin-Heidelberg-New York: Springer Verlag.

Schittkowski, K. (1987), *More s for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems 282, Berlin-Heidelberg-New York: Springer Verlag.

Schittkowski, K. and Stoer, J. (1979), "A Factorization Method for the Solution of Constrained Linear Least-Squares Problems Allowing Subsequent Data Changes", *Numer. Math.*, 31, 431-463.

Stewart, G.W. (1967), "A Modification of Davidon's Minimization method to Accept Difference Approximations of Derivatives", *J. Assoc. Comput. Mach.* 14, 72-83.

Wedin, P.A. and Lindström, P. (1987), *Methods and Software for Nonlinear Least-Squares Problems*, University of Umea, Report No. UMINF 133.87.

Whitaker,D.; Triggs; C.M.; and John, J.A. (1990), "Construction of Block Designs using Mathematical Programming", *J. R. Statist. Soc. B*, 52, 497-503.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/OR® User's Guide: Mathematical Programming, Version 8*, Cary, NC: SAS Institute Inc., 1999. 566 pp.

SAS/OR® User's Guide: Mathematical Programming, Version 8

ISBN 1–58025–491–8

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

1st printing, October 1999