Chapter 11
Nonlinear Optimization Examples

Chapter Table of Contents

Chapter 11

Nonlinear Optimization Examples

Overview

The IML procedure offers a set of optimization subroutines for minimizing or maximizing a continuous nonlinear function $f = f(x)$ of n parameters, where $x = (x_1, \ldots, x_n)^T$. The parameters can be subject to boundary constraints and linear or nonlinear equality and inequality constraints. The following set of optimization subroutines is available:

NLPCG	Conjugate Gradient Method
NLPDD	Double Dogleg Method
NLPNMS	Nelder-Mead Simplex Method
NLPNRA	Newton-Raphson Method
NLPNRR	Newton-Raphson Ridge Method
NLPQN	(Dual) Quasi-Newton Method
NLPQUA	Quadratic Optimization Method
NLPTR	Trust-Region Method

The following subroutines are provided for solving nonlinear least-squares problems:

NLPLM	Levenberg-Marquardt Least-Squares Method
NLPHQN	Hybrid Quasi-Newton Least-Squares Methods

A least-squares problem is a special form of minimization problem where the objective function is defined as a sum of squares of other (nonlinear) functions.

$$f(x) = \frac{1}{2}\{f_1^2(x) + \cdots + f_m^2(x)\}$$

Least-squares problems can usually be solved more efficiently by the least-squares subroutines than by the other optimization subroutines.

The following subroutines are provided for the related problems of computing finite difference approximations for first- and second-order derivatives and of determining a feasible point subject to boundary and linear constraints:

NLPFDD	Approximate Derivatives by Finite Differences
NLPFEA	Feasible Point Subject to Constraints

Each optimization subroutine works iteratively. If the parameters are subject only to linear constraints, all optimization and least-squares techniques are *feasible-point methods*; that is, they move from feasible point $x^{(k)}$ to a better feasible point $x^{(k+1)}$ by a step in the search direction $s^{(k)}$, $k = 1, 2, 3, \ldots$. If you do not provide a feasible starting point $x^{(0)}$, the optimization methods call the algorithm used in the NLPFEA subroutine, which tries to compute a starting point that is feasible with respect to the boundary and linear constraints.

The NLPNMS and NLPQN subroutines permit nonlinear constraints on parameters. For problems with nonlinear constraints, these subroutines do not use a feasible-point method; instead, the algorithms begin with whatever starting point you specify, whether feasible or infeasible.

Each optimization technique requires a continuous objective function $f = f(x)$ and all optimization subroutines except the NLPNMS subroutine require continuous first-order derivatives of the objective function f. If you do not provide the derivatives of f, they are approximated by finite difference formulas. You can use the NLPFDD subroutine to check the correctness of analytical derivative specifications.

Most of the results obtained from the IML procedure optimization and least-squares subroutines can also be obtained by using the NLP procedure in the SAS/OR product.

The advantages of the IML procedure are as follows:

- You can use matrix algebra to specify the objective function, nonlinear constraints, and their derivatives in IML modules.
- The IML procedure offers several subroutines that can be used to specify the objective function or nonlinear constraints, many of which would be very difficult to write for the NLP procedure.
- You can formulate your own termination criteria by using the *"ptit"* module argument.

The advantages of the NLP procedure are as follows:

- Although identical optimization algorithms are used, the NLP procedure can be much faster because of the interactive and more general nature of the IML product.
- Analytic first- and second-order derivatives can be computed with a special compiler.
- Additional optimization methods are available in the NLP procedure that do not fit into the framework of this package.
- Data set processing is much easier than in the IML procedure. You can save results in output data sets and use them in following runs.
- The printed output contains more information.

Getting Started

Unconstrained Rosenbrock Function

The Rosenbrock function is defined as

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

The minimum function value $f^* = f(x^*) = 0$ is at the point $x^* = (1, 1)$.

The following code calls the NLPTR subroutine to solve the optimization problem:

```
proc iml;
   title 'Test of NLPTR subroutine: Gradient Specified';
   start F_ROSEN(x);
      y1 = 10. * (x[2] - x[1] * x[1]);
      y2 = 1. - x[1];
      f  = .5 * (y1 * y1 + y2 * y2);
      return(f);
   finish F_ROSEN;

   start G_ROSEN(x);
      g = j(1,2,0.);
      g[1] = -200.*x[1]*(x[2]-x[1]*x[1]) - (1.-x[1]);
      g[2] =  100.*(x[2]-x[1]*x[1]);
      return(g);
   finish G_ROSEN;

   x = {-1.2 1.};
   optn = {0 2};
   call nlptr(rc,xres,"F_ROSEN",x,optn) grd="G_ROSEN";
   quit;
```

The NLPTR is a trust-region optimization method. The F_ROSEN module represents the Rosenbrock function, and the G_ROSEN module represents its gradient. Specifying the gradient can reduce the number of function calls by the optimization subroutine. The optimization begins at the initial point $x = (-1.2, 1)$. For more information on the NLPTR subroutine and its arguments, see the section “NLPTR Call” on page 667. For details on the options vector, which is given by the OPTN vector in the preceding code, see the section “Options Vector” on page 319.

A portion of the output produced by the NLPTR subroutine is shown in Figure 11.1 on page 300.

```
                      Trust Region Optimization

                     Without Parameter Scaling
            CRP Jacobian Computed by Finite Differences

                Parameter Estimates               2

                         Optimization Start

Active Constraints             0  Objective Function           12.1
Max Abs Gradient           107.8  Radius                          1
Element


                                            Max Abs          Trust
       Rest  Func Act   Objective   Obj Fun Gradient          Region
 Iter  arts Calls Con    Function    Change  Element  Lambda  Radius

    1     0     2   0     2.36594    9.7341   2.3189       0   1.000
    2     0     5   0     2.05926    0.3067   5.2875   0.385   1.526
    3     0     8   0     1.74390    0.3154   5.9934       0   1.086
    .
    .
    .

   22     0    31   0  1.3128E-16 6.96E-10 1.977E-7       0 0.00314

                        Optimization Results

Iterations                    22  Function Calls                32
Hessian Calls                 23  Active Constraints             0
Objective Function  1.312814E-16  Max Abs Gradient     1.9773384E-7
                                  Element
Lambda                         0  Actual Over Pred               0
                                  Change
Radius               0.003140192

ABSGCONV convergence criterion satisfied.

                        Optimization Results
                         Parameter Estimates
                                               Gradient
                                              Objective
             N Parameter         Estimate      Function

             1 X1                1.000000     0.000000198
             2 X2                1.000000    -0.000000105

             Value of Objective Function = 1.312814E-16
```

Figure 11.1. NLPTR Solution to the Rosenbrock Problem

Since $f(x) = \frac{1}{2}\{f_1^2(x) + f_2^2(x)\}$, you can also use least-squares techniques in this situation. The following code calls the NLPLM subroutine to solve the problem. The output is shown in Figure **??** on page **??**.

```
proc iml;
   title 'Test of NLPLM subroutine: No Derivatives';
   start F_ROSEN(x);
     y = j(1,2,0.);
     y[1] = 10. * (x[2] - x[1] * x[1]);
     y[2] = 1. - x[1];
    return(y);
```

```
      finish F_ROSEN;

      x = {-1.2 1.};
      optn = {2 2};
      call nlplm(rc,xres,"F_ROSEN",x,optn);
      quit;
```

The Levenberg-Marquardt least-squares method, which is the method used by the NLPLM subroutine, is a modification of the trust-region method for nonlinear least-squares problems. The F_ROSEN module represents the Rosenbrock function. Note that for least-squares problems, the m functions $f_1(x), \ldots, f_m(x)$ are specified as elements of a vector; this is different from the manner in which $f(x)$ is specified for the other optimization techniques. No derivatives are specified in the preceding code, so the NLPLM subroutine computes finite difference approximations. For more information on the NLPLM subroutine, see the section "NLPLM Call" on page 645.

Constrained Betts Function

The linearly constrained Betts function (Hock & Schittkowski 1981) is defined as

$$f(x) = 0.01x_1^2 + x_2^2 - 100$$

with boundary constraints

$$\begin{aligned} 2 \leq & \quad x_1 \quad \leq 50 \\ -50 \leq & \quad x_2 \quad \leq 50 \end{aligned}$$

and linear constraint

$$10x_1 - x_2 \geq 10$$

The following code calls the NLPCG subroutine to solve the optimization problem. The infeasible initial point $x^0 = (-1, -1)$ is specified, and a portion of the output is shown in Figure 11.2.

```
   proc iml;
      title 'Test of NLPCG subroutine: No Derivatives';
      start F_BETTS(x);
         f = .01 * x[1] * x[1] + x[2] * x[2] - 100.;
         return(f);
      finish F_BETTS;

      con = {  2. -50.  .   .,
              50.  50.  .   .,
              10.  -1. 1. 10.};
      x = {-1. -1.};
      optn = {0 2};
      call nlpcg(rc,xres,"F_BETTS",x,optn,con);
      quit;
```

The NLPCG subroutine performs conjugate gradient optimization. It requires only function and gradient calls. The F_BETTS module represents the Betts function, and since no module is defined to specify the gradient, first-order derivatives are computed by finite difference approximations. For more information on the NLPCG subroutine, see the section “NLPCG Call” on page 634. For details on the constraint matrix, which is represented by the CON matrix in the preceding code, see the section “Parameter Constraints” on page 317.

```
NOTE: Initial point was changed to be feasible for boundary and
                      linear constraints.


                       Optimization Start
                       Parameter Estimates
                                       Gradient           Lower
                                      Objective           Bound
   N Parameter         Estimate        Function      Constraint

   1 X1                6.800000        0.136000        2.000000
   2 X2               -1.000000       -2.000000      -50.000000

                       Optimization Start
                       Parameter Estimates
                                 Upper
                                 Bound
                            Constraint

                             50.000000
                             50.000000

             Value of Objective Function = -98.5376



                       Linear Constraints

   1   59.00000 :      10.0000  <=   +   10.0000 * X1  -   1.0000 * X2


                Conjugate-Gradient Optimization

      Automatic Restart Update (Powell, 1977; Beale, 1972)
            Gradient Computed by Finite Differences

              Parameter Estimates               2
              Lower Bounds                      2
              Upper Bounds                      2
              Linear Constraints                1
```

Figure 11.2. NLPCG Solution to Betts Problem

```
                         Optimization Start

Active Constraints               0  Objective Function       -98.5376
Max Abs Gradient                 2
Element


                                            Max Abs           Slope
       Rest  Func Act   Objective  Obj Fun Gradient    Step  Search
 Iter  arts Calls Con    Function   Change  Element    Size   Direc

    1     0     3   0   -99.54682   1.0092   0.1346   0.502  -4.018
    2     1     7   1   -99.96000   0.4132  0.00272  34.985 -0.0182
    3     2     9   1   -99.96000 1.851E-6        0   0.500  -74E-7

                        Optimization Results

Iterations                       3  Function Calls                10
Gradient Calls                   9  Active Constraints             1
Objective Function          -99.96  Max Abs Gradient               0
                                    Element
Slope of Search      -7.398365E-6
Direction

ABSGCONV convergence criterion satisfied.


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

     1 X1                2.000000        0.040000     Lower BC
     2 X2            -1.24028E-10               0

                Value of Objective Function = -99.96



              Linear Constraints Evaluated at Solution

  [1]       10.00000  =  -10.0000  +   10.0000 * X1        -    1.0000 * X2
```

Figure 11.2. (continued)

Since the initial point $(-1, -1)$ is infeasible, the subroutine first computes a feasible starting point. Convergence is achieved after three iterations, and the optimal point is given to be $x^* = (2, 0)$ with an optimal function value of $f^* = f(x^*) = -99.96$. For more information on the printed output, see the section “Printing the Optimization History” on page 334.

Rosen-Suzuki Problem

The Rosen-Suzuki problem is a function of four variables with three nonlinear constraints on the variables. It is taken from Problem 43 of Hock and Schittkowski (1981). The objective function is

$$f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

The nonlinear constraints are

$$\begin{aligned}0 &\leq 8 - x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 \\ 0 &\leq 10 - x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 \\ 0 &\leq 5 - 2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4\end{aligned}$$

Since this problem has nonlinear constraints, only the NLPQN and NLPNMS subroutines are available to perform the optimization. The following code solves the problem with the NLPQN subroutine:

```
proc iml;
   start F_HS43(x);
      f = x*x‘ + x[3]*x[3] - 5*(x[1] + x[2]) - 21*x[3] + 7*x[4];
      return(f);
   finish F_HS43;
   start C_HS43(x);
      c = j(3,1,0.);
      c[1] = 8 - x*x‘ - x[1] + x[2] - x[3] + x[4];
      c[2] = 10 - x*x‘ - x[2]*x[2] - x[4]*x[4] + x[1] + x[4];
      c[3] = 5 - 2.*x[1]*x[1] - x[2]*x[2] - x[3]*x[3]
               - 2.*x[1] + x[2] + x[4];
      return(c);
   finish C_HS43;
   x = j(1,4,1);
   optn= j(1,11,.); optn[2]= 3; optn[10]= 3; optn[11]=0;
   call nlpqn(rc,xres,"F_HS43",x,optn) nlc="C_HS43";
```

The F_HS43 module specifies the objective function, and the C_HS43 module specifies the nonlinear constraints. The OPTN vector is passed to the subroutine as the *opt* input argument. See the section “Options Vector” on page 319 for more information. The value of OPTN[10] represents the total number of nonlinear constraints, and the value of OPTN[11] represents the number of equality constraints. In the preceding code, OPTN[10]=3 and OPTN[11]=0, which indicate that there are three constraints, all of which are inequality constraints. In the subroutine calls, instead of separating missing input arguments with commas, you can specify optional arguments with keywords, as in the CALL NLPQN statement in the preceding code. For details on the CALL NLPQN statement, see the section “NLPQN Call” on page 658.

The initial point for the optimization procedure is $x = (1, 1, 1, 1)$, and the optimal point is $x^* = (0, 1, 2, -1)$, with an optimal function value of $f(x^*) = -44$. Part of the output produced is shown in Figure 11.3 on page 305.

```
                   Dual Quasi-Newton Optimization

          Modified VMCWD Algorithm of Powell (1978, 1982)

     Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
             Lagrange Multiplier Update of Powell(1982)
              Gradient Computed by Finite Differences
   Jacobian Nonlinear Constraints Computed by Finite Differences

               Parameter Estimates                4
               Nonlinear Constraints              3

                         Optimization Start

Objective Function            -19  Maximum Constraint             0
                                   Violation
Maximum Gradient of            17
the Lagran Func


                                                            Maximum
                                  Maximum                      Grad
                                     Con-                   Element
                                  straint Predicted          of the
        Rest   Func   Objective    Viola-  Function    Step  Lagran
Iter    arts  Calls    Function      tion Reduction    Size    Func

   1       0      2   -41.88007    1.8988   13.6803   1.000   5.647
   2       0      3   -48.83264    3.0280    9.5464   1.000   5.041
   3       0      4   -45.33515    0.5452    2.6179   1.000   1.061
   4       0      5   -44.08667    0.0427    0.1732   1.000  0.0297
   5       0      6   -44.00011  0.000099  0.000218   1.000 0.00906
   6       0      7   -44.00001  2.573E-6  0.000014   1.000 0.00219
   7       0      8   -44.00000  9.118E-8  5.097E-7   1.000 0.00022
```

Figure 11.3. Solution to the Rosen-Suzuki Problem by the NLPQN Subroutine

```
                        Optimization Results

Iterations                        7  Function Calls                  9
Gradient Calls                    9  Active Constraints              2
Objective Function   -44.00000026  Maximum Constraint   9.1176306E-8
                                   Violation
Maximum Projected    0.0002265341  Value Lagrange                -44
Gradient                           Function
Maximum Gradient of    0.00022158  Slope of Search      -5.097332E-7
the Lagran Func                    Direction

FCONV2 convergence criterion satisfied.

WARNING: The point x is feasible only at the LCEPSILON= 1E-7 range.


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

     1 X1            -0.000001248       -5.000002    -0.000012804
     2 X2                1.000027       -2.999945        0.000222
     3 X3                1.999993      -13.000027    -0.000054166
     4 X4               -1.000003        4.999995    -0.000020681

             Value of Objective Function = -44.00000026


                  Value of Lagrange Function = -44
```

Figure 11.3. (continued)

In addition to the standard iteration history, the NLPQN subroutine includes the following information for problems with nonlinear constraints:

- *conmax* is the maximum value of all constraint violations.
- *pred* is the value of the predicted function reduction used with the GTOL and FTOL2 termination criteria.
- *alfa* is the step size α of the quasi-Newton step.
- *lfgmax* is the maximum element of the gradient of the Lagrange function.

Details

Global Versus Local Optima

All the IML optimization algorithms converge toward local rather than global optima. The smallest local minimum of an objective function is called the global minimum, and the largest local maximum of an objective function is called the global maximum. Hence, the subroutines may occasionally fail to find the global optimum. Suppose you have the function, $f(x) = \frac{1}{27}(3x_1^4 - 28x_1^3 + 84x_1^2 - 96x_1 + 64) + x_2^2$, which has a local minimum at $f(1,0) = 1$ and a global minimum at the point $f(4,0) = 0$.

The following statements use two calls of the NLPTR subroutine to minimize the preceding function. The first call specifies the initial point $xa = (0.5, 1.5)$, and the second call specifies the initial point $xb = (3, 1)$. The first call finds the local optimum $x^* = (1, 0)$, and the second call finds the global optimum $x^* = (4, 0)$.

```
proc iml;
   start F_GLOBAL(x);
      f=(3*x[1]**4-28*x[1]**3+84*x[1]**2-96*x[1]+64)/27 + x[2]**2;
      return(f);
   finish F_GLOBAL;
   xa = {.5 1.5};
   xb = {3 -1};
   optn = {0 2};
   call nlptr(rca,xra,"F_GLOBAL",xa,optn);
   call nlptr(rcb,xrb,"F_GLOBAL",xb,optn);
   print xra xrb;
```

One way to find out whether the objective function has more than one local optimum is to run various optimizations with a pattern of different starting points.

For a more mathematical definition of optimality, refer to the *Kuhn-Tucker theorem* in standard optimization literature. Using a rather nonmathematical language, a local minimizer x^* satisfies the following conditions:

- There exists a small, feasible neighborhood of x^* that does not contain any point x with a smaller function value $f(x) < f(x^*)$.
- The vector of first derivatives (gradient) $g(x^*) = \nabla f(x^*)$ of the objective function f (projected toward the feasible region) at the point x^* is zero.
- The matrix of second derivatives $G(x^*) = \nabla^2 f(x^*)$ (Hessian matrix) of the objective function f (projected toward the feasible region) at the point x^* is positive definite.

A local maximizer has the largest value in a feasible neighborhood and a negative definite Hessian.

The iterative optimization algorithm terminates at the point x^t, which should be in a small neighborhood (in terms of a user-specified termination criterion) of a local optimizer x^*. If the point x^t is located on one or more active boundary or general linear constraints, the local optimization conditions are valid only for the feasible region. That is,

- the projected gradient, $Z^T g(x^t)$, must be sufficiently small
- the projected Hessian, $Z^T G(x^t) Z$, must be positive definite for minimization problems or negative definite for maximization problems

If there are n active constraints at the point x^t, the nullspace Z has zero columns and the projected Hessian has zero rows and columns. A matrix with zero rows and columns is considered positive as well as negative definite.

Kuhn-Tucker Conditions

The nonlinear programming (NLP) problem with one objective function f and m constraint functions c_i, which are continuously differentiable, is defined as follows:

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

In the preceding notation, n is the dimension of the function $f(x)$, and m_e is the number of equality constraints. The linear combination of objective and constraint functions

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

is the *Lagrange function*, and the coefficients λ_i are the *Lagrange multipliers*.

If the functions f and c_i are twice differentiable, the point x^* is an *isolated local minimizer* of the NLP problem, if there exists a vector $\lambda^* = (\lambda_1^*, \ldots, \lambda_m^*)$ that meets the following conditions:

- Kuhn-Tucker conditions

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

- Second-order condition

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

$$y^T \nabla_x c_i(x^*) = 0 i = 1, \ldots, m_e, \ \text{ and } \forall i \in m_e + 1, \ldots, m; \lambda_i^* > 0$$

  satisfies

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

In practice, you cannot expect that the constraint functions $c_i(x^*)$ will vanish within machine precision, and determining the set of active constraints at the solution x^* may not be simple.

Definition of Return Codes

The return code, which is represented by the output parameter *rc* in the optimization subroutines, indicates the reason for optimization termination. A positive value indicates successful termination, while a negative value indicates unsuccessful termination. Table 11.1 gives the reason for termination associated with each return code.

Table 11.1. Summary of Return Codes

rc	**Reason for Optimization Termination**
1	ABSTOL criterion satisfied (absolute F convergence)
2	ABSFTOL criterion satisfied (absolute F convergence)
3	ABSGTOL criterion satisfied (absolute G convergence)
4	ABSXTOL criterion satisfied (absolute X convergence)
5	FTOL criterion satisfied (relative F convergence)
6	GTOL criterion satisfied (relative G convergence)
7	XTOL criterion satisfied (relative X convergence)
8	FTOL2 criterion satisfied (relative F convergence)
9	GTOL2 criterion satisfied (relative G convergence)
10	n linear independent constraints are active at *xr* and none of them could be released to improve the function value
-1	objective function cannot be evaluated at starting point
-2	derivatives cannot be evaluated at starting point
-3	objective function cannot be evaluated during iteration
-4	derivatives cannot be evaluated during iteration
-5	optimization subroutine cannot improve the function value (this is a very general formulation and is used for various circumstances)
-6	there are problems in dealing with linearly dependent active constraints (changing the LCSING value in the *par* vector can be helpful)
-7	optimization process stepped outside the feasible region and the algorithm to return inside the feasible region was not successful (changing the LCEPS value in the *par* vector can be helpful)
-8	either the number of iterations or the number of function calls is larger than the prespecified values in the *tc* vector (MAXIT and MAXFU)
-9	this return code is temporarily not used (it is used in PROC NLP indicating that more CPU than a prespecified value was used)
-10	a feasible starting point cannot be computed

Objective Function and Derivatives

The input argument *fun* refers to an IML module that specifies a function that returns f, a vector of length m for least-squares subroutines or a scalar for other optimization subroutines. The returned f contains the values of the objective function (or the least-squares functions) at the point x. Note that for least-squares problems, you must specify the number of function values, m, with the first element of the *opt* argument to allocate memory for the return vector. All the modules that you can specify as input arguments (*"fun"*, *"grd"*, *"hes"*, *"jac"*, *"nlc"*, *"jacnlc"*, and *"ptit"*) allow only a single input argument, x, which is the parameter vector. Using the GLOBAL clause, you can provide more input arguments for these modules. Refer to the middle of the section, "Compartmental Analysis" for an example.

All the optimization algorithms assume that f is continuous inside the feasible region. For nonlinearly constrained optimization, this is also required for points outside the feasible region. Sometimes the objective function cannot be computed for all points of the specified feasible region; for example, the function specification may contain the SQRT or LOG function, which cannot be evaluated for negative arguments. You must make sure that the function and derivatives of the starting point can be evaluated. There are two ways to prevent large steps into infeasible regions of the parameter space during the optimization process:

- The preferred way is to restrict the parameter space by introducing more boundary and linear constraints. For example, the boundary constraint $x_j >= 1\text{E}{-10}$ prevents infeasible evaluations of $\log(x_j)$. If the function module takes the square root or the log of an intermediate result, you can use nonlinear constraints to try to avoid infeasible function evaluations. However, this may not ensure feasibility.
- Sometimes the preferred way is difficult to practice, in which case the function module may return a missing value. This may force the optimization algorithm to reduce the step length or the radius of the feasible region.

All the optimization techniques except the NLPNMS subroutine require continuous first-order derivatives of the objective function f. The NLPTR, NLPNRA, and NLPNRR techniques also require continuous second-order derivatives. If you do not provide the derivatives with the IML modules *"grd"*, *"hes"*, or *"jac"*, they are automatically approximated by finite difference formulas. Approximating first-order derivatives by finite differences usually requires n additional calls of the function module. Approximating second-order derivatives by finite differences using only function calls can be extremely computationally expensive. Hence, if you decide to use the NLPTR, NLPNRA, or NLPNRR subroutines, you should specify at least analytical first-order derivatives. Then, approximating second-order derivatives by finite differences requires only n or $2n$ additional calls of the function and gradient modules.

For all input and output arguments, the subroutines assume that

- the number of parameters n corresponds to the number of columns. For example, x, the input argument to the modules, and g, the output argument returned by the *"grd"* module, are row vectors with n entries, and G, the Hessian matrix returned by the *"hes"* module, must be a symmetric $n \times n$ matrix.
- the number of functions, m, corresponds to the number of rows. For example, the vector f returned by the *"fun"* module must be a column vector with m entries, and in least-squares problems, the Jacobian matrix **J** returned by the *"jac"* module must be an $m \times n$ matrix.

You can verify your analytical derivative specifications by computing finite difference approximations of the derivatives of f with the NLPFDD subroutine. For most applications, the finite difference approximations of the derivatives will be very precise. Occasionally, difficult objective functions and zero x coordinates will cause problems. You can use the *par* argument to specify the number of accurate digits in the evaluation of the objective function; this defines the step size h of the first- and second-order finite difference formulas. See the section “Finite Difference Approximations of Derivatives” on page 314.

Note: For some difficult applications, the finite difference approximations of derivatives that are generated by default may not be precise enough to solve the optimization or least-squares problem. In such cases, you may be able to specify better derivative approximations by using a better approximation formula. You can submit your own finite difference approximations using the IML modules *"grd"*, *"hes"*, *"jac"*, or *"jacnlc"*. See Example 11.3 on page 343 for an illustration.

In many applications, calculations used in the computation of f can help compute derivatives at the same point efficiently. You can save and reuse such calculations with the GLOBAL clause. As with many other optimization packages, the subroutines perform calls of the *"grd"*, *"hes"*, or *"jac"* modules only after a call of the *"fun"* module.

The following statements specify modules for the function, gradient, and Hessian matrix of the Rosenbrock problem:

```
proc iml;
   start F_ROSEN(x);
      y1 = 10. * (x[2] - x[1] * x[1]);
      y2 = 1. - x[1];
      f  = .5 * (y1 * y1 + y2 * y2);
      return(f);
   finish F_ROSEN;

   start G_ROSEN(x);
      g = j(1,2,0.);
      g[1] = -200.*x[1]*(x[2]-x[1]*x[1]) - (1.-x[1]);
      g[2] =  100.*(x[2]-x[1]*x[1]);
      return(g);
   finish G_ROSEN;

   start H_ROSEN(x);
      h = j(2,2,0.);
      h[1,1] = -200.*(x[2] - 3.*x[1]*x[1]) + 1.;
      h[2,2] =  100.;
      h[1,2] = -200. * x[1];
      h[2,1] = h[1,2];
      return(h);
   finish H_ROSEN;
```

The following statements specify a module for the Rosenbrock function when considered as a least-squares problem. They also specify the Jacobian matrix of the least-squares functions.

```
proc iml;
   start F_ROSEN(x);
      y = j(1,2,0.);
      y[1] = 10. * (x[2] - x[1] * x[1]);
      y[2] = 1. - x[1];
     return(y);
   finish F_ROSEN;

   start J_ROSEN(x);
      jac = j(2,2,0.);
      jac[1,1] = -20. * x[1]; jac[1,2] = 10.;
      jac[2,1] = -1.;         jac[2,2] =  0.;
      return(jac);
   finish J_ROSEN;
```

Diagonal or Sparse Hessian Matrices

In the unconstrained or only boundary constrained case, the NLPNRA algorithm can take advantage of diagonal or sparse Hessian matrices submitted by the *"hes"* module. If the Hessian matrix G of the objective function f has a large proportion of zeros, you may save computer time and memory by specifying a sparse Hessian of dimension $nn \times 3$ rather than a dense $n \times n$ Hessian. Each of the nn rows (i, j, g) of the matrix returned by the sparse Hessian module defines a nonzero element g_{ij} of the Hessian matrix. The row and column location is given by i and j, and g gives the nonzero value. During the optimization process, only the values g can be changed in each call of the Hessian module *"hes"*; the sparsity structure (i, j) must be kept the same. That means that some of the values g can be zero for particular values of x. To allocate sufficient memory before the first call of the Hessian module, you must specify the number of rows, nn, by setting the ninth element of the *opt* argument.

Example 22 of Moré, Garbow, and Hillstrom (1981) illustrates the sparse Hessian module input. The objective function, which is the Extended Powell's Singular Function, for $n = 40$ is a least-squares problem:

$$f(x) = \frac{1}{2}\{f_1^2(x) + \cdots + f_m^2(x)\}$$

with

$$\begin{aligned}
f_{4i-3}(x) &= x_{4i-3} + 10x_{4i-2} \\
f_{4i-2}(x) &= \sqrt{5}(x_{4i-1} - x_{4i}) \\
f_{4i-1}(x) &= (x_{4i-2} - 2x_{4i-1})^2 \\
f_{4i}(x) &= \sqrt{10}(x_{4i-3} - x_{4i})^2
\end{aligned}$$

The function and gradient modules are

```
start f_nlp22(x);
   n=ncol(x);
   f = 0.;
   do i=1 to n-3 by 4;
```

```
         f1 = x[i] + 10. * x[i+1];
         r2 = x[i+2] - x[i+3];
         f2 = 5. * r2;
         r3 = x[i+1] - 2. * x[i+2];
         f3 = r3 * r3;
         r4 = x[i] - x[i+3];
         f4 = 10. * r4 * r4;
         f = f + f1 * f1 + r2 * f2 + f3 * f3 + r4 * r4 * f4;
      end;
      f = .5 * f;
      return(f);
   finish f_nlp22;

   start g_nlp22(x);
      n=ncol(x);
      g = j(1,n,0.);
         do i=1 to n-3 by 4;
            f1 = x[i] + 10. * x[i+1];
            f2 = 5. * (x[i+2] - x[i+3]);
            r3 = x[i+1] - 2. * x[i+2];
            f3 = r3 * r3;
            r4 = x[i] - x[i+3];
            f4 = 10. * r4 * r4;
            g[i]   = f1 + 2. * r4 * f4;
            g[i+1] = 10. * f1 + 2. * r3 * f3;
            g[i+2] = f2 - 4. * r3 * f3;
            g[i+3] = -f2 - 2. * r4 * f4;
         end;
         return(g);
   finish g_nlp22;
```

You can specify the sparse Hessian with the following module:

```
   start hs_nlp22(x);
      n=ncol(x);
      nnz = 8 * (n / 4);
      h = j(nnz,3,0.);
      j = 0;
      do i=1 to n-3 by 4;
         f1 = x[i] + 10. * x[i+1];
         f2 = 5. * (x[i+2] - x[i+3]);
         r3 = x[i+1] - 2. * x[i+2];
         f3 = r3 * r3;
         r4 = x[i] - x[i+3];
         f4 = 10. * r4 * r4;
         j= j + 1; h[j,1] = i; h[j,2] = i;
         h[j,3] = 1. + 4. * f4;
         h[j,3] = h[j,3] + 2. * f4;
         j= j+1; h[j,1] = i; h[j,2] = i+1;
         h[j,3] = 10.;
         j= j+1; h[j,1] = i; h[j,2] = i+3;
         h[j,3] = -4. * f4;
         h[j,3] = h[j,3] - 2. * f4;
         j= j+1; h[j,1] = i+1; h[j,2] = i+1;
         h[j,3] = 100. + 4. * f3;
         h[j,3] = h[j,3] + 2. * f3;
         j= j+1; h[j,1] = i+1; h[j,2] = i+2;
         h[j,3] = -8. * f3;
         h[j,3] = h[j,3] - 4. * f3;
         j= j+1; h[j,1] = i+2; h[j,2] = i+2;
         h[j,3] = 5. + 16. * f3;
         h[j,3] = h[j,3] + 8. * f3;
         j= j+1; h[j,1] = i+2; h[j,2] = i+3;
         h[j,3] = -5.;
         j= j+1; h[j,1] = i+3; h[j,2] = i+3;
         h[j,3] = 5. + 4. * f4;
         h[j,3] = h[j,3] + 2. * f4;
      end;
      return(h);
   finish hs_nlp22;

   n = 40;
   x = j(1,n,0.);
   do i=1 to n-3 by 4;
      x[i] = 3.; x[i+1] = -1.; x[i+3] = 1.;
   end;
   opt = j(1,11,.); opt[2]= 3; opt[9]= 8 * (n / 4);
   call nlpnra(xr,rc,"f_nlp22",x,opt) grd="g_nlp22" hes="hs_nlp22";
```

Note: If the sparse form of Hessian defines a diagonal matrix (that is, $i = j$ in all nn rows), the NLPNRA algorithm stores and processes a diagonal matrix G. If you do not specify any general linear constraints, the NLPNRA subroutine uses only order n memory.

Finite Difference Approximations of Derivatives

If the optimization technique needs first- or second-order derivatives and you do not specify the corresponding IML modules *"grd"*, *"hes"*, *"jac"*, or *"jacnlc"*, the derivatives are approximated by finite difference formulas using only calls of the module *"fun"*. If the optimization technique needs second-order derivatives and you specify the *"grd"* module but not the *"hes"* module, the subroutine approximates the second-order derivatives by finite differences using n or $2n$ calls of the *"grd"* module.

The eighth element of the *opt* argument specifies the type of finite difference approximation used to compute first- or second-order derivatives and whether the finite difference intervals, h, should be computed by an algorithm of Gill, Murray, Saunders, and Wright (1983). The value of *opt*[8] is a two-digit integer, ij.

- If *opt*[8] is missing or $j = 0$, the fast but not very precise forward difference formulas are used; if $j \neq 0$, the numerically more expensive central difference formulas are used.
- If *opt*[8] is missing or $i \neq 1, 2,$ or 3, the finite difference intervals h are based only on the information of *par*[8] or *par*[9], which specifies the number of

accurate digits to use in evaluating the objective function and nonlinear constraints, respectively. If $i = 1, 2$, or 3, the intervals are computed with an algorithm by Gill, Murray, Saunders, and Wright (1983). For $i = 1$, the interval is based on the behavior of the objective function; for $i = 2$, the interval is based on the behavior of the nonlinear constraint functions; and for $i = 3$, the interval is based on the behavior of both the objective function and the nonlinear constraint functions.

Forward Difference Approximations

- 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, when the *"grd"* module is not specified (Dennis and Schnabel 1983): for a dense Hessian matrix, $n + n^2/2$ 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, when the *"grd"* module is specified (Dennis and Schnabel 1983): 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, when the *"grd"* module is not specified (Abramowitz and Stegun 1972): for a dense Hessian matrix, $2n + 2n^2$ additional function calls are needed.

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

- Second-order derivatives based on gradient calls, when the *"grd"* module is specified: $2n$ additional gradient calls are needed.

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

The step sizes h_j, $j = 1, \ldots, n$, are defined as follows:

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

If the algorithm of Gill, Murray, Saunders, and Wright (1983) is not used to compute η_j, a constant value $\eta = \eta_j$ is used depending on the value of *par*[8].

- If the number of accurate digits is specified by *par*$[8] = k_1$, then η is set to 10^{-k_1}.
- If *par*[8] is not specified, η is set to the machine precision, ϵ.

If central difference formulas are not specified, the optimization algorithm will switch automatically from the forward-difference formula to a corresponding central-difference formula during the iteration process if one of the following two criteria is satisfied:

- The absolute maximum gradient element is less than or equal to 100 times the ABSGTOL threshold.
- The term on the left of the GTOL criterion is less than or equal to $\max(1\text{E}{-}6,\ 100{\times}\text{GTOL threshold})$. The 1E$-$6 ensures that the switch is performed even if you set the GTOL threshold to zero.

The algorithm of Gill, Murray, Saunders, and Wright (1983) that computes the finite difference intervals h_j can be very expensive in the number of function calls it uses. If this algorithm is required, it is performed twice, once before the optimization process starts and once after the optimization terminates.

Many applications need considerably more time for computing second-order derivatives than for computing first-order derivatives. In such cases, you should use a quasi-Newton or conjugate gradient technique.

If you specify a vector, *c*, of nc nonlinear constraints with the "*nlc*" module but you do not specify the "*jacnlc*" module, the first-order formulas can be used to compute finite difference approximations of the $nc \times n$ Jacobian matrix of the nonlinear constraints.

$$(\nabla c_i) = \left(\frac{\partial c_i}{\partial x_j} \right), \quad i = 1, \ldots, nc, \;\; j = 1, \ldots, n$$

You can specify the number of accurate digits in the constraint evaluations with *par*[9]. This specification also defines the step sizes h_j, $j = 1, \ldots, n$.

Note: If you are not able to specify analytic derivatives and if the finite-difference approximations provided by the subroutines are not good enough to solve your optimization problem, you may be able to implement better finite-difference approximations with the "*grd*", "*hes*", "*jac*", and "*jacnlc*" module arguments.

Parameter Constraints

You can specify constraints in the following ways:

- The matrix input argument "*blc*" enables you to specify boundary and general linear constraints.
- The IML module input argument "*nlc*" enables you to specify general constraints, particularly nonlinear constraints.

Specifying the BLC Matrix

The input argument "*blc*" specifies an $n_1 \times n_2$ constraint matrix, where n_1 is two more than the number of linear constraints, and n_2 is given by

$$n2 = \begin{cases} n & \text{if } 1 \leq n1 \leq 2 \\ n+2 & \text{if } n1 > 2 \end{cases}$$

The first two rows define lower and upper bounds for the n parameters, and the remaining $c = n_1 - 2$ rows define general linear equality and inequality constraints. Missing values in the first row (lower bounds) substitute for the largest negative floating point value, and missing values in the second row (upper bounds) substitute for the largest positive floating point value. Columns $n+1$ and $n+2$ of the first two rows are not used.

The following c rows of the "*blc*" argument specify c linear equality or inequality constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \quad (\leq \mid = \mid \geq) \quad b_i, \quad i = 1, \ldots, c$$

Each of these c rows contains the coefficients a_{ij} in the first n columns. Column $n+1$ specifies the kind of constraint, as follows:

- *blc*$[n+1] = 0$ indicates an equality constraint.
- *blc*$[n+1] = 1$ indicates a $\geq$ inequality constraint.
- *blc*$[n+1] = -1$ indicates a $\leq$ inequality constraint.

Column $n+2$ specifies the right-hand side, b_i. A missing value in any of these rows corresponds to a value of zero.

For example, suppose you have a problem with the following constraints on x_1,x_2, x_3, x_4:

$$\begin{array}{ccccc} 2 & \leq & x_1 & \leq & 100 \\ & & x_2 & \leq & 40 \\ 0 & \leq & x_4 & & \end{array}$$

$$\begin{array}{rcrcrcrcl} 4x_1 & + & 3x_2 & - & x_3 & & & \leq & 30 \\ & & x_2 & & & + & 6x_4 & \geq & 17 \\ x_1 & - & x_2 & & & & & = & 8 \end{array}$$

The following statements specify the matrix CON, which can be used as the "*blc*" argument to specify the preceding constraints:

```
proc iml;
   con = {   2   .   .   0    .    . ,
           100  40   .   .    .    . ,
             4   3  -1   .   -1   30 ,
             .   1   .   6    1   17 ,
             1  -1   .   .    0    8   };
```

Specifying the NLC and JACNLC Modules

The input argument "*nlc*" specifies an IML module that returns a vector, c, of length nc, with the values, c_i, of the nc linear or nonlinear constraints

$$\begin{array}{rcll} c_i(x) & = & 0, & i = 1, \ldots, nec, \\ c_i(x) & \geq & 0, & i = nec + 1, \ldots, nc, \end{array}$$

for a given input parameter point x.

Note: You must specify the number of equality constraints, nec, and the total number of constraints, nc, returned by the "*nlc*" module to allocate memory for the return vector. You can do this with the *opt*[11] and *opt*[10] arguments, respectively.

For example, consider the problem of minimizing the objective function $f(x_1, x_2) = x_1 x_2$ in the interior of the unit circle, $x_1^2 + x_2^2 \leq 1$. The constraint can also be written as $c_1(x) = 1 - x_1^2 - x_2^2 \geq 0$. The following statements specify modules for the objective and constraint functions and call the NLPNMS subroutine to solve the minimization problem:

```
proc iml;
   start F_UC2D(x);
      f = x[1] * x[2];
      return(f);
   finish F_UC2D;

   start C_UC2D(x);
      c = 1. - x * x‘;
      return(c);
   finish C_UC2D;

   x = j(1,2,1.);
   optn= j(1,10,.); optn[2]= 3; optn[10]= 1;
   CALL NLPNMS(rc,xres,"F_UC2D",x,optn) nlc="C_UC2D";
```

To avoid typing multiple commas, you can specify the "*nlc*" input argument with a keyword, as in the preceding code. The number of elements of the return vector is specified by OPTN$[10] = 1$. There is a missing value in OPTN$[11]$, so the subroutine assumes there are zero equality constraints.

The NLPQN algorithm uses the $nc \times n$ Jacobian matrix of first-order derivatives

$$(\nabla_x c_i(x)) = \left(\frac{\partial c_i}{\partial x_j} \right), \quad i = 1, \ldots, nc, \quad j = 1, \ldots, n$$

of the nc equality and inequality constraints, c_i, for each point passed during the iteration. You can use the "*jacnlc*" argument to specify an IML module that returns the Jacobian matrix **JC**. If you specify the "*nlc*" module without using the "*jacnlc*" argument, the subroutine uses finite difference approximations of the first-order derivatives of the constraints.

Note: The COBYLA algorithm in the NLPNMS subroutine and the NLPQN subroutine are the only optimization techniques that enable you to specify nonlinear constraints with the "*nlc*" input argument.

Options Vector

The options vector, represented by the *"opt"* argument, enables you to specify a variety of options, such as the amount of printed output or particular update or line-search techniques. Table 11.2 gives a summary of the available options.

Table 11.2. Summary of the Elements of the Options Vector

Index	Description
1	specifies minimization, maximization, or the number of least-squares functions
2	specifies the amount of printed output
3	NLPDD, NLPLM, NLPNRA, NLPNRR, NLPTR: specifies the scaling of the Hessian matrix (HESCAL)
4	NLPCG, NLPDD, NLPHQN, NLPQN: specifies the update technique (UPDATE)
5	NLPCG, NLPHQN, NLPNRA, NLPQN (with no nonlinear constraints): specifies the line-search technique (LIS)
6	NLPHQN: specifies version of hybrid algorithm (VERSION) NLPQN with nonlinear constraints: specifies version of μ update
7	NLPDD, NLPHQN, NLPQN: specifies initial Hessian matrix (INHESSIAN)
8	Finite Difference Derivatives: specifies type of differences and how to compute the difference interval
9	NLPNRA: specifies the number of rows returned by the sparse Hessian module
10	NLPNMS, NLPQN: specifies the total number of constraints returned by the *"nlc"* module
11	NLPNMS, NLPQN: specifies the number of equality constraints returned by the *"nlc"* module

The following list contains detailed explanations of the elements of the options vector:

- **opt[1]**
  indicates whether the problem is minimization or maximization. The default, $opt[1] = 0$, specifies a minimization problem, and $opt[1] = 1$ specifies a maximization problem. For least-squares problems, $opt[1] = m$ specifies the number of functions or observations, which is the number of values returned by the *"fun"* module. This information is necessary to allocate memory for the return vector of the *"fun"* module.

- **opt[2]**
  specifies the amount of output printed by the subroutine. The higher the value of $opt[2]$, the more printed output is produced. The following table indicates the specific items printed for each value.

Value of *opt*[2]	**Printed Output**
0	No printed output is produced. This is the default.
1	The summaries for optimization start and termination are produced, as well as the iteration history.
2	The initial and final parameter estimates are also printed.
3	The values of the termination criteria and other control parameters are also printed.
4	The parameter vector, x, is also printed after each iteration.
5	The gradient vector, g, is also printed after each iteration.

- **opt[3]**
  selects a scaling for the Hessian matrix, $\mathbf{G}$. This option is relevant only for the NLPDD, NLPLM, NLPNRA, NLPNRR, and NLPTR subroutines. If *opt*$[3] \neq 0$, the first iteration and each restart iteration set the diagonal scaling matrix $\mathbf{D}^{(0)} = diag(d_i^{(0)})$, where

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

  and $G_{i,i}^{(0)}$ are the diagonal elements of the Hessian matrix, and ϵ is the machine precision. The diagonal scaling matrix $\mathbf{D}^{(0)} = diag(d_i^{(0)})$ is updated as indicated in the following table.

Value of *opt*[3]	**Scaling Update**
0	No scaling is done.
1	Moré (1978) scaling update: $$d_i^{(k+1)} = \max\left(d_i^{(k)}, \sqrt{\max(|G_{i,i}^{(k)}|, \epsilon)}\right)$$
2	Dennis, Gay, and Welsch (1981) scaling update: $$d_i^{(k+1)} = \max\left(0.6 * d_i^{(k)}, \sqrt{\max(|G_{i,i}^{(k)}|, \epsilon)}\right)$$
3	d_i is reset in each iteration: $d_i^{(k+1)} = \sqrt{\max(|G_{i,i}^{(k)}|, \epsilon)}$

  For the NLPDD, NLPNRA, NLPNRR, and NLPTR subroutines, the default is *opt*$[3] = 0$; for the NLPLM subroutine, the default is *opt*$[3] = 1$.

- **opt[4]**
  defines the update technique for (dual) quasi-Newton and conjugate gradient techniques. This option applies to the NLPCG, NLPDD, NLPHQN, and NLPQN subroutines. For the NLPCG subroutine, the following update techniques are available.

Value of *opt*[4]	**Update Method for NLPCG**
1	automatic restart method of Powell (1977) and Beale (1972). This is the default.
2	Fletcher-Reeves update (Fletcher 1987)
3	Polak-Ribiere update (Fletcher 1987)
4	Conjugate-descent update of Fletcher (1987)

For the unconstrained or linearly constrained NLPQN subroutine, the following update techniques are available.

Value of *opt*[4]	**Update Method for NLPQN**
1	dual Broyden, Fletcher, Goldfarb, and Shanno (DBFGS) update of the Cholesky factor of the Hessian matrix. This is the default.
2	dual Davidon, Fletcher, and Powell (DDFP) update of the Cholesky factor of the Hessian matrix
3	original Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update of the inverse Hessian matrix
4	original Davidon, Fletcher, and Powell (DFP) update of the inverse Hessian matrix

For the NLPQN subroutine used with the *"nlc"* module and for the NLPDD and NLPHQN subroutines, only the first two update techniques in the second table are available.

- **opt[5]**
  defines the line-search technique for the unconstrained or linearly constrained NLPQN subroutine, as well as the NLPCG, NLPHQN, and NLPNRA subroutines. Refer to Fletcher (1987) for an introduction to line-search techniques. The following table describes the available techniques.

Value of *opt*[5]	**Line-Search Method**
1	This method needs the same number of function and gradient calls for cubic interpolation and cubic extrapolation; it is similar to a method used by the Harwell subroutine library.
2	This method needs more function than gradient calls for quadratic and cubic interpolation and cubic extrapolation; it is implemented as shown in Fletcher (1987) and can be modified to exact line search with the *par*[6] argument (see the "Control Parameters Vector" section on page 332). This is the default for the NLPCG, NLPNRA, and NLPQN subroutines.
3	This method needs the same number of function and gradient calls for cubic interpolation and cubic extrapolation; it is implemented as shown in Fletcher (1987) and can be modified to exact line search with the *par*[6] argument.
4	This method needs the same number of function and gradient calls for stepwise extrapolation and cubic interpolation.
5	This method is a modified version of the *opt*[5]=4 method.
6	This method is the golden section line search of Polak (1971), which uses only function values for linear approximation.
7	This method is the bisection line search of Polak (1971), which uses only function values for linear approximation.
8	This method is the Armijo line-search technique of Polak (1971), which uses only function values for linear approximation.

For the NLPHQN least-squares subroutine, the default is a special line-search method that is based on an algorithm developed by Lindström and Wedin (1984). Although it needs more memory, this method sometimes works better with large least-squares problems.

- **opt[6]**
  is used only for the NLPHQN subroutine and the NLPQN subroutine with nonlinear constraints.

  In the NLPHQN subroutine, it defines the criterion for the decision of the hybrid algorithm to step in a Gauss-Newton or a quasi-Newton search direction. You can specify one of the three criteria that correspond to the methods of Fletcher and Xu (1987). The methods are HY1 (*opt*[6]=1), HY2 (*opt*[6]=2), and HY3 (*opt*[6]=2), and the default is HY2.

  In the NLPQN subroutine with nonlinear constraints, it defines the version of the algorithm used to update the vector μ of the Lagrange multipliers. The default is *opt*[6]=2, which specifies the approach of Powell (1982a,b). You can specify the approach of Powell (1978b) with *opt*[6]=1.

- **opt[7]**
  defines the type of start matrix, $G^{(0)}$, used for the Hessian approximation. This option applies only to the NLPDD, NLPHQN, and NLPQN subroutines. If *opt*[7]=0, which is the default, the quasi-Newton algorithm starts with a multi-

ple of the identity matrix where the scalar factor depends on *par*[10]; otherwise, it starts with the Hessian matrix computed at the starting point $x^{(0)}$.

- **opt[8]**
  defines the type of finite difference approximation used to compute first- or second-order derivatives and whether the finite difference intervals, h, should be computed using an algorithm of Gill, Murray, Saunders, and Wright (1983). The value of *opt*[8] is a two-digit integer, ij.

  If *opt*[8] is missing or $j = 0$, the fast but not very precise forward difference formulas are used; if $j \neq 0$, the numerically more expensive central difference formulas are used.

  If *opt*[8] is missing or $i \neq 1, 2,$ or 3, the finite difference intervals h are based only on the information of *par*[8] or *par*[9], which specifies the number of accurate digits to use in evaluating the objective function and nonlinear constraints, respectively. If $i = 1, 2,$ or 3, the intervals are computed with an algorithm by Gill, Murray, Saunders, and Wright (1983). For $i = 1$, the interval is based on the behavior of the objective function; for $i = 2$, the interval is based on the behavior of the nonlinear constraint functions; and for $i = 3$, the interval is based on the behavior of both the objective function and the nonlinear constraint functions.

  The algorithm of Gill, Murray, Saunders, and Wright (1983) that computes the finite difference intervals h_j can be very expensive in the number of function calls it uses. If this algorithm is required, it is performed twice, once before the optimization process starts and once after the optimization terminates. See the "Finite Difference Approximations of Derivatives" section on page 314 for details.

- **opt[9]**
  indicates that the Hessian module *"hes"* returns a sparse definition of the Hessian, in the form of an $nn \times 3$ matrix instead of the default dense $n \times n$ matrix. If *opt[9]* is zero or missing, the Hessian module must return a dense $n \times n$ matrix. If you specify *opt*$[9] = nn$, the module must return a sparse $nn \times 3$ table. See the "Objective Function and Derivatives" section on page 309 for more details. This option applies only to the NLPNRA algorithm. If the dense specification contains a large proportion of analytical zero derivatives, the sparse specification may save memory and computer time.

- **opt[10]**
  specifies the total number of nonlinear constraints returned by the *"nlc"* module. If you specify nc nonlinear constraints with the *"nlc"* argument module, you must specify *opt*$[10] = nc$ to allocate memory for the return vector.

- **opt[11]**
  specifies the number of nonlinear equality constraints returned by the *"nlc"* module. If the first nec constraints are equality constraints, you must specify *opt*$[11] = nec$. The default value is *opt*$[11] = 0$.

Termination Criteria

The input argument *tc* specifies a vector of bounds corresponding to a set of termination criteria that are tested in each iteration. If you do not specify an IML module with the *"ptit"* argument, these bounds determine when the optimization process stops.

If you specify the *"ptit"* argument, the *"tc"* argument is ignored. The module specified by the *"ptit"* argument replaces the subroutine that is used by default to test the termination criteria. The module is called in each iteration with the current location, x, and the value, f, of the objective function at x. The module must give a return code, rc, that decides whether the optimization process is to be continued or terminated. As long as the module returns $rc = 0$, the optimization process continues. When the module returns $rc \neq 0$, the optimization process stops.

If you use the *tc* vector, the optimization techniques stop the iteration process when at least one of the corresponding set of termination criteria are satisfied. Table 11.3 and Table 11.4 indicate the criterion associated with each element of the *tc* vector. There is a default for each criterion, and if you specify a missing value for the corresponding element of the *tc* vector, the default value is used. You can avoid termination with respect to the ABSFTOL, ABSGTOL, ABSXTOL, FTOL, FTOL2, GTOL, GTOL2, and XTOL criteria by specifying a value of zero for the corresponding element of the *tc* vector.

Table 11.3. Termination Criteria for the NLPNMS Subroutine

Index	Description
1	maximum number of iterations (MAXIT)
2	maximum number of function calls (MAXFU)
3	absolute function criterion (ABSTOL)
4	relative function criterion (FTOL)
5	relative function criterion (FTOL2)
6	absolute function criterion (ABSFTOL)
7	FSIZE value used in FTOL criterion
8	relative parameter criterion (XTOL)
9	absolute parameter criterion (ABSXTOL)
9	size of final trust-region radius ρ (COBYLA algorithm)
10	XSIZE value used in XTOL criterion

Table 11.4. Termination Criteria for Other Subroutines

Index	Description
1	maximum number of iterations (MAXIT)
2	maximum number of function calls (MAXFU)
3	absolute function criterion (ABSTOL)
4	relative gradient criterion (GTOL)
5	relative gradient criterion (GTOL2)
6	absolute gradient criterion (ABSGTOL)
7	relative function criterion (FTOL)
8	predicted function reduction criterion (FTOL2)
9	absolute function criterion (ABSFTOL)
10	FSIZE value used in GTOL and FTOL criterion
11	relative parameter criterion (XTOL)
12	absolute parameter criterion (ABSXTOL)
13	XSIZE value used in XTOL criterion

Criteria Used by All Techniques

The following list indicates the termination criteria that are used with all the optimization techniques:

- **tc[1]**
  specifies the maximum number of iterations in the optimization process (MAXIT). The default values are

NLPNMS:	MAXIT=1000
NLPCG:	MAXIT=400
Others:	MAXIT=200

- **tc[2]**
  specifies the maximum number of function calls in the optimization process (MAXFU). The default values are

NLPNMS:	MAXFU=3000
NLPCG:	MAXFU=1000
Others:	MAXFU=500

- **tc[3]**
  specifies the absolute function convergence criterion (ABSTOL). For minimization, termination requires $f^{(k)} = f(x^{(k)}) \leq \mathit{ABSTOL}$, while for maximization, termination requires $f^{(k)} = f(x^{(k)}) \geq \mathit{ABSTOL}$. The default values are the negative and positive square roots of the largest double precision value, for minimization and maximization, respectively.

These criteria are useful when you want to divide a time-consuming optimization problem into a series of smaller problems.

Termination Criteria for NLPNMS

Since the Nelder-Mead simplex algorithm does not use derivatives, no termination criteria are available that are based on the gradient of the objective function.

When the NLPNMS subroutine implements Powell's COBYLA algorithm, it uses only one criterion other than the three used by all the optimization techniques. The COBYLA algorithm is a trust-region method that sequentially reduces the radius, ρ, of a spheric trust region from the start radius, ρ_{beg}, which is controlled with the *par*[2] argument, to the final radius, ρ_{end}, which is controlled with the *tc*[9] argument. The default value for *tc*[9] is ρ_{end} =1E−4. Convergence to small values of ρ_{end} may take many calls of the function and constraint modules and may result in numerical problems.

In addition to the criteria used by all techniques, the original Nelder-Mead simplex algorithm uses several other termination criteria, which are described in the following list:

- **tc[4]**
  specifies the relative function convergence criterion (FTOL). Termination requires a small relative difference between the function values of the vertices in the simplex with the largest and smallest function values.

  $$\frac{|f_{hi}^{(k)} - f_{lo}^{(k)}|}{\max(|f_{hi}^{(k)})|, \mathit{FSIZE})} \leq \mathit{FTOL}$$

  where *FSIZE* is defined by *tc*[7]. The default value is $tc[4] = 10^{-\mathrm{FDIGITS}}$, where FDIGITS is controlled by the *par*[8] argument. The *par*[8] argument has a default value of $\log_{10}(\epsilon)$, where ϵ is the machine precision. Hence, the default value for *FTOL* is ϵ.

- **tc[5]**
  specifies another relative function convergence criterion (FTOL2). Termination requires a small standard deviation of the function values of the $n+1$ simplex vertices $x_0^{(k)}, \ldots, x_n^{(k)}$.

  $$\sqrt{\frac{1}{n+1}\sum_l (f(x_l^{(k)}) - \overline{f}(x^{(k)}))^2} \leq \mathit{FTOL2}$$

  where $\overline{f}(x^{(k)}) = \frac{1}{n+1}\sum_l f(x_l^{(k)})$. If there are a active boundary constraints at $x^{(k)}$, the mean and standard deviation are computed only for the $n+1-a$ unconstrained vertices. The default is $tc[5]$ =1E−6.

- **tc[6]**
  specifies the absolute function convergence criterion (ABSFTOL). Termination requires a small absolute difference between the function values of the vertices in the simplex with the largest and smallest function values.

  $$|f_{hi}^{(k)} - f_{lo}^{(k)}| \leq \mathit{ABSFTOL}$$

The default is $tc[6] = 0$.

- **tc[7]**
  specifies the FSIZE value used in the FTOL termination criterion. The default is $tc[7] = 0$.

- **tc[8]**
  specifies the relative parameter convergence criterion (XTOL). Termination requires a small relative parameter difference between the vertices with the largest and smallest function values.

$$\frac{\max_j |x_j^{lo} - x_j^{hi}|}{\max(|x_j^{lo}|, |x_j^{hi}|, \mathit{XSIZE})} \leq \mathit{XTOL}$$

  The default is $tc[8]$ =1E−8.

- **tc[9]**
  specifies the absolute parameter convergence criterion (ABSXTOL). Termination requires either a small length, $\alpha^{(k)}$, of the vertices of a restart simplex or a small simplex size, $\delta^{(k)}$.

$$\begin{aligned} \alpha^{(k)} &\leq \mathit{ABSXTOL} \\ \delta^{(k)} &\leq \mathit{ABSXTOL} \end{aligned}$$

  where $\delta^{(k)}$ is defined as the L1 distance of the simplex vertex with the smallest function value, $y^{(k)}$, to the other n simplex points, $x_l^{(k)} \neq y$.

$$\delta^{(k)} = \sum_{x_l \neq y} \parallel x_l^{(k)} - y^{(k)} \parallel_1$$

  The default is $tc[9]$ =1E−8.

- **tc[10]**
  specifies the XSIZE value used in the XTOL termination criterion. The default is $tc[10] = 0$.

Termination Criteria for Unconstrained and Linearly Constrained Techniques

- **tc[4]**
  specifies the relative gradient convergence criterion (GTOL). For all techniques except the NLPCG subroutine, termination requires that the normalized predicted function reduction is small.

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

where *FSIZE* is defined by *tc*[10]. For the NLPCG technique (where a reliable Hessian estimate is not available),

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

is used. The default is $tc[4] =$1E$-$8.

- **tc[5]**
  specifies another relative gradient convergence criterion (GTOL2). This criterion is used only by the NLPLM subroutine.

  $$\max_j \frac{|g_j(x^{(k)})|}{\sqrt{f(x^{(k)}) G_{j,j}^{(k)}}} \leq \mathit{GTOL2}$$

  The default is *tc*[5]=0.

- **tc[6]**
  specifies the absolute gradient convergence criterion (ABSGTOL). Termination requires that the maximum absolute gradient element be small.

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

  The default is $tc[6] =$1E$-$5.

- **tc[7]**
  specifies the relative function convergence criterion (FTOL). Termination requires a small relative change of the function value in consecutive iterations.

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

  where $FSIZE$ is defined by *tc*[10]. The default is $tc[7] = 10^{-\mathrm{FDIGITS}}$, where FDIGITS is controlled by the *par*[8] argument. The *par*[8] argument has a default value of $\log_{10}(\epsilon)$, where ϵ is the machine precision. Hence, the default for *FTOL* is ϵ.

- **tc[8]**
  specifies another function convergence criterion (FTOL2). For least-squares problems, termination requires a small predicted reduction of the objective function, $df^{(k)} \approx f(x^{(k)}) - f(x^{(k)} + s^{(k)})$. The predicted reduction is computed by approximating the objective function by the first two terms of the Taylor series and substituting the Newton step, $s^{(k)} = -G^{(k)-1} g^{(k)}$, as follows:

  $$\begin{aligned} df^{(k)} &= -g^{(k)T} s^{(k)} - \frac{1}{2} s^{(k)T} G^{(k)} s^{(k)} \\ &= -\frac{1}{2} s^{(k)T} g^{(k)} \\ &\leq \mathit{FTOL2} \end{aligned}$$

The FTOL2 criterion is the unscaled version of the GTOL criterion. The default is *tc*[8]=0.

- **tc[9]**
  specifies the absolute function convergence criterion (ABSFTOL). Termination requires a small change of the function value in consecutive iterations.

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

  The default is *tc*[9]=0.

- **tc[10]**
  specifies the FSIZE value used in the GTOL and FTOL termination criteria. The default is *tc*[10]=0.

- **tc[11]**
  specifies the relative parameter convergence criterion (XTOL). Termination requires a small relative parameter change in consecutive iterations.

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

  The default is *tc*[11]=0.

- **tc[12]**
  specifies the absolute parameter convergence criterion (ABSXTOL). Termination requires a small Euclidean distance between parameter vectors in consecutive iterations.

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

  The default is *tc*[12]=0.

- **tc[13]**
  specifies the XSIZE value used in the XTOL termination criterion. The default is *tc*[13]=0.

Termination Criteria for Nonlinearly Constrained Techniques

The only algorithm available for nonlinearly constrained optimization other than the NLPNMS subroutine is the NLPQN subroutine, when you specify the *"nlc"* module argument. This method, unlike the other optimization methods, does not monotonically reduce the value of the objective function or some kind of merit function that combines objective and constraint functions. Instead, the algorithm uses the watchdog technique with backtracking of Chamberlain and others (1982). Therefore, no termination criteria are implemented that are based on the values x or f in consecutive iterations. In addition to the criteria used by all optimization techniques, there are three other termination criteria available; these are based on the Lagrange function

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

and its gradient

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

where m denotes the total number of constraints, $g = g(x)$ is the gradient of the objective function, and λ is the vector of Lagrange multipliers. The Kuhn-Tucker conditions require that the gradient of the Lagrange function is zero at the optimal point (x^*, λ^*), as follows:

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

- **tc[4]**
  specifies the GTOL criterion, which requires that the normalized predicted function reduction be small.

  $$\frac{|g(x^{(k)})s(x^{(k)})| + \sum_{i=1}^{m} |\lambda_i c_i(x^{(k)})|}{\max(|f(x^{(k)})|, \mathit{FSIZE})} \leq \mathit{GTOL}$$

  where *FSIZE* is defined by the *tc*[10] argument. The default is *tc*[4] =1E−8.

- **tc[6]**
  specifies the ABSGTOL criterion, which requires that the maximum absolute gradient element of the Lagrange function be small.

  $$\max_j |\{\nabla_x L(x^{(k)}, \lambda^{(k)})\}_j| \leq \mathit{ABSGTOL}$$

  The default is *tc*[6] =1E−5.

- **tc[8]**
  specifies the FTOL2 criterion, which requires that the predicted function reduction be small.

  $$|g(x^{(k)})s(x^{(k)})| + \sum_{i=1}^{m} |\lambda_i c_i| \leq \mathit{FTOL2}.$$

  The default is *tc*[8] =1E−6. This is the criterion used by the programs VMCWD and VF02AD of Powell (1982b).

Control Parameters Vector

For all optimization and least-squares subroutines, the input argument *par* specifies a vector of parameters that control the optimization process. For the NLPFDD and NLPFEA subroutines, the *par* argument is defined differently. For each element of the *par* vector there exists a default value, and if you specify a missing value, the default is used. Table 11.5 summarizes the uses of the *par* argument for the optimization and least-squares subroutines.

Table 11.5. Summary of the Control Parameters Vector

Index	**Description**
1	specifies the singularity criterion (SINGULAR)
2	specifies the initial step length or trust-region radius
3	specifies the range for active (violated) constraints (LCEPS)
4	specifies the Lagrange multiplier threshold for constraints (LCDEACT)
5	specifies a criterion to determine linear dependence of constraints (LCSING)
6	specifies the required accuracy of the line-search algorithms (LSPRECISION)
7	reduces the line-search step size in successive iterations (DAMPSTEP)
8	specifies the number of accurate digits used in evaluating the objective function (FDIGITS)
9	specifies the number of accurate digits used in evaluating the nonlinear constraints (CDIGITS)
10	specifies a scalar factor for the diagonal of the initial Hessian (DIAHES)

- **par[1]**
  specifies the singularity criterion for the decomposition of the Hessian matrix (SINGULAR). The value must be between zero and one, and the default is $par[1] =$1E-8.

- **par[2]**
  specifies different features depending on the subroutine in which it is used. In the NLPNMS subroutine, it defines the size of the start simplex. For the original Nelder-Mead simplex algorithm, the default value is $par[2] = 1$; for the COBYLA algorithm, the default is $par[2] = 0.5$. In the NLPCG, NLPQN, and NLPHQN subroutines, the *par*[2] argument specifies an upper bound for the initial step length for the line search during the first five iterations. The default initial step length is $par[2] = 1$. In the NLPTR, NLPDD, and NLPLM subroutines, the *par*[2] argument specifies a factor for the initial trust-region radius, Δ. For highly nonlinear functions, the default step length or trust-region radius can result in arithmetic overflows. In that case, you can specify stepwise decreasing values of *par*[2], such as *par*[2]=1E-1, *par*[2]=1E-2, *par*[2]=1E-4, until the subroutine starts to iterate successfully.

- **par[3]**
  specifies the range (LCEPS) for active and violated linear constraints. The ith constraint is considered an active constraint if the point $x^{(k)}$ satisfies the condition

  $$\left| \sum_{j=1}^{n} a_{ij} x_j^{(k)} - b_i \right| \le \mathit{LCEPS}(|b_i| + 1)$$

  where *LCEPS* is the value of *par*[3] and a_{ij} and b_i are defined as in the section "Parameter Constraints" on page 317. Otherwise, the constraint i is either an inactive inequality or a violated inequality or equality constraint. The default is *par*$[3]$ =1E$-$8. During the optimization process, the introduction of rounding errors can force the subroutine to increase the value of *par*[3] by a power of 10, but the value will never become larger than 1E$-$3.

- **par[4]**
  specifies a threshold (LCDEACT) for the Lagrange multiplier that decides whether an active inequality constraint must remain active or can be deactivated. For maximization, *par*[4] must be positive, and for minimization, *par*[4] must be negative. The default is

  $$\mathit{par}[4] = \pm \min\left(0.01, \max\left(0.1 \times \mathit{ABSGTOL},\ \ 0.001 \times \mathit{gmax}^{(k)}\right)\right)$$

  where the positive value is for maximization and the negative value is for minimization. *ABSGTOL* is the value of the absolute gradient criterion, and *gmax*$^{(k)}$ is the maximum absolute element of the gradient, $g^{(k)}$, or the projected gradient, $Z^T g^{(k)}$.

- **par[5]**
  specifies a criterion (LCSING) 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 is *par*$[5]$ =1E$-$8. As the value of *par*[5] increases, more active constraints are recognized as being linearly dependent. If the value of *par*[5] is larger than 0.1, it is reset to 0.1, and if it is negative, it is reset to zero.

- **par[6]**
  specifies the degree of accuracy (LSPRECISION) that should be obtained by the second or third line-search algorithm. This argument can be used with the NLPCG, NLPHQN, and NLPNRA algorithms and with the NLPQN algorithm if the *"nlc"* argument is specified. Usually, an imprecise line search is computationally inexpensive and successful, but for more difficult optimization problems, a more precise and time consuming line search may be necessary. Refer to Fletcher (1987) for details. If you have numerical problems, you should decrease the value of the *par*[6] argument to obtain a more precise line search. The default values are given in the following table.

Subroutine	Update Method	Default value
NLPCG	All	*par*[6] = 0.1
NLPHQN	DBFGS	*par*[6] = 0.1
NLPHQN	DDFP	*par*[6] = 0.06
NLPNRA	No update	*par*[6] = 0.9
NLPQN	BFGS, DBFGS	*par*[6] = 0.4
NLPQN	DFP, DDFP	*par*[6] = 0.06

- **par[7]**
  specifies a scalar factor (DAMPSTEP) that can be used to reduce the step size in each of the first five iterations. In each of these iterations, the starting step size, $\alpha^{(0)}$, can be no larger than the value of *par*[7] times the step size obtained by the line-search algorithm in the previous iteration. If *par*[7] is missing or if*par*[7]=0, which is the default, the starting step size in iteration t is computed as a function of the function change from the former iteration, $f^{(t-1)} - f^{(t)}$. If the computed value is outside the interval $[0.1, 10.0]$, it is moved to the next endpoint. You can further restrict the starting step size in the first five iterations with the *par*[2] argument.

- **par[8]**
  specifies the number of accurate digits (FDIGITS) used to evaluate the objective function. The default is $-\log_{10}(\epsilon)$, where ϵ is the machine precision, and fractional values are permitted. This value is used to compute the step size h for finite difference derivatives and the default value for the FTOL termination criterion.

- **par[9]**
  specifies the number of accurate digits (CDIGITS) used to evaluate the nonlinear constraint functions of the *"nlc"* module. The default is $-\log_{10}(\epsilon)$, where ϵ is the machine precision, and fractional values are permitted. The value is used to compute the step size h for finite difference derivatives. If first-order derivatives are specified by the *"jacnlc"* module, the *par*[9] argument is ignored.

- **par[10]**
  specifies a scalar factor (DIAHES) for the diagonal of the initial Hessian approximation. This argument is available in the NLPDD, NLPHQN, and NLPQN subroutines. If the *opt*[7] argument is not specified, the initial Hessian approximation is a multiple of the identity matrix determined by the magnitude of the initial gradient $g(x^{(0)})$. The value of the *par*[10] argument is used to specify $par[10] \times \mathbf{I}$ for the initial Hessian in the quasi-Newton algorithm.

Printing the Optimization History

Each optimization and least-squares subroutine prints the optimization history, as long as $opt[2] \geq 1$ and you do not specify the *"ptit"* module argument. You can use this output to check for possible convergence problems. If you specify the *"ptit"* argument, you can enter a print command inside the module, which is called at each iteration.

The amount of information printed depends on the *opt*[2] argument. See the section "Options Vector" on page 319.

The output consists of three main parts:

- **Optimization Start Output**
  The following information about the initial state of the optimization can be printed:
  - 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, at least one of which is temporarily released from the working set due to negative Lagrange multipliers.
  - the value of the objective function at the starting point
  - the value of the largest absolute (projected) gradient element
  - the initial trust-region radius for the NLPTR and NLPLM subroutines
- **General Iteration History**
  In general, the iteration history consists of one line of printed output for each iteration, with the exception of the Nelder-Mead simplex method. The NLPNMS subroutine prints a line only after several internal iterations because some of the termination tests are time-consuming compared to the simplex operations and because the subroutine typically uses many iterations.

  The iteration history always includes the following columns:
  - *iter* is the iteration number.
  - *nrest* is the number of iteration restarts.
  - *nfun* is the number of function calls.
  - *act* is the number of active constraints.
  - *optcrit* is the value of the optimization criterion.
  - *difcrit* is the difference between adjacent function values.
  - *maxgrad* is the maximum of the absolute (projected) gradient components.

  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.

  Some subroutines print additional information at each iteration; for details see the entry corresponding to each subroutine in the "Nonlinear Optimization and Related Subroutines" section on page 631.
- **Optimization Result Output**
  The output ends with the following information about 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 (+), there are more

active constraints, at least one of which is temporarily released from the working set due to negative Lagrange multipliers.

- the value of the objective function at the final point
- the value of the largest absolute (projected) gradient element

Nonlinear Optimization Examples

Example 11.1. Chemical Equilibrium

The following example is used in many test libraries for nonlinear programming. It appeared originally in Bracken and McCormick (1968).

The problem is to determine the composition of a mixture of various chemicals that satisfy the mixture's chemical equilibrium state. The second law of thermodynamics implies that at a constant temperature and pressure, a mixture of chemicals satisfies its chemical equilibrium state 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 free energy of the mixture.

The following notation is used in this problem:

m	number of chemical elements in the mixture
n	number of compounds in the mixture
x_j	number of moles for compound j, $j = 1, \dots, n$
s	total number of moles in the mixture, $s = \sum_{i=1}^n x_j$
a_{ij}	number of atoms of element i in a molecule of compound j
b_i	atomic weight of element i in the mixture $i = 1, \dots, n$

The constraints for the mixture are as follows. Each of the compounds must have a nonnegative number of moles.

$$x_j \geq 0, \quad j = 1, \dots, n$$

There is a mass balance relationship for each element. Each relation is given by a linear equality constraint.

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

The objective function is the total free energy of the mixture.

$$f(x) = \sum_{j=1}^n x_j \left[c_j + \ln\left(\frac{x_j}{s}\right) \right]$$

where

$$c_j = \left(\frac{F^0}{RT}\right)_j + \ln(P)$$

and $\left(F^0/RT\right)_j$ is the model standard free energy function for the j^{th} compound. The value of $\left(F^0/RT\right)_j$ is found in existing tables. P is the total pressure in atmospheres.

The problem is to determine the parameters x_j that minimize the objective function $f(x)$ subject to the nonnegativity and linear balance constraints. To illustrate this, consider the following situation. Determine the equilibrium composition of compound $\frac{1}{2}N_2H_4 + \frac{1}{2}O_2$ at temperature $T = 3500^\circ K$ and pressure $P = 750$ *psi*. The following table gives a summary of the information necessary to solve the problem.

				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

The following statements solve the minimization problem:

```
proc iml;
   c = { -6.089 -17.164 -34.054  -5.914 -24.721
         -14.986 -24.100 -10.708 -26.662 -22.179 };
   start F_BRACK(x) global(c);
      s = x[+];
      f = sum(x # (c + log(x / s)));
      return(f);
   finish F_BRACK;

   con = { .  .  .  .  .  .  .  .  .  .    .   .  ,
           .  .  .  .  .  .  .  .  .  .    .   .  ,
           1. 2. 2. .  .  1. .  .  .  1.   0.  2. ,
           .  .  .  1. 2. 1. 1. .  .  .    0.  1. ,
           .  .  1. .  .  .  1. 1. 2. 1.   0.  1. };
   con[1,1:10] = 1.e-6;

   x0 = j(1,10, .1);
   optn = {0 3};

   title 'NLPTR subroutine: No Derivatives';
   call nlptr(xres,rc,"F_BRACK",x0,optn,con);
```

The F_BRACK module specifies the objective function, $f(x)$. The matrix CON specifies the constraints. The first row gives the lower bound for each parameter, and to prevent the evaluation of the $\log(x)$ function for values of x that are too small, the lower bounds are set here to 1E$-$6. The following three rows contain the three linear equality constraints.

The starting point, which must be given to specify the number of parameters, is represented by X0. The first element of the OPTN vector specifies a minimization problem, and the second element specifies the amount of printed output.

The CALL NLPTR statement runs trust-region minimization. In this case, since no analytic derivatives are specified, the `F_BRACK` module is used to generate finite difference approximations for the gradient vector and Hessian matrix.

The output is shown in the following figures. The iteration history does not show any problems.

```
                          Optimization Start

Active Constraints                  3  Objective Function   -45.05516448
Max Abs Gradient     4.4710303342  Radius                             1
Element


                                            Max Abs          Trust
       Rest  Func Act   Objective  Obj Fun Gradient          Region
 Iter  arts Calls Con    Function   Change  Element  Lambda  Radius

    1     0     2   3'  -47.33413   2.2790   4.3613   2.456   1.000
    2     0     3   3'  -47.70051   0.3664   7.0044   0.908   0.418
    3     0     4   3   -47.73117   0.0307   5.3051       0   0.359
    4     0     5   3   -47.73426  0.00310   3.7015       0   0.118
    5     0     6   3   -47.73982  0.00555   2.3054       0  0.0169
    6     0     7   3   -47.74846  0.00864   1.3029  90.184 0.00476
    7     0     9   3   -47.75796  0.00950   0.5073       0  0.0134
    8     0    10   3   -47.76094  0.00297   0.0988       0  0.0124
    9     0    11   3   -47.76109 0.000155  0.00447       0  0.0111
   10     0    12   3   -47.76109 3.385E-7 0.000011       0 0.00332

                        Optimization Results

Iterations                         10  Function Calls                 13
Hessian Calls                      11  Active Constraints              3
Objective Function   -47.76109086  Max Abs Gradient     7.3901293E-6
                                   Element
Lambda                              0  Actual Over Pred                0
                                   Change
Radius               0.0033214552
```

The output lists the optimal parameters with the gradient.

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

          1 X1                0.040668        -9.785055
          2 X2                0.147730       -19.570111
          3 X3                0.783154       -34.792170
          4 X4                0.001414       -12.968920
          5 X5                0.485247       -25.937841
          6 X6                0.000693       -22.753976
          7 X7                0.027399       -28.190992
          8 X8                0.017947       -15.222060
          9 X9                0.037314       -30.444119
         10 X10               0.096871       -25.007115

          Value of Objective Function = -47.76109086
```

The three equality constraints are satisfied at the solution.

```
              Linear Constraints Evaluated at Solution


  [1] ACT -3.053E-16  = -2.0000 + 1.0000 * X1 + 2.0000 * X2
          + 2.0000 * X3 + 1.0000 * X6 + 1.0000 * X10

  [2] ACT -1.735E-17  = -1.0000 + 1.0000 * X4 + 2.0000 * X5
          + 1.0000 * X6 + 1.0000 * X7

  [3] ACT -1.527E-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 also printed. The elements of the projected gradient must be small to satisfy a first-order optimality condition.

```
                  First Order Lagrange Multipliers


                                              Lagrange
                Active Constraint            Multiplier

              Linear EC      [1]              -9.785055
              Linear EC      [2]             -12.968922
              Linear EC      [3]             -15.222061


                         Projected Gradient

                            Free    Projected
                       Dimension     Gradient

                               1  0.000000328
                               2 -9.703359E-8
                               3 3.2183113E-8
                               4 -0.000007390
                               5 -0.000005172
                               6 -0.000005669
                               7 -0.000000937
```

Example 11.2. Network Flow and Delay

The following example is taken from the user's guide of the GINO program (Liebman, Lasdon, Schrage, and Waren 1986). A simple network of five roads (arcs) can be illustrated by a path diagram.

The five roads connect four intersections illustrated by numbered nodes. Each minute, F vehicles enter and leave the network. The parameter x_{ij} refers to the flow from node i to node j. The requirement that traffic that flows 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 limit on the number of vehicles that can be handled per minute. These limits, denoted c_{ij}, can be enforced by boundary constraints:

$$0 \leq x_{ij} \leq c_{ij}$$

The goal in this problem is to maximize the flow, which is equivalent to maximizing the objective function $f(x)$, where $f(x)$ is

$$f(x) = x_{24} + x_{34}$$

The boundary constraints are

$$\begin{array}{ccc} 0 \leq & x_{12}, x_{32}, x_{34} & \leq 10 \\ 0 \leq & x_{13}, x_{24} & \leq 30 \end{array}$$

and the flow constraints are

$$\begin{array}{rcl} x_{13} & = & x_{32} + x_{34} \\ x_{24} & = & x_{12} + x_{32} \\ x_{12} + x_{13} & = & x_{24} + x_{34} \end{array}$$

The three linear equality constraints are linearly dependent. One of them is deleted automatically by the optimization subroutine. The following notation is used in this example:

$$X1 = x_{12}, \quad X2 = x_{13}, \quad X3 = x_{32}, \quad X4 = x_{24}, \quad X5 = x_{34}$$

Even though the NLPCG subroutine is used, any other optimization subroutine would also solve this small problem.

```
proc iml;
   title 'Maximum Flow Through a Network';
   start MAXFLOW(x);
      f = x[4] + x[5];
      return(f);
   finish MAXFLOW;

   con = {  0.  0.  0.  0.  0.   .   . ,
           10. 30. 10. 30. 10.   .   . ,
            0.  1. -1.  0. -1.  0.  0. ,
            1.  0.  1. -1.  0.  0.  0. ,
            1.  1.  0. -1. -1.  0.  0. };
   x = j(1,5, 1.);
   optn = {1 3};
   call nlpcg(xres,rc,"MAXFLOW",x,optn,con);
```

The optimal solution is shown in the following output.

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

 1 X1               10.000000             0     Upper BC
 2 X2               10.000000             0
 3 X3               10.000000      1.000000     Upper BC
 4 X4               20.000000      1.000000
 5 X5            -1.11022E-16             0     Lower BC

              Value of Objective Function = 30
```

Finding the maximum flow through a network is equivalent to solving a simple linear optimization problem, and for large problems, the LP procedure or the NETFLOW procedure of the SAS/OR product can be used. On the other hand, finding a traffic pattern that minimizes the total delay to move F vehicles per minute from node 1 to node 4 includes nonlinearities that need nonlinear optimization techniques. As traffic volume increases, speed decreases. Let t_{ij} be the travel time on arc (i, j) and assume that the following formulas describe the travel time as decreasing functions of the amount of traffic:

$$\begin{aligned}
t_{12} &= 5 + 0.1x_{12}/(1 - x_{12}/10) \\
t_{13} &= x_{13}/(1 - x_{13}/30) \\
t_{32} &= 1 + x_{32}/(1 - x_{32}/10) \\
t_{24} &= x_{24}/(1 - x_{24}/30) \\
t_{34} &= 5 + x_{34}/(1 - x_{34}/10)
\end{aligned}$$

These formulas use the road capacities (upper bounds), and you can assume that $F = 5$ vehicles per minute have to be moved through the network. The objective is now to minimize

$$f = f(x) = t_{12}x_{12} + t_{13}x_{13} + t_{32}x_{32} + t_{24}x_{24} + t_{34}x_{34}$$

The constraints are

$$\begin{array}{rcl} 0 \leq & x_{12}, x_{32}, x_{34} & \leq 10 \\ 0 \leq & x_{13}, x_{24} & \leq 30 \end{array}$$

$$\begin{array}{rcl} x_{13} & = & x_{32} + x_{34} \\ x_{24} & = & x_{12} + x_{32} \\ x_{24} + x_{34} & = & F = 5 \end{array}$$

In the following code, the NLPNRR subroutine is used to solve the minimization problem:

```
proc iml;
   title 'Minimize Total Delay in Network';
   start MINDEL(x);
      t12 = 5. + .1 * x[1] / (1. - x[1] / 10.);
      t13 = x[2] / (1. - x[2] / 30.);
      t32 = 1. + x[3] / (1. - x[3] / 10.);
      t24 = x[4] / (1. - x[4] / 30.);
      t34 = 5. + .1 * x[5] / (1. - x[5] / 10.);
      f = t12*x[1] + t13*x[2] + t32*x[3] + t24*x[4] + t34*x[5];
      return(f);
   finish MINDEL;

   con = {  0.  0.  0.  0.  0.   .   . ,
           10. 30. 10. 30. 10.   .   . ,
            0.  1. -1.  0. -1.  0.  0. ,
            1.  0.  1. -1.  0.  0.  0. ,
            0.  0.  0.  1.  1.  0.  5. };

   x = j(1,5, 1.);
   optn = {0 3};
   call nlpnrr(xres,rc,"MINDEL",x,optn,con);
```

The optimal solution is shown in the following output.

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

     1 X1                2.500001       5.777778
     2 X2                2.499999       5.702478
     3 X3           5.551115E-17        1.000000     Lower BC
     4 X4                2.500001       5.702481
     5 X5                2.499999       5.777778

             Value of Objective Function = 40.303030303
```

The active constraints and corresponding Lagrange multiplier estimates (costs) are shown in the following output.

```
                    Linear Constraints Evaluated at Solution

  [1] ACT          0  =  0 + 1.0000 * X2 - 1.0000 * X3 - 1.0000 * X5

  [2] ACT 4.4409E-16  =  0 + 1.0000 * X1 + 1.0000 * X3 - 1.0000 * X4

  [3] ACT          0  = -5.0000 + 1.0000 * X4 + 1.0000 * X5



                          First Order Lagrange Multipliers


                                                   Lagrange
                       Active Constraint         Multiplier

                    Lower BC       X3              0.924702
                    Linear EC      [1]             5.702479
                    Linear EC      [2]             5.777777
                    Linear EC      [3]            11.480257
```

Example 11.3. Compartmental Analysis

Numerical Considerations

An important class of nonlinear models involves a dynamic description of the response rather than an explicit description. These models arise often in chemical kinetics, pharmacokinetics, and ecological compartmental modeling. Two examples are presented in this section. Refer to Bates and Watts (1988) for a more general introduction to the topic.

In this class of problems, function evaluations, as well as gradient evaluations, are not done in full precision. Evaluating a function involves the numerical solution of a differential equation with some prescribed precision. Therefore, two choices exist for evaluating first- and second-order derivatives:

- differential equation approach
- finite difference approach

In the differential equation approach, the components of the Hessian and the gradient are written as a solution of a system of differential equations that can be solved simultaneously with the original system. However, the size of a system of differential equations, n, would suddenly increase to n^2+2n. This huge increase makes the finite difference approach an easier one.

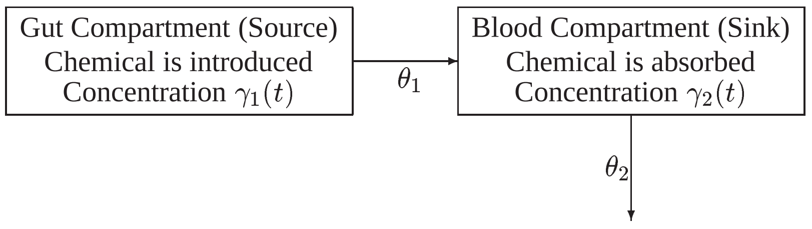
With the finite difference approach, a very delicate balance of all the precision requirements of every routine must exist. In the examples that follow, notice the relative levels of precision that are imposed on different modules. Since finite differences are used to compute the first- and second-order derivatives, it is incorrect to set the precision of the ODE solver at a coarse level because that would render the numerical estimation finite difference worthless.

A coarse computation of the solution of the differential equation cannot be accompanied by very fine computation of the finite difference estimates of the gradient and the Hessian. That is, you cannot set the precision of the differential equation solver to be 1E−4 and perform the finite difference estimation with a precision of 1E−10. In addition, this precision must be well-balanced with the termination criteria imposed on the optimization process.

In general, if the precision of the function evaluation is $O(\epsilon)$, the gradient should be computed by finite differences $O(\sqrt{\epsilon})$, and the Hessian should be computed with finite differences $O(\epsilon^{\frac{1}{3}})$. [*]

Diffusion of Tetracycline

Consider the concentration of tetracycline hydrochloride in blood serum. The tetracycline is administered to a subject orally, and the concentration of the tetracycline in the serum is measured. The biological system to be modeled will consist of two compartments: a gut compartment in which tetracycline is injected and a blood compartment that absorbs the tetracycline from the gut compartment for delivery to the body. Let $\gamma_1(t)$ and $\gamma_2(t)$ be the concentrations at time t in the gut and the serum, respectively. Let θ_1 and θ_2 be the transfer parameters. The model is depicted as follows.



The rates of flow of the drug are described by the following pair of ordinary differential equations:

$$
\begin{aligned}
\frac{d\gamma_1(t)}{dt} &= -\theta_1\gamma_1(t) \\
\frac{d\gamma_2(t)}{dt} &= \theta_1\gamma_1(t) - \theta_2\gamma_2(t)
\end{aligned}
$$

[*] In Release 6.09 and in later releases, you can specify the step size h in the finite difference formulas.

The initial concentration of the tetracycline in the gut is unknown, and while the concentration in the blood can be measured at all times, initially it is assumed to be zero. Therefore, for the differential equation, the initial conditions will be given by

$$\begin{aligned}\gamma_1(0) &= \theta_3 \\ \gamma_2(0) &= 0\end{aligned}$$

Also, a nonnegativity constraint is imposed on the parameters θ_1, θ_2, and θ_3, although for numerical purposes, you may need to use a small value instead of zero for these bounds (such as 1E$-$7).

Suppose y_i is the observed serum concentration at time t_i. The parameters are estimated by minimizing the sum of squares of the differences between the observed and predicted serum concentrations:

$$\sum_i (y_i - \gamma_2(t_i))^2$$

The following IML program illustrates how to combine the NLPDD subroutine and the ODE subroutine to estimate the parameters $(\theta_1, \theta_2, \theta_3)$ of this model. The input data are the measurement time and the concentration of the tetracycline in the blood. For more information on the ODE call, see the “ODE Call” section on page 670.

```
data tetra;
   input t c @@;
   datalines;
 1 0.7   2 1.2   3 1.4   4 1.4   6 1.1
 8 0.8  10 0.6  12 0.5  16 0.3
;

proc iml;
   use tetra;
   read all into tetra;
   start f(theta) global(thmtrx,t,h,tetra,eps);
      thmtrx = ( -theta[1] || 0 )     //
               (  theta[1] || -theta[2] );
      c = theta[3]//0 ;
      t = 0 // tetra[,1];
      call ode( r1, "der",c , t, h) j="jac" eps=eps;
      f = ssq((r1[2,])‘-tetra[,2]);
      return(f);
   finish;

   start der(t,x) global(thmtrx);
      y = thmtrx*x;
      return(y);
   finish;

   start jac(t,x) global(thmtrx);
      y = thmtrx;
      return(y);
```

```
finish;

h      = {1.e-14 1. 1.e-5};
opt    = {0 2 0 1 };
tc     = repeat(.,1,12);
tc[1]  = 100;
tc[7]  = 1.e-8;
par    = { 1.e-13 . 1.e-10 . . . . };
con    = j(1,3,0.);
itheta = { .1  .3  10};
eps    = 1.e-11;

call nlpdd(rc,rx,"f",itheta) blc=con opt=opt tc=tc par=par;
```

The output from the optimization process is shown in Output 11.3.1.

Output 11.3.1. Printed Output for Tetracycline Diffusion Problem

```
                        Optimization Start
                       Parameter Estimates
                                       Gradient           Lower      Upper
                                      Objective           Bound      Bound
     N Parameter         Estimate      Function      Constraint      Constraint

     1 X1                0.100000      76.48208               0      .
     2 X2                0.300000     -48.32095               0      .
     3 X3               10.000000       1.66610               0      .


             Value of Objective Function = 4.1469872335




                     Double Dogleg Optimization

     Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)

                     Without Parameter Scaling
              Gradient Computed by Finite Differences

                Parameter Estimates              3
                Lower Bounds                     3
                Upper Bounds                     0

                       Optimization Start

Active Constraints             0  Objective Function   4.1469872326
Max Abs Gradient       76.543381  Radius                          1
Element


                                            Max Abs          Slope
      Rest  Func Act   Objective  Obj Fun Gradient          Search
 Iter  arts Calls Con    Function   Change  Element  Lambda   Direc

    1     0     5   0     3.12117   1.0258    124.3  67.129  -8.023
    2     0     6   0     0.89524   2.2259  14.1471   1.885  -5.021
    3     0     7   0     0.32333   0.5719   3.7144   1.186  -0.786
    .
    .
    .
   31     0    38   0     0.03565 4.24E-11 3.196E-6       0 -18E-12
```

Output 11.3.1. (continued)

```
                         Optimization Results

Iterations                     31  Function Calls                 39
Gradient Calls                 33  Active Constraints              0
Objective Function    0.035648021  Max Abs Gradient      3.195746E-6
                                   Element



                        Optimization Results

Slope of Search      -1.76538E-11  Radius              1
Direction

GCONV convergence criterion satisfied.



                        Optimization Results
                         Parameter Estimates
                                              Gradient
                                             Objective
             N Parameter         Estimate        Function

             1 X1                0.182440        -0.00251
             2 X2                0.436010         0.00122
             3 X3                6.020476        -0.0001875

             Value of Objective Function = 0.0356480211
```

The differential equation model is linear, and in fact, it can be solved using an eigenvalue decomposition (this is not always feasible without complex arithmetic). Alternately, the availability and the simplicity of the closed form representation of the solution enables you to replace the solution produced by the ODE routine with the simpler and faster analytical solution. Closed forms are not expected to be easily available for nonlinear systems of differential equations, which is why the preceding solution was introduced.

The closed form of the solution requires a change to the function $f(\cdot)$. The functions needed as arguments of the ODE routine, namely the *der* and *jac* modules, can be removed.

```
   start f(th) global(theta,tetra);
      theta = th;
      vv    = v(tetra[,1]);
      error = ssq(vv-tetra[,2]);
      return(error);
   finish;

   start v(t) global(theta);
      v = theta[3]*theta[1]/(theta[2]-theta[1])*
              (exp(-theta[1]*t)-exp(-theta[2]*t));
      return(v);
   finish;
```

```
call nlpdd(rc,rx,"f",itheta) blc=con opt=opt tc=tc par=par;
```

The parameter estimates, which are shown in Output 11.3.2, are close to those obtained by the first solution.

Output 11.3.2. Second Set of Parameter Estimates for Tetracycline Diffusion

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

      1 X1                0.183025    -0.000003196
      2 X2                0.434482     0.000002274
      3 X3                5.995241    -0.000001035

      Value of Objective Function = 0.0356467763
```

Because of the nature of the closed form of the solution, you may want to add an additional constraint to guarantee that $\theta_2 \neq \theta_1$ at any time during the optimization. This will prevent a possible division by 0 or a value near 0 in the execution of the $v(\cdot)$ function. For example, you might add the constraint

$$\theta_2 - \theta_1 \geq 10^{-7}$$

Chemical Kinetics of Pyrolysis of Oil Shale

Pyrolysis is a chemical change effected by the action of heat, and this example considers the pyrolysis of oil shale described in Ziegel and Gorman (1980). Oil shale contains organic material that is bonded to the rock. To extract oil from the rock, heat is applied, and the organic material is decomposed into oil, bitumen, and other by-products. The model is given by

$$\begin{aligned}
\frac{d\gamma_1(t)}{dt} &= -(\theta_1 + \theta_4)\gamma_1(t)\iota(t,\theta_5) \\
\frac{d\gamma_2(t)}{dt} &= [\theta_1\gamma_1(t) - (\theta_2 + \theta_3)\gamma_2(t)]\iota(t,\theta_5) \\
\frac{d\gamma_3(t)}{dt} &= [\theta_4\gamma_1(t) + \theta_2\gamma_2(t)]\iota(t,\theta_5)
\end{aligned}$$

with the initial conditions

$$\gamma_1(t) = 100, \quad \gamma_2(t) = 0, \quad \gamma_3(t) = 0$$

A dead time is assumed to exist in the process. That is, no change occurs up to time θ_5. This is controlled by the indicator function $\iota(t,\theta_5)$, which is given by

$$\iota(t,\theta_5) = \begin{cases} 0 & \text{if } t < \theta_5 \\ 1 & \text{if } t \geq \theta_5 \end{cases}$$

where $\theta_5 \geq 0$. Only one of the cases in Ziegel and Gorman (1980) is analyzed in this report, but the others can be handled in a similar manner. The following IML program illustrates how to combine the NLPQN subroutine and the ODE subroutine to estimate the parameters θ_i in this model:

```
data oil ( drop=temp);
   input temp time bitumen oil;
   datalines;
673     5      0.      0.
673     7      2.2     0.
673    10     11.5     0.7
673    15     13.7     7.2
673    20     15.1    11.5
673    25     17.3    15.8
673    30     17.3    20.9
673    40     20.1    26.6
673    50     20.1    32.4
673    60     22.3    38.1
673    80     20.9    43.2
673   100     11.5    49.6
673   120      6.5    51.8
673   150      3.6    54.7
;

proc iml;
   use oil;
   read all into a;

/****************************************************************/
/* The INS function inserts a value given by FROM into a vector */
/* given by INTO, sorts the result, and posts the global matrix */
/* that can be used to delete the effects of the point FROM.    */
/****************************************************************/
   start ins(from,into) global(permm);
      in    =  into // from;
      x     =  i(nrow(in));
      permm = inv(x[rank(in),]);
      return(permm*in);
   finish;

   start der(t,x) global(thmtrx,thet);
      y     = thmtrx*x;
      if ( t <= thet[5] )  then y = 0*y;
      return(y);
   finish;

   start jac(t,x) global(thmtrx,thet);
      y     = thmtrx;
      if ( t <= thet[5] )  then y = 0*y;
      return(y);
   finish;

   start f(theta) global(thmtrx,thet,time,h,a,eps,permm);
   thet = theta;
   thmtrx = (-(theta[1]+theta[4]) ||        0           || 0 )//
            (theta[1]              || -(theta[2]+theta[3]) || 0 )//
            (theta[4]              || theta[2]            || 0 );
   t = ins( theta[5],time);
   c = { 100, 0, 0};
   call ode( r1, "der",c , t , h) j="jac" eps=eps;

/* send the intermediate value to the last column */
   r = (c ||r1) * permm;
   m = r[2:3,(2:nrow(time))];
   mm = m‘- a[,2:3];
   call qr(q,r,piv,lindep,mm);
   v = det(r);
   return(abs(v));
finish;

opt = {0 2 0 1 };
tc = repeat(.,1,12);
tc[1] = 100;
tc[7] = 1.e-7;
par = { 1.e-13 . 1.e-10 . . . .};
con = j(1,5,0.);
h = {1.e-14 1. 1.e-5};
time = (0 // a[,1]);
eps = 1.e-5;
itheta = { 1.e-3 1.e-3 1.e-3 1.e-3 1.};

call nlpqn(rc,rx,"f",itheta)  blc=con opt=opt tc=tc par=par;
```

The parameter estimates are shown in Output 11.3.3.

Output 11.3.3. Parameter Estimates for Oil Shale Pyrolysis

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

 1 X1                0.013692                  150.14987
 2 X2                0.012939                  248.78071
 3 X3                0.016303                 -144.46645
 4 X4                0.006638                 -318.57862
 5 X5                1.044177                   -3.16737

 Value of Objective Function = 85.597262124
```

Again, compare the solution using the approximation produced by the ODE subroutine to the solution obtained through the closed form of the given differential equation. Impose the following additional constraint to avoid a possible division by 0 when evaluating the function:

$$\theta_2 + \theta_3 - \theta_1 - \theta_4 \geq 10^{-7}$$

The closed form of the solution requires a change in the function $f(\cdot)$. The functions needed as arguments of the ODE routine, namely the `der` and `jac` modules, can be removed.

```
start f(thet) global(time,a);
   do i = 1 to nrow(time);
      t   = time[i];
      v1  = 100;
      if ( t >= thet[5] ) then
         v1 = 100*ev(t,thet[1],thet[4],thet[5]);
      v2 = 0;
      if ( t >= thet[5] ) then
         v2 = 100*thet[1]/(thet[2]+thet[3]-thet[1]-thet[4])*
               (ev(t,thet[1],thet[4],thet[5])-
                ev(t,thet[2],thet[3],thet[5]));
      v3 = 0;
      if ( t >= thet[5] ) then
         v3  = 100*thet[4]/(thet[1]+thet[4])*
           (1. - ev(t,thet[1],thet[4],thet[5])) +
           100*thet[1]*thet[2]/(thet[2]+thet[3]-thet[1]-thet[4])*(
           (1.-ev(t,thet[1],thet[4],thet[5]))/(thet[1]+thet[4]) -
           (1.-ev(t,thet[2],thet[3],thet[5]))/(thet[2]+thet[3])    );
      y = y // (v1 || v2 || v3);
   end;
   mm = y[,2:3]-a[,2:3];
   call qr(q,r,piv,lindep,mm);
   v = det(r);
   return(abs(v));
finish;

start ev(t,a,b,c);
   return(exp(-(a+b)*(t-c)));
finish;

con     = { 0.  0.  0.  0.   .   .  . ,
             .   .   .   .   .    . . ,
            -1   1   1  -1   .   1  1.e-7 };
time    =  a[,1];
par     = { 1.e-13 . 1.e-10 . . . .};
itheta  = { 1.e-3 1.e-3 1.e-2 1.e-3 1.};

call nlpqn(rc,rx,"f",itheta)  blc=con opt=opt tc=tc par=par;
```

The parameter estimates are shown in Output 11.3.4.

Output 11.3.4. Second Set of Parameter Estimates for Oil Shale Pyrolysis

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

       1 X1                0.017178             -0.005291
       2 X2                0.008912              0.002413
       3 X3                0.020007             -0.000520
       4 X4                0.010494             -0.002890
       5 X5                7.771534           0.000003217

       Value of Objective Function = 20.689350642
```

Example 11.4. MLEs for Two-Parameter Weibull Distribution

This example considers a data set given in Lawless (1982). The data are the number of days it took rats painted with a carcinogen to develop carcinoma. The last two observations are censored. Maximum likelihood estimates (MLEs) and confidence intervals for the parameters of the Weibull distribution are computed. In the following code, the data set is given in the vector CARCIN, and the variables P and M give the total number of observations and the number of uncensored observations. The set D represents the indices of the observations.

```
proc iml;
   carcin = { 143  164  188  188  190  192  206
              209  213  216  220  227  230  234
              246  265  304  216  244 };
   p = ncol(carcin); m = p - 2;
```

The three-parameter Weibull distribution uses three parameters: a scale parameter, a shape parameter, and a location parameter. This example computes MLEs and corresponding 95% confidence intervals for the scale parameter, σ, and the shape parameter, c, for a constant value of the location parameter, $\theta = 0$. The program can be generalized to estimate all three parameters. Note that Lawless (1982) denotes σ, c, and θ by α, β, and μ, respectively.

The observed likelihood function of the three-parameter Weibull distribution is

$$L(\theta,\sigma,c) = \frac{c^m}{\sigma^m} \prod_{i\in D} \left(\frac{t_i - \theta}{\sigma}\right)^{c-1} \prod_{i=1}^{p} \exp\left\{-\left(\frac{t_i-\theta}{\sigma}\right)^c\right\}$$

and the log likelihood, $\ell(\theta,\sigma,c) = \log L(\theta,\sigma,c)$, is

$$\ell(\theta,\sigma,c) = m\log c - mc\log\sigma + (c-1)\sum_{i\in D}\log(t_i-\theta) - \sum_{i=1}^{p}\left(\frac{t_i-\theta}{\sigma}\right)^c$$

The log-likelihood function, $\ell(\theta,\sigma,c)$, for $\theta = 0$ is the objective function to be maximized to obtain the MLEs $(\hat{\sigma},\hat{c})$:

```
start f_weib2(x) global(carcin,thet);
   /* x[1]=sigma and x[2]=c */
   p = ncol(carcin); m = p - 2;
   sum1 = 0.; sum2 = 0.;
   do i = 1 to p;
      temp = carcin[i] - thet;
      if i <= m then sum1 = sum1 + log(temp);
      sum2 = sum2 + (temp / x[1])##x[2];
   end;
   f = m*log(x[2]) - m*x[2]*log(x[1]) + (x[2]-1)*sum1 - sum2;
   return(f);
finish f_weib2;
```

The derivatives of ℓ with respect to the parameters θ, σ, and c are given in Lawless (1982). The following code specifies a gradient module, which computes $\partial\ell/\partial\sigma$ and $\partial\ell/\partial c$:

```
start g_weib2(x) global(carcin,thet);
   /* x[1]=sigma and x[2]=c */
   p = ncol(carcin); m = p - 2;
   g = j(1,2,0.);
   sum1 = 0.; sum2 = 0.; sum3 = 0.;
   do i = 1 to p;
      temp = carcin[i] - thet;
      if i <= m then sum1 = sum1 + log(temp);
      sum2 = sum2 + (temp / x[1])##x[2];
      sum3 = sum3 + ((temp / x[1])##x[2]) * (log(temp / x[1]));
   end;
   g[1] = -m * x[2] / x[1] + sum2 * x[2] / x[1];
   g[2] = m / x[2] - m * log(x[1]) + sum1 - sum3;
   return(g);
finish g_weib2;
```

The MLEs are computed by maximizing the objective function with the trust-region algorithm in the NLPTR subroutine. The following code specifies starting values for the two parameters, $c = \sigma = 0.5$, and to avoid infeasible values during the optimization process, it imposes lower bounds of $c, \sigma >= 10^{-6}$. The optimal parameter values are saved in the variable XOPT, and the optimal objective function value is saved in the variable FOPT.

```
n = 2; thet = 0.;
x0 = j(1,n,.5);
optn = {1 2};
con = { 1.e-6 1.e-6 ,
          .     .    };
call nlptr(rc,xres,"f_weib2",x0,optn,con,,,,"g_weib2");
/*--- Save result in xopt, fopt ---*/
xopt = xres‘; fopt = f_weib2(xopt);
```

The results shown in Output 11.4.1 are the same as those given in Lawless (1982).

Output 11.4.1. Parameter Estimates for Carcinogen Data

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

         1 X1              234.318611    1.3363283E-9
         2 X2                6.083147    -7.850915E-9

         Value of Objective Function = -88.23273515
```

The following code computes confidence intervals based on the asymptotic normal distribution. These will be compared with the profile-likelihood-based confidence intervals computed in the next example. The diagonal of the inverse Hessian (as calculated by the NLPFDD subroutine) is used to calculate the standard error.

```
call nlpfdd(f,g,hes2,"f_weib2",xopt,,"g_weib2");
hin2 = inv(hes2);
/* quantile of normal distribution */
prob = .05;
noqua = probit(1. - prob/2);
stderr = sqrt(abs(vecdiag(hin2)));
xlb = xopt - noqua * stderr;
xub = xopt + noqua * stderr;
print "Normal Distribution Confidence Interval";
print xlb xopt xub;
```

Output 11.4.2. Confidence Interval Based on Normal Distribution

```
        Normal Distribution Confidence Interval


                XLB      XOP2       XUB

          215.41298 234.31861 253.22425
          3.9894574 6.0831471 8.1768368
```

Example 11.5. Profile-Likelihood-Based Confidence Intervals

This example calculates confidence intervals based on the profile likelihood for the parameters estimated in the previous example. The following introduction on profile-likelihood methods is based on the paper of Venzon and Moolgavkar (1988).

Let $\hat{\theta}$ be the maximum likelihood estimate (MLE) of a parameter vector $\theta_0 \in \mathcal{R}^n$ and let $\ell(\theta)$ be the log-likelihood function defined for parameter values $\theta \in \Theta \subset \mathcal{R}^n$.

The profile-likelihood method reduces $\ell(\theta)$ to a function of a single parameter of interest, $\beta = \theta_j$, where $\theta = (\theta_1, \ldots, \theta_j, \ldots, \theta_n)'$, by treating the others as nuisance parameters and maximizing over them. The profile likelihood for β is defined as

$$\tilde{\ell}_j(\beta) = \max_{\theta \in \Theta_j(\beta)} \ell(\theta)$$

where $\Theta_j(\beta) = \{\theta \in \Theta : \theta_j = \beta\}$. Define the complementary parameter set $\omega = (\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_n)'$ and $\hat{\omega}(\beta)$ as the optimizer of $\tilde{\ell}_j(\beta)$ for each value of β. Of course, the maximum of function $\tilde{\ell}_j(\beta)$ is located at $\beta = \hat{\theta}_j$. The profile-likelihood-based confidence interval for parameter θ_j is defined as

$$\{\beta : \ell(\hat{\theta}) - \tilde{\ell}_j(\beta) \le \frac{1}{2} q_1(1-\alpha)\}$$

where $q_1(1-\alpha)$ is the $(1-\alpha)^{th}$ quantile of the χ^2 distribution with one degree of freedom. The points (β_l, β_u) are the endpoints of the profile-likelihood-based confidence interval for parameter $\beta = \theta_j$. The points β_l and β_u can be computed as the solutions of a system of n nonlinear equations $f_i(x)$ in n parameters, where $x = (\beta, \omega)$:

$$\begin{bmatrix} \ell(\theta) - \ell^* \\ \frac{\partial \ell}{\partial \omega}(\theta) \end{bmatrix} = 0$$

where ℓ^* is the constant threshold $\ell^* = \ell(\hat{\theta}) - \frac{1}{2} q_1(1-\alpha)$. The first of these n equations defines the locations β_l and β_u where the function $\ell(\theta)$ cuts ℓ^*, and the remaining $n-1$ equations define the optimality of the $n-1$ parameters in ω. Jointly, the n equations define the locations β_l and β_u where the function $\tilde{\ell}_j(\beta)$ cuts the constant threshold ℓ^*, which is given by the roots of $\tilde{\ell}_j(\beta) - \ell^*$. Assuming that the two solutions $\{\beta_l, \beta_u\}$ exist (they do not if the quantile $q_1(1-\alpha)$ is too large), this system of n nonlinear equations can be solved by minimizing the sum of squares of the n functions $f_i(\beta, \omega)$:

$$F = \frac{1}{2} \sum_{i=1}^{n} f_i^2(\beta, \omega)$$

For a solution of the system of n nonlinear equations to exist, the minimum value of the convex function F must be zero.

The following code defines the module for the system of $n = 2$ nonlinear equations to be solved:

```
start f_plwei2(x) global(carcin,ipar,lstar);
   /* x[1]=sigma, x[2]=c */
   like = f_weib2(x);
   grad = g_weib2(x);
   grad[ipar] = like - lstar;
   return(grad‘);
finish f_plwei2;
```

The following code implements the Levenberg-Marquardt algorithm with the NLPLM subroutine to solve the system of two equations for the left and right endpoints of the interval. The starting point is the optimizer $(\hat{\sigma}, \hat{c})$, as computed in the previous example, moved toward the left or right endpoint of the interval by an initial step (refer to Venzon and Moolgavkar 1988). This forces the algorithm to approach the specified endpoint.

```
/* quantile of chi**2 distribution */
chqua = cinv(1-prob,1); lstar = fopt - .5 * chqua;
optn = {2 0};
do ipar = 1 to 2;
/* Compute initial step: */
/* Choose (alfa,delt) to go in right direction */
/* Venzon & Moolgavkar (1988), p.89 */
   if ipar=1 then ind = 2; else ind = 1;
   delt = - inv(hes2[ind,ind]) * hes2[ind,ipar];
```

```
      alfa = - (hes2[ipar,ipar] - delt‘ * hes2[ind,ipar]);
      if alfa > 0 then alfa = .5 * sqrt(chqua / alfa);
      else do;
         print "Bad alpha";
         alfa = .1 * xopt[ipar];
      end;
      if ipar=1 then delt = 1 || delt;
         else delt = delt || 1;

   /* Get upper end of interval */
      x0 = xopt + (alfa * delt)‘;
   /* set lower bound to optimal value */
      con2 = con; con2[1,ipar] = xopt[ipar];
      call nlplm(rc,xres,"f_plwei2",x0,optn,con2);
      f = f_plwei2(xres); s = ssq(f);
      if (s < 1.e-6) then xub[ipar] = xres[ipar];
         else xub[ipar] = .;

   /* Get lower end of interval */
      x0 = xopt - (alfa * delt)‘;
   /* reset lower bound and set upper bound to optimal value */
      con2[1,ipar] = con[1,ipar]; con2[2,ipar] = xopt[ipar];
      call nlplm(rc,xres,"f_plwei2",x0,optn,con2);
      f = f_plwei2(xres); s = ssq(f);
      if (s < 1.e-6) then xlb[ipar] = xres[ipar];
         else xlb[ipar] = .;
   end;
   print "Profile-Likelihood Confidence Interval";
   print xlb xopt xub;
```

The results, shown in Output 11.5.1, are close to the results shown in Output 11.4.2.

Output 11.5.1. Confidence Interval Based on Profile Likelihood

```
               Profile-Likelihood Confidence Interval


                         XLB      XOP2       XUB

                    215.1963 234.31861  255.2157
                   4.1344126 6.0831471 8.3063797
```

Example 11.6. Survival Curve for Interval Censored Data

In some studies, subjects are assessed only periodically for outcomes or responses of interest. In such situations, the occurrence times of these events are not observed directly; instead they are known to have occurred within some time interval. The times of occurrence of these events are said to be *interval censored*. A first step in the analysis of these interval censored data is the estimation of the distribution of the event occurrence times.

In a study with n subjects, denote the raw interval censored observations by $\{(L_i, R_i] : 1 \leq i \leq n\}$. For the ith subject, the event occurrence time T_i lies in

$(L_i, R_i]$, where L_i is the last assessment time at which there was no evidence of the event, and R_i is the earliest time when a positive assessment was noted (if it was observed at all). If the event does not occur before the end of the study, R_i is given a value larger than any assessment time recorded in the data.

A set of nonoverlapping time intervals $I_j = (q_j, p_j],\ 1 \leq j \leq m$ is generated over which the survival curve $S(t) = \Pr[T > t]$ is estimated. Refer to Peto (1973) and Turnbull (1976) for details. Assuming the independence of T_i and $(L_i, R_i]$, and also independence across subjects, the likelihood of the data $\{T_i \in (L_i, R_i], 1 \leq i \leq n\}$ can be constructed in terms of the pseudo-parameters $\theta_j = \Pr[T_i \in I_j], 1 \leq i \leq m$. The conditional likelihood of $\theta = (\theta_1, \ldots, \theta_m)$ is

$$L(\theta) = \prod_{i=1}^{n} \left(\sum_{j=1}^{m} x_{ij}\theta_j \right)$$

where x_{ij} is 1 or 0 according to whether I_j is a subset of $(L_i, R_i]$. The maximum likelihood estimates, $\hat{\theta}_j, 1 \leq j \leq m$, yield an estimator $\hat{S}(t)$ of the survival function $S(t)$, which is given by

$$\hat{S}(t) = \begin{cases} 1 & t \leq q_1 \\ \sum_{i=j+1}^{m} \hat{\theta}_i & p_j \leq t \leq q_{j+1}, \quad 1 \leq j \leq m-1 \\ 0 & t \geq p_m \end{cases}$$

$\hat{S}(t)$ remains undefined in the intervals (q_j, p_j) where the function may decrease in an arbitrary way. The asymptotic covariance matrix of $\widehat{\theta}$ is obtained by inverting the estimated matrix of second partial derivatives of the negative log likelihood (Peto 1973, Turnbull 1976). You can then compute the standard errors of the survival function estimators by the delta method and approximate the confidence intervals for survival function by using normal distribution theory.

The following code estimates the survival curve for interval censored data. As an illustration, consider an experiment to study the onset of a special kind of palpable tumor in mice. Forty mice exposed to a carcinogen were palpated for the tumor every two weeks. The times to the onset of the tumor are interval censored data. These data are contained in the data set CARCIN. The variable L represents the last time the tumor was not yet detected, and the variable R represents the first time the tumor was palpated. Three mice died tumor free, and one mouse was tumor free by the end of the 48-week experiment. The times to tumor for these four mice were considered right censored, and they were given an R value of 50 weeks.

```
data carcin;
   input id l r @@;
   datalines;
   1  20  22    11  30  32     21  22  24    31  34  36
   2  22  24    12  32  34     22  22  24    32  34  36
   3  26  28    13  32  34     23  28  30    33  36  38
   4  26  28    14  32  34     24  28  30    34  38  40
```

```
       5  26  28    15  34  36     25  32  34    35  38  40
       6  26  28    16  36  38     26  32  34    36  42  44
       7  28  30    17  42  44     27  32  34    37  42  44
       8  28  30    18  30  50     28  32  34    38  46  48
       9  30  32    19  34  50     29  32  34    39  28  50
      10  30  32    20  20  22     30  32  34    40  48  50
       ;

   proc iml;
      use carcin;
      read all var{l r};
      nobs= nrow(l);
     /*********************************************************
         construct the nonoverlapping intervals (Q,P) and
         determine the number of pseudo-parameters (NPARM)
      *********************************************************/
      pp= unique(r); npp= ncol(pp);
      qq= unique(l); nqq= ncol(qq);
      q= j(1,npp, .);
      do;
         do i= 1 to npp;
            do j= 1 to nqq;
               if ( qq[j] < pp[i] ) then q[i]= qq[j];
            end;
            if q[i] = qq[nqq] then goto lab1;
         end;
      lab1:
      end;

      if i > npp then nq= npp;
      else            nq= i;
      q= unique(q[1:nq]);
      nparm= ncol(q);
      p= j(1,nparm, .);
      do i= 1 to nparm;
         do j= npp to 1 by -1;
            if ( pp[j] > q[i] ) then p[i]= pp[j];
         end;
      end;

     /********************************************************
         generate the X-matrix for the likelihood
      ********************************************************/
      _x= j(nobs, nparm, 0);
      do j= 1 to nparm;
         _x[,j]= choose(l <= q[j]  & p[j] <= r, 1, 0);
      end;

     /********************************************************
        log-likelihood function (LL)
      ********************************************************/
      start LL(theta) global(_x,nparm);
         xlt= log(_x * theta‘);
         f= xlt[+];
      return(f);
   finish LL;

  /*********************************************************
     gradient vector (GRAD)
   ********************************************************/
   start GRAD(theta) global(_x,nparm);
      g= j(1,nparm,0);
      tmp= _x # (1 /  (_x * theta‘) );
      g= tmp[+,];
      return(g);
   finish GRAD;

  /**************************************************************
     estimate the pseudo-parameters using quasi-newton technique
   *************************************************************/
   /* options */
   optn= {1 2};

   /* constraints */
   con= j(3, nparm + 2, .);
   con[1, 1:nparm]= 1.e-6;
   con[2:3, 1:nparm]= 1;
   con[3,nparm + 1]=0;
   con[3,nparm + 2]=1;

   /* initial estimates */
   x0= j(1, nparm, 1/nparm);

   /* call the optimization routine */
   call nlpqn(rc,rx,"LL",x0,optn,con,,,,"GRAD");

  /*************************************************************
    survival function estimate (SDF)
   ************************************************************/
   tmp1= cusum(rx[nparm:1]);
   sdf= tmp1[nparm-1:1];


  /*************************************************************
    covariance matrix of the first nparm-1 pseudo-parameters (SIGMA2)
   *************************************************************/
   mm= nparm - 1;
   _x= _x - _x[,nparm] * (j(1, mm, 1) || {0});
   h= j(mm, mm, 0);
   ixtheta= 1 / (_x * ((rx[,1:mm]) || {1})‘);
   if _zfreq then
      do i= 1 to nobs;
         rowtmp= ixtheta[i] # _x[i,1:mm];
         h= h + (_freq[i] # (rowtmp‘ * rowtmp));
      end;
   else do i= 1 to nobs;
      rowtmp= ixtheta[i] # _x[i,1:mm];
      h= h + (rowtmp‘ * rowtmp);
```

```
 end;
 sigma2= inv(h);

/**************************************************************
  standard errors of the estimated survival curve (SIGMA3)
 **************************************************************/
 sigma3= j(mm, 1, 0);
 tmp1= sigma3;
 do i= 1 to mm;
    tmp1[i]= 1;
    sigma3[i]= sqrt(tmp1‘ * sigma2 * tmp1);
 end;

/**************************************************************
  95% confidence limits for the survival curve (LCL,UCL)
 **************************************************************/
/* confidence limits */
 tmp1= probit(.975);
 *print tmp1;
 tmp1= tmp1 * sigma3;
 lcl= choose(sdf > tmp1, sdf - tmp1, 0);
 ucl= sdf + tmp1;
 ucl= choose( ucl > 1., 1., ucl);

/**************************************************************
  print estimates of pseudo-parameters
 **************************************************************/
 reset center noname;
 q= q‘;
 p= p‘;
 theta= rx‘;
 print ,"Parameter Estimates", ,q[colname={q}] p[colname={p}]
       theta[colname={theta} format=12.7],;

/**************************************************************
  print survival curve estimates and confidence limits
 **************************************************************/
 left= {0} // p;
 right= q // p[nparm];
 sdf= {1} // sdf // {0};
 lcl= {.} // lcl //{.};
 ucl= {.} // ucl //{.};
 print , "Survival Curve Estimates and 95% Confidence Intervals", ,
       left[colname={left}] right[colname={right}]
       sdf[colname={estimate} format=12.4]
       lcl[colname={lower} format=12.4]
       ucl[colname={upper} format=12.4];
```

The iteration history produced by the NLPQN subroutine is shown in Output 11.6.1

Output 11.6.1. Iteration History for the NLPQN Subroutine

```
                           Dual Quasi-Newton Optimization

             Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)

                        Parameter Estimates             12
                        Lower Bounds                    12
                        Upper Bounds                    12
                        Linear Constraints               1

                                 Optimization Start

Active Constraints          1  Objective Function          -93.3278404
Max Abs Gradient Element    65.361558529



                                            Objective Max Abs
                Func.   Active    Objective    Function  Gradient   Step
 Iter    Rest   Calls   Constr.   Function     Change    Element    Size   Slope

    1      0     3       1       -88.51201   4.8158    16.6594    0.0256 -305.2
    2      0     4       1       -87.42665   1.0854    10.8769    1.000  -2.157
    3      0     5       1       -87.27408   0.1526     5.4965    1.000  -0.366
    4      0     7       1       -87.17314   0.1009     2.2856    2.000  -0.113
    5      0     8       1       -87.16611   0.00703    0.3444    1.000  -0.0149
    6      0    10       1       -87.16582   0.000287   0.0522    1.001  -0.0006
    7      0    12       1       -87.16581   9.128E-6   0.00691   1.133  -161E-7
    8      0    14       1       -87.16581   1.712E-7   0.00101   1.128  -303E-9

                                Optimization Results

Iterations                           8  Function Calls                      15
Gradient Calls                      11  Active Constraints                   1
Objective Function         -87.16581343  Max Abs Gradient Element   0.0010060788
Slope of Search Direction  -3.033154E-7

NOTE:GCONV convergence criterion satisfied.
NOTE: At least one element of the (projected) gradient is greater than 1e-3.
```

The estimates of the pseudo-parameter for the nonoverlapping intervals are shown in Output 11.6.2.

Output 11.6.2. Estimates for the Probability of Event Occurrence

```
                      Parameter Estimates


                        Q         P        THETA

                       20        22    0.0499997
                       22        24    0.0749988
                       26        28    0.0999978
                       28        30    0.1033349
                       30        32    0.0806014
                       32        34    0.2418023
                       34        36    0.0873152
                       36        38    0.0582119
                       38        40    0.0582119
                       42        44    0.0873152
                       46        48    0.0291055
                       48        50    0.0291055
```

The survival curve estimates and confidence intervals are displayed in Output 11.6.3.

Output 11.6.3. Survival Estimates and Confidence Intervals

```
              Survival Curve Estimates and 95% Confidence Intervals



                LEFT     RIGHT     ESTIMATE        LOWER        UPPER

                   0        20       1.0000         .            .
                  22        22       0.9500       0.8825       1.0000
                  24        26       0.8750       0.7725       0.9775
                  28        28       0.7750       0.6456       0.9044
                  30        30       0.6717       0.5252       0.8182
                  32        32       0.5911       0.4363       0.7458
                  34        34       0.3493       0.1973       0.5013
                  36        36       0.2619       0.1194       0.4045
                  38        38       0.2037       0.0720       0.3355
                  40        42       0.1455       0.0293       0.2617
                  44        46       0.0582       0.0000       0.1361
                  48        48       0.0291       0.0000       0.0852
                  50        50       0.0000         .            .
```

In this program, the quasi-Newton technique is used to maximize the likelihood function. You can replace the quasi-Newton routine by other optimization routines, such as the NLPNRR subroutine, which performs Newton-Raphson ridge optimization, or the NLPCG subroutine, which performs conjugate gradient optimization. Depending on the number of parameters and the number of observations, these optimization routines do not perform equally well. For survival curve estimation, the quasi-Newton technique seems to work fairly well since the number of parameters to be estimated is usually not too large.

Example 11.7. A Two-Equation Maximum Likelihood Problem

This example and notation are taken from Bard (1974). A two-equation model is used to fit U.S. production data for the years 1909-1949, where z_1 is capital input, z_2 is labor input, z_3 is real output, z_4 is time in years (with 1929 as the origin), and z_5 is the ratio of price of capital services to wage scale.

```
proc iml;
   z={ 1.33135 0.64629 0.4026 -20 0.24447,
       1.39235 0.66302 0.4084 -19 0.23454,
       1.41640 0.65272 0.4223 -18 0.23206,
       1.48773 0.67318 0.4389 -17 0.22291,
       1.51015 0.67720 0.4605 -16 0.22487,
       1.43385 0.65175 0.4445 -15 0.21879,
       1.48188 0.65570 0.4387 -14 0.23203,
       1.67115 0.71417 0.4999 -13 0.23828,
       1.71327 0.77524 0.5264 -12 0.26571,
       1.76412 0.79465 0.5793 -11 0.23410,
       1.76869 0.71607 0.5492 -10 0.22181,
       1.80776 0.70068 0.5052  -9 0.18157,
       1.54947 0.60764 0.4679  -8 0.22931,
       1.66933 0.67041 0.5283  -7 0.20595,
        1.93377 0.74091 0.5994  -6 0.19472,
        1.95460 0.71336 0.5964  -5 0.17981,
        2.11198 0.75159 0.6554  -4 0.18010,
        2.26266 0.78838 0.6851  -3 0.16933,
        2.33228 0.79600 0.6933  -2 0.16279,
        2.43980 0.80788 0.7061  -1 0.16906,
        2.58714 0.84547 0.7567   0 0.16239,
        2.54865 0.77232 0.6796   1 0.16103,
        2.26042 0.67880 0.6136   2 0.14456,
        1.91974 0.58529 0.5145   3 0.20079,
        1.80000 0.58065 0.5046   4 0.18307,
        1.86020 0.62007 0.5711   5 0.18352,
        1.88201 0.65575 0.6184   6 0.18847,
        1.97018 0.72433 0.7113   7 0.20415,
        2.08232 0.76838 0.7461   8 0.18847,
        1.94062 0.69806 0.6981   9 0.17800,
        1.98646 0.74679 0.7722  10 0.19979,
        2.07987 0.79083 0.8557  11 0.21115,
        2.28232 0.88462 0.9925  12 0.23453,
        2.52779 0.95750 1.0877  13 0.20937,
        2.62747 1.00285 1.1834  14 0.19843,
        2.61235 0.99329 1.2565  15 0.18898,
        2.52320 0.94857 1.2293  16 0.17203,
        2.44632 0.97853 1.1889  17 0.18140,
        2.56478 1.02591 1.2249  18 0.19431,
        2.64588 1.03760 1.2669  19 0.19492,
        2.69105 0.99669 1.2708  20 0.17912 };
```

The two-equation model in five parameters $c_1, \ldots, c_5$ is

$$\begin{aligned} g_1 &= c_1 10^{c_2 z_4} [c_5 z_1^{-c_4} + (1-c_5) z_2^{-c_4}]^{-c_3/c_4} - z_3 = 0 \\ g_2 &= [\frac{c_5}{1-c_5}] \left(\frac{z_1}{z_2} \right)^{-1-c_4} - z_5 = 0 \end{aligned}$$

where the variables z_1 and z_2 are considered dependent (endogenous) and the variables z_3, z_4, and z_5 are considered independent (exogenous).

Differentiation of the two equations g_1 and g_2 with respect to the endogenous variables z_1 and z_2 yields the Jacobian matrix $\partial g_i / \partial z_j$ for $i = 1, 2$ and $j = 1, 2$, where i corresponds to rows (equations) and j corresponds to endogenous variables (refer to Bard 1974). You must consider parameter sets for which the elements of the Jacobian and the logarithm of the determinant cannot be computed. In such cases, the function module must return a missing value.

```
start fiml(pr) global(z);
   c1 = pr[1]; c2 = pr[2]; c3 = pr[3]; c4 = pr[4]; c5 = pr[5];
  /* 1. Compute Jacobian */
   lndet = 0 ;
```

```
      do t= 1 to 41;
         j11 = (-c3/c4) * c1 * 10 ##(c2 * z[t,4]) * (-c4) * c5 *
               z[t,1]##(-c4-1) * (c5 * z[t,1]##(-c4) + (1-c5) *
               z[t,2]##(-c4))##(-c3/c4 -1);
         j12 = (-c3/c4) * (-c4) * c1 * 10 ##(c2 * z[t,4]) * (1-c5) *
               z[t,2]##(-c4-1) * (c5 * z[t,1]##(-c4) + (1-c5) *
               z[t,2]##(-c4))##(-c3/c4 -1);
         j21 = (-1-c4)*(c5/(1-c5))*z[t,1]##( -2-c4)/ (z[t,2]##(-1-c4));
         j22 = (1+c4)*(c5/(1-c5))*z[t,1]##( -1-c4)/ (z[t,2]##(-c4));

         j = (j11 || j12 ) // (j21 || j22) ;
         if any(j = .) then detj = 0.;
            else detj = det(j);
         if abs(detj) < 1.e-30 then do;
            print t detj j;
            return(.);
         end;
         lndet = lndet + log(abs(detj));
      end;
```

Assuming that the residuals of the two equations are normally distributed, the likelihood is then computed as in Bard (1974). The following code computes the logarithm of the likelihood function:

```
     /* 2. Compute Sigma */
      sb = j(2,2,0.);
      do t= 1 to 41;
         eq_g1 = c1 * 10##(c2 * z[t,4]) * (c5*z[t,1]##(-c4)
                 + (1-c5)*z[t,2]##(-c4))##(-c3/c4) - z[t,3];
         eq_g2 = (c5/(1-c5)) * (z[t,1] / z[t,2])##(-1-c4) - z[t,5];
         resid = eq_g1 // eq_g2;
         sb = sb + resid * resid‘;
      end;
      sb = sb / 41;
     /* 3. Compute log L */
      const = 41. * (log(2 * 3.1415) + 1.);
      lnds = 0.5 * 41 * log(det(sb));
      logl = const - lndet + lnds;
      return(logl);
   finish fiml;
```

There are potential problems in computing the power and log functions for an unrestricted parameter set. As a result, optimization algorithms that use line search will fail more often than algorithms that restrict the search area. For that reason, the NLPDD subroutine is used in the following code to maximize the log-likelihood function:

```
   pr = j(1,5,0.001);
   optn = {0 2};
   tc = {. . . 0};
   call nlpdd(rc, xr,"fiml", pr, optn,,tc);
   print "Start" pr, "RC=" rc, "Opt Par" xr;
```

Part of the iteration history is shown in Output 11.7.1.

Output 11.7.1. Iteration History for Two-Equation ML Problem

```
                        Double Dogleg Optimization

        Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)

                        Without Parameter Scaling
                 Gradient Computed by Finite Differences

                   Parameter Estimates               5

                                         Optimization Start

Active Constraints                      0  Objective Function      909.72691311
Max Abs Gradient Element     41115.729089  Radius                             1



                                        Objective  Max Abs
              Func.   Active    Obj.    Function   Gradient
 Iter  Rest   Calls   Constr.   Func.   Change     Element    Lambda   Slope

    1   0       2     0       85.24836  824.5     3676.4    711.8     -71032
    2   0       7     0       45.14682  40.1015   3382.0    2881.2    -29.683
    3   0      10     0       43.46797  1.6788    208.4     95.020    -3.348
   35   0      64     0     -110.77858  5.68E-14  0.000111  41.795    -34E-17
   36   1     101     0     -110.77858  5.68E-14  0.000119  4E12      -32E-20
   36   2     145     0     -110.77858         0  0.000119  3.2E16    -46E-24

                                Optimization Results

Iterations                             36  Function Calls                     146
Gradient Calls                         41  Active Constraints                   0
Objective Function           -110.7785811  Max Abs Gradient Element     0.0001186267
Slope of Search Direction    -4.55096E-23  Radius                     3.771173E-19
```

The results are very close to those reported by Bard (1974). Bard also reports different approaches to the same problem that can lead to very different MLEs.

Output 11.7.2. Parameter Estimates

```
                                      Parameter Estimates
                                                       Gradient
                                                       Objective
                     N Parameter         Estimate        Function

                     1 X1                0.583884    -0.000004817
                     2 X2                0.005882     0.000011377
                     3 X3                1.362817    -0.000003229
                     4 X4                0.475091    -0.000018103
                     5 X5                0.447072        0.000119

                     Value of Objective Function = -110.7785811
```

Example 11.8. Time-Optimal Heat Conduction

This example illustrates a nontrivial application of the NLPQN algorithm that requires nonlinear constraints, which are specified by the *nlc* module. The example is listed as problem 91 in Hock & Schittkowski (1981). The problem describes a time-optimal heating process minimizing the simple objective function

$$f(x) = \sum_{j=1}^{n} x_j^2$$

subjected to a rather difficult inequality constraint:

$$c(x) = 10^{-4} - h(x) \geq 0$$

where $h(x)$ is defined as

$$\begin{aligned}
h(x) &= \int_0^1 \left(\sum_{i=1}^{30} \alpha_i(s)\rho_i(x) - k_0(s) \right)^2 ds \\
\alpha_i(s) &= \mu_i^2 A_i \cos(\mu_i s) \\
\rho_i(x) &= -\mu_i^2 \left[\exp\left(-\mu_i^2 \sum_{j=1}^{n} x_j^2 \right) - 2\exp\left(-\mu_i^2 \sum_{j=2}^{n} x_j^2 \right) + \cdots \right. \\
&\qquad \left. + (-1)^{n-1} 2\exp\left(-\mu_i^2 x_n^2 \right) + (-1)^n \right] \\
k_0(s) &= 0.5(1 - s^2) \\
A_i &= \frac{2\sin\mu_i}{\mu_i + \sin\mu_i \cos\mu_i}, \\
\mu &= (\mu_1, \ldots, \mu_{30})' \,, \quad \text{where } \mu_i \tan(\mu_i) = 1
\end{aligned}$$

The gradient of the objective function f, $g(x) = 2x$, is easily supplied to the NLPQN subroutine. However, the analytical derivatives of the constraint are not used; instead, finite difference derivatives are computed.

In the following code, the vector MU represents the first 30 positive values μ_i that satisfy $\mu_i \tan(\mu_i) = 1$:

```
proc iml;
   mu = { 8.6033358901938E-01 ,    3.4256184594817E+00 ,
          6.4372981791719E+00 ,    9.5293344053619E+00 ,
          1.2645287223856E+01 ,    1.5771284874815E+01 ,
          1.8902409956860E+01 ,    2.2036496727938E+01 ,
          2.5172446326646E+01 ,    2.8309642854452E+01 ,
          3.1447714637546E+01 ,    3.4586424215288E+01 ,
          3.7725612827776E+01 ,    4.0865170330488E+01 ,
          4.4005017920830E+01 ,    4.7145097736761E+01 ,
               5.0285366337773E+01 ,    5.3425790477394E+01 ,
               5.6566344279821E+01 ,    5.9707007305335E+01 ,
               6.2847763194454E+01 ,    6.5988598698490E+01 ,
               6.9129502973895E+01 ,    7.2270467060309E+01 ,
               7.5411483488848E+01 ,    7.8552545984243E+01 ,
               8.1693649235601E+01 ,    8.4834788718042E+01 ,
               8.7975960552493E+01 ,    9.1117161394464E+01  };
```

The vector $A = (A_1, \ldots, A_{30})'$ depends only on μ and is computed only once, before the optimization starts:

```
   nmu  = nrow(mu);
   a = j(1,nmu,0.);
   do i = 1 to nmu;
      a[i] = 2*sin(mu[i]) / (mu[i] + sin(mu[i])*cos(mu[i]));
   end;
```

The constraint is implemented with the QUAD subroutine, which performs numerical integration of scalar functions in one dimension. The subroutine calls the module `fquad` that supplies the integrand for $h(x)$. For details on the QUAD call, see the “QUAD Call” section on page 697.

```
  /* This is the integrand used in h(x) */
   start fquad(s) global(mu,rho);
      z = (rho * cos(s*mu) - 0.5*(1. - s##2))##2;
      return(z);
   finish;

  /* Obtain nonlinear constraint h(x) */
   start h(x) global(n,nmu,mu,a,rho);
      xx = x##2;
      do i= n-1 to 1 by -1;
         xx[i] = xx[i+1] + xx[i];
      end;
      rho = j(1,nmu,0.);
      do i=1 to nmu;
         mu2 = mu[i]##2;
         sum = 0; t1n = -1.;
         do j=2 to n;
            t1n = -t1n;
            sum = sum + t1n * exp(-mu2*xx[j]);
         end;
         sum = -2*sum + exp(-mu2*xx[1]) + t1n;
         rho[i] = -a[i] * sum;
      end;
      aint = do(0,1,.5);
      call quad(z,"fquad",aint) eps=1.e-10;
      v = sum(z);
      return(v);
   finish;
```

The modules for the objective function, its gradient, and the constraint $c(x) \geq 0$ are given in the following code:

```
  /* Define modules for NLPQN call: f, g, and c */
   start F_HS88(x);
      f = x * x‘;
      return(f);
   finish F_HS88;

   start G_HS88(x);
      g = 2 * x;
      return(g);
   finish G_HS88;

   start C_HS88(x);
      c = 1.e-4 - h(x);
      return(c);
   finish C_HS88;
```

The number of constraints returned by the *"nlc"* module is defined by *opt*$[10] = 1$. The ABSGTOL termination criterion (maximum absolute value of the gradient of the Lagrange function) is set by *tc*$[6] = 1\text{E}-4$.

```
   print ’Hock & Schittkowski Problem #91 (1981) n=5, INSTEP=1’;
   opt  = j(1,10,.);
   opt[2]=3;
   opt[10]=1;
   tc   = j(1,12,.);
   tc[6]=1.e-4;
   x0 = {.5 .5 .5 .5 .5};
   n = ncol(x0);
   call nlpqn(rc,rx,"F_HS88",x0,opt,,tc) grd="G_HS88" nlc="C_HS88";
```

Part of the iteration history and the parameter estimates are shown in Output 11.8.1.

Output 11.8.1. Iteration History and Parameter Estimates

```
                      Dual Quasi-Newton Optimization
             Modified VMCWD Algorithm of Powell (1978, 1982)

        Dual Broyden - Fletcher - Goldfarb - Shanno Update (DBFGS)
                Lagrange Multiplier Update of Powell(1982)
      Jacobian Nonlinear Constraints Computed by Finite Differences

                  Parameter Estimates                5
                  Nonlinear Constraints              1

                                        Optimization Start

Objective Function                      1.25  Max Constr. Violation  0.0952775105
Max Grad of the Lagran Func     1.1433393372


                                                      Maximum
                                                      Gradient
                                                       Element
                           Max.      Pred.             of the
            Func.  Obj.    Constr.   Func.    Step    Lagrange
Iter  Rest  Calls  Func.   Viol.     Red.     Size    Function

 1     0    3     0.81165  0.0869    1.7562   0.100   1.325
 2     0    4     0.18232  0.1175    0.6220   1.000   1.207
 3*    0    5     0.34567  0.0690    0.9321   1.000   0.639
 4     0    6     0.77700  0.0132    0.3498   1.000   1.329
 .
 .
 .
21     0   30     1.36266  8.02E-12  1.079E-6 1.000   0.00009

                           Optimization Results

Iterations                21  Function Calls                      31
Grad. Calls               23  Active Constraints                   1
Obj. Func.      1.3626568064  Max. Constr. Viol.         8.017286E-12
Max. Proj. Grad. 0.000096451  Value Lagrange Function    1.3626568149
Max. Grad. of the Lagran Func    0.0000887635  Slope  -1.079452E-6

NOTE:  ABSGCONV convergence criterion satisfied.

                      Optimization Results
                      Parameter Estimates

                                      Gradient        Gradient
                                      Objective        Lagrange
  N Parameter         Estimate        Function        Function

   1 X1               0.860296        1.720593     0.000030988
   2 X2           -0.000002262    -0.000004524    -0.000015387
   3 X3               0.643468        1.286936     0.000021570
   4 X4              -0.456614       -0.913227     0.000088763
   5 X5            0.000000904     0.000001807     0.000077409

           Value of Objective Function = 1.3626568064
           Value of Lagrange Function  = 1.3626568149
```

Problems 88 to 92 of Hock and Schittkowski (1981) specify the same optimization problem for $n = 2$ to $n = 6$. You can solve any of these problems with the preceding code by submitting a vector of length n as the initial estimate, x_0.

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,” *Journal of Optimization Theory and Applications*, 48, 359–377.

Anderson, B.D.O. and Moore, J.B. (1979), *Optimal Filtering*, New Jersey: Prentice-Hall.

Bard, Y. (1974), *Nonlinear Parameter Estimation*, New York: Academic Press.

Bates, D.M. and Watts, D.G. (1988), *Nonlinear Regression Analysis and Its Applications*, New York: John Wiley & Sons, Inc.

Beale, E.M.L. (1972), “A Derivation of Conjugate Gradients,” *Numerical Methods for Nonlinear Optimization*, ed. F. A. Lootsma, 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 & 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.

De Jong, P. (1988), “The Likelihood for a State Space Model,” *Biometrika*, 75, 165–169.

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 which 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.

Eskow, E. and Schnabel, R.B. (1991), “Algorithm 695: Software for a New Modified Cholesky Factorization,” *ACM Trans. Math. Software*, 17, 306–312.

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

Fletcher, R. and Powell, M.J.D. (1963), “A Rapidly Convergent Descent Method for Minimization,” *Computer Journal*, 6, 163–168.

Fletcher, R. and Xu, C. (1987), “Hybrid Methods for Nonlinear Least Squares,” *Journal of Numerical Analysis*, 7, 371–389.

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.

Hartmann, W. (1991), *The NLP Procedure: Extended User's Guide*, Releases 6.08 and 6.10, Cary, NC: SAS Institute Inc.

Hock, W. and Schittkowski, K. (1981), "Test Examples for Nonlinear Programming Codes," *Lecture Notes in Economics and Mathematical Systems 187*, 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 Models and Methods for Lifetime Data*, New York: John Wiley & 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 Linesearch Algorithm for Nonlinear Least-Squares Problems," *Mathematical Programming*, 29, 268−296.

Lütkepohl, H. (1991), *Introduction to Multiple Time Series Analysis*, Berlin: Springer-Verlag.

Moré, J.J. (1978), "The Levenberg-Marquardt Algorithm: Implementation and Theory," *Lecture Notes in Mathematics 630*, ed. G.A. Watson, 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 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," *Computer Journal*, 7, 308−313.

Peto, R. (1973), "Experimental Survival Curves for Interval-Censored Data," *Appl. Statist*, 22, 86−91.

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

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

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

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

Powell, J.M.D. (1982a), "Extensions to Subroutine VF02AD," *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," *Computer Journal*, 3, 175–184.

Schittkowski, K. (1978), "An Adaptive Precision Method for the Numerical Solution of Constrained Optimization Problems Applied to a Time-Optimal Heating Process," *Proceedings of the 8th IFIP Conference on Optimization Techniques*, Heidelberg, New York: Springer-Verlag.

Schittkowski, K. (1987), *More Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems 282*, Berlin, Heidelberg: 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.

Turnbull, B.W. (1976), "The Empirical Distribution Function from Arbitrarily Grouped, Censored and Truncated Data," *J. Royal Statist. Soc. Ser. B*, 38, 290–295.

Venzon, D.J. and Moolgavkar, S.H. (1988), "A Method for Computing Profile-Likelihood-Based Confidence Intervals," *Applied Statistics*, 37, 87–94.

Wedin, P.A. and Lindström, P. (1987), *Methods and Software for Nonlinear Least Squares Problems*, University of Umea, Report No. UMINF 133.87.

Ziegel, E. R. and J. W. Gorman (1980), "Kinetic Modelling with Multipurpose Data," *Technometrics*, 27, 352–357.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/IML User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. 846 pp.

SAS/IML User's Guide, Version 8

ISBN 1–58025–553–1

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

1st printing, October 1999