NLPL1: A Fortran Implementation of an SQP Algorithm for Minimizing Sums of Absolute Function Values - User's Guide -

Address: Prof. K. Schittkowski
Siedlerstr. 3
D - 95488 Eckersdorf
Germany

Phone: (+49) 921 32887

E-mail: klaus@schittkowski.de

Web: http://www.klaus-schittkowski.de

Date: December, 2009

Abstract

The Fortran subroutine NLPL1 solves constrained nonlinear programming problems, where the sum of absolute nonlinear function values is to be minimized. It is assumed that all functions are continuously differentiable. By introducing additional variables and nonlinear inequality constraints, the problem is transformed into a general smooth nonlinear program subsequently solved by the sequential quadratic programming (SQP) code NLPQLP. The usage of the code is documented, and two illustrative examples are presented.

Keywords: L_1 optimization, data fitting, SQP, sequential quadratic programming, nonlinear programming, numerical algorithms, Fortran codes

1 Introduction

L_1 optimization problems arise in many practical situations, for example in approximation or when fitting a model function to given data in the L_1-norm. In this particular case, a mathematical model is available in form of one or several equations, and the goal is to estimate some unknown parameters of the model. Exploited are available experimental data, to minimize the distance of the model function, in most cases evaluated at certain time values, from measured data at the same time values. An extensive discussion of data fitting especially in case of dynamical systems is given by Schittkowski [3], and the code NLPL1 is part of the software system EASY-FIT.

The mathematical problem we want to solve, is given in the form

$$x \in I\!R^n : \begin{array}{l} \min \sum_{i=1}^{l} |f_i(x)| \\ g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ , \\ x_l \leq x \leq x_u \ . \end{array} \tag{1}$$

It is assumed that f_1, …, f_l and g_1, …, g_m are continuously differentiable functions.

In this paper, we consider the question how an existing nonlinear programming code can be used to solve constrained L_1 optimization problems in an efficient and robust way after a suitable transformation. In a very similar way, also L_∞, min-max and least squares problems can be solved efficiently by an SQP code, see Schittkowski [3, 6, 8, 9].

The transformation of a min-max problem into a special nonlinear program is described in Section 2. Sections 3 to 5 contain a documentation of the Fortran code and two example implementations.

2 The Transformed Optimization Problem

We consider the constrained nonlinear L_1 problem (1), and introduce $2l$ additional variables z_1, …, z_{2l} and $2l$ additional nonlinear inequality constraints of the form

$$\begin{array}{rcl} f_i(x) + z_i & \geq & 0 \ , \\ -f_i(x) + z_{l+i} & \geq & 0 \ , \end{array} \tag{2}$$

$i = 1, \ldots, l$. The following equivalent problem is to be solved by an SQP method,

$$
\begin{array}{ll}
 & \min\ z \\
 & g_j(x) = 0 \quad ,\ j = 1, \ldots, m_e\ , \\
 & g_j(x) \geq 0 \quad ,\ j = m_e + 1, \ldots, m\ , \\
(x, z) \in I\!R^{n+2l} : & f_i(x) + z_i \geq 0 \quad ,\ i = 1, \ldots, l\ , \\
 & -f_i(x) + z_{l+i} \geq 0 \quad ,\ i = 1, \ldots, l\ , \\
 & x_l \leq x \leq x_n \quad , \\
 & z \geq 0 \quad .
\end{array}
\tag{3}
$$

In this case, the quadratic programming subproblem wich has to be solved in each step of an SQP method, has the form

$$
\begin{array}{ll}
 & \min\ \frac{1}{2}(d^T, e) B_k \begin{pmatrix} d \\ e \end{pmatrix} + e \\
 & \nabla g_j(x_k)^T d + g_j(x_k) = 0 \quad ,\ j = 1, \ldots, m_e\ , \\
 & \nabla g_j(x_k)^T d + g_j(x_k) \geq 0 \quad ,\ j = m_e + 1, \ldots, m\ , \\
(d, e) \in I\!R^{n+2l} : & \nabla f_i(x_k)^T d + e_i + f_i(x_k) + z_i^k \geq 0 \quad ,\ i = 1, \ldots, l\ , \\
 & -\nabla f_i(x_k)^T d + e_{l+i} - f_i(x_k) + z_{l+i}^k \geq 0 \quad ,\ i = 1, \ldots, l\ , \\
 & x_l - x_k \leq d \leq x_u - x_k \quad , \\
 & e \geq 0 \quad .
\end{array}
\tag{4}
$$

$B_k \in I\!R^{n+1} \times I\!R^{n+1}$ is a quasi-Newton update matrix of the Lagrangian function of (3). A new iterate is then obtained from

$$
x_{k+1} = x_k + \alpha_k d_k \quad , \quad z_{k+1} = z_k + \alpha_k e_k \ ,
$$

where $d_k \in I\!R^n$ and $e_k \in I\!R$ are a solution of (4) and α_k a steplength parameter obtained from forcing a sufficient descent of a merit function.

The proposed transformation (3) is independent of the SQP method used, so that available codes can be used in the form of a *black box*. Our implementation calls the code NLPQLP [7].

3 Calling Sequence

In this section, we describe the arguments of subroutine NLPL1 in detail.

Usage:

```
    CALL   NLPL1 (         M,         ME,    LMMAX,        L,         N,
/                       LLN1,    LLMNN2,        X,     FUNC,       RES,
/                       GRAD,         U,       XL,       XU,       ACC,
/                      ACCQP,    RESSIZ,   MAXFUN    MAXIT,     MAXNM,
/                       RHOB,    IPRINT,     IOUT,    IFAIL,        WA,
/                        LWA,       KWA,     LKWA,   LOGWA,    LLOGWA )
```

Definition of the parameters:

M :	Number of constraints, i.e., m.
ME :	Number of equality constraints, i.e., m_e.
LMMAX :	Row dimension of GRAD and dimension of FUNC. LMMAX must be at least one and not smaller than L + M.
L :	Number of terms in objective function, i.e., l.
N :	Number of variables, i.e., n.
LLN1 :	Dimensioning parameter for X, XL, and XU, must be equal to 2*L + N + 1.
LLMNN2 :	Dimensioning parameter, must be set to M + 2*N + 6*L + 2 when calling NLPL1.
X(LLN1) :	On input, the first N positions of X have to contain an initial guess for the solution. On return, X is replaced by the last computed iterate.
FUNC(LMMAX) :	Function values passed to NLPL1 by reverse communication, i.e., the first L positions contain the L residual values $f_i(x)$, $i = 1, \ldots, l$, the subsequent M coefficients the constraint values $g_j(x)$, $j = 1, \ldots ,m$.
RES :	On return, RES contains the final objective function value $\lvert f_1(x)\rvert + \ldots + \lvert f_l(x)\rvert$.
GRAD(LMMAX,N) :	The array is used to pass gradients of residuals and constraints to NLPL1 by reverse communication. In the driving program, the row dimension of GRAD must be equal to LMMAX. The first L rows contain L gradients of residual functions $\nabla f_i(x)$ at x, $i = 1, \ldots, l$, the subsequent M rows gradients of constraint functions $\nabla g_j(x)$, $j = 1, \ldots, m$.

U(LLMNN2) :	On return, U contains the multipliers with respect to the last computed iterate. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the following N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.
XL(LLN1), XU(LLN1) :	On input, the one-dimensional arrays XL and XU must contain the upper and lower bounds of the variables.
ACC :	The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be much smaller than the accuracy by which gradients are computed.
ACCQP :	The tolerance is passed to the QP solver to perform several tests, for example whether optimality conditions are satisfied or whether a number is considered as zero or not. If ACCQP is less or equal to zero, then the machine precision is computed by NLPQLP and subsequently multiplied by 10.0.
RESSIZE :	The user must indicate a guess for the approximate size of the objective function. RESSIZE must not be negative.
MAXFUN :	The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20).
MAXIT :	Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).
MAXNM :	Stack size for storing merit function values at previous iterations for non-monotone line search (e.g. 10).
RHOB :	Parameter for initializing a restart in case of IFAIL=2 by setting the BFGS-update matrix to rhob*I, where I denotes the identity matrix. The number of restarts is bounded by MAXFUN. No restart is performed if RHOB is set to zero. Must be non-negative (e.g. 100).
IPRINT :	Specification of the desired output level:

	0 - No output of the program.
	1 - Only final convergence analysis.
	2 - One line of intermediate results for each iteration.
	3 - More detailed information for each iteration.
	4 - More line search data displayed.
IOUT :	Integer indicating the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.
IFAIL :	The parameter shows the reason for terminating a solution process. Initially IFAIL must be set to zero. On return IFAIL could contain the following values:
	-2 - Compute new gradient values.
	-1 - Compute new function values.
	0 - Optimality conditions satisfied.
	1 - Stop after MAXIT iterations.
	2 - Uphill search direction.
	3 - Underflow when computing new BFGS-update matrix.
	4 - Line search exceeded MAXFUN iterations.
	5 - Length of a working array too short.
	6 - False dimensions, M>MMAX, N$\geq$NMAX, or MNN2$\neq$M+N+N+2.
	7 - Search direction close to zero at infeasible iterate.
	8 - Starting point violates lower or upper bound.
	9 - Wrong input parameter, e.g., MODE, IPRINT, IOUT.
	10 - Inconsistency in QP, division by zero.
	>100 - Error message of QP solver.
WA(LWA) :	WA is a real working array of length LWA.
LWA :	Length of the real working array WA. LWA must be at least 5*(N+2)*(N+2)/2 + MW*NMAX + 9*MW + 6*LMAX + 4*MMAX + 35*NMAX + 200, where MW = max(M+L,min(MOUT,M+L+L)). MOUT is an internal bound set to 500.
KWA(LKWA) :	KWA is an integer working array of length LKWA.
LKWA :	Length of the integer working array KWA. LKWA must be at least 2*MW + max(N+1,MW/NMAX) + 25. On return, KWA(1) and KWA(2) contain the number of function and derivative evaluations, respectively.

LOGWA(LLOGWA) :	Logical working array of length LLOGWA.
LLOGWA :	Length of the logical array LOGWA. The length LLOGWA of the logical array must be at least 9*L + 5*M + 11.

4 Program Organization

All declarations of real numbers must be done in double precision. Subroutine NLPL1 must be linked with the user-provided main program, the SQP code NLPQLP [7], and the quadratic programming code QL [5].

NLPL1 is implemented in form of a Fortran subroutine. Model functions and gradients are passed by reverse communication. The user has to provide functions and gradients in the same program which executes NLPL1, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in the first N positions of X.
2. Compute residual and constraint function values values, and store them in a one-dimensional double precision array FUNC. The first L positions contain the L residual values $f_i(x)$, $i = 1, \ldots, l$, the subsequent M coefficients the constraint values $g_j(x)$, $j = 1, \ldots, m$.
3. Compute gradients of residual and constraint functions, and store them in a two-dimensional double precision array GRAD. The first L rows contain gradients of residual functions $\nabla f_i(x)$ at x, $i = 1, \ldots, l$, the subsequent M rows gradients of constraint functions $\nabla g_j(x)$, $j = 1, \ldots, m$.
4. Set IFAIL=0 and execute NLPL1.
5. If NLPL1 returns with IFAIL=-1, compute residual function values and constraint values for the arguments found in X, and store them in FUNC in the order shown above. Then call NLPL1 again, but do not change IFAIL.
6. If NLPL1 terminates with IFAIL=-2, compute gradient values subject to variables stored in X, and store them in GRAD as indicated above. Then call NLPL1 again without changing IFAIL.
7. If NLPL1 terminates with IFAIL=0, the internal stopping criteria are satisfied. The variable values found in X are considered as a local solution of the min-max optimization problem.
8. In case of IFAIL>0, an error occurred.

If analytical derivatives are not available, additional function calls are required for gradient approximations, for example by forward differences, two-sided differences, or even higher order formulae.

Some of the termination reasons depend on the accuracy used for approximating gradients. If we assume that all functions and gradients are computed within machine precision and that the implementation is correct, there remain only the following possibilities that could cause an error message:

1. The termination parameter ACC is too small, so that the numerical algorithm plays around with round-off errors without being able to improve the solution. Especially the Hessian approximation of the Lagrangian function becomes unstable in this case. A straightforward remedy is to restart the optimization cycle again with a larger stopping tolerance.

2. The constraints are contradicting, i.e., the set of feasible solutions is empty. There is no way to find out, whether a general nonlinear and non-convex set possesses a feasible point or not. Thus, the nonlinear programming algorithms will proceed until running in any of the mentioned error situations. In this case, there the correctness of the model must be checked very carefully.

3. Constraints are feasible, but some of them there are degenerate, for example if some of the constraints are redundant. One should know that SQP algorithms require satisfaction of the so-called constraint qualification, i.e., that gradients of active constraints are linearly independent at each iterate and in a neighborhood of the optimal solution. In this situation, it is recommended to check the formulation of the model.

However, some of the error situations do also occur, if because of wrong or non-accurate gradients, the quadratic programming subproblem does not yield a descent direction for the underlying merit function. In this case, one should try to improve the accuracy of function evaluations, scale the model functions in a proper way, or start the algorithm from other initial values.

5 Examples

To give a simple example how to organize the code in case of two explicitly given functions, we consider Rosenbrock's banana function, see test problem TP1 of Hock and Schittkowski [1],

$$x_1, x_2 \in I\!R : \ \min |10(x_2 - x_1^2)| + |1 - x_1| \tag{5}$$

The Fortran source code for executing NLPL1 is listed below. Gradients are computed analytically.

```
      IMPLICIT          NONE
      INTEGER           N, M, L, LLMNN2, LMMAX, LLN1, LLM, LWA,
     /                  LKWA, LLOGWA
```

```
      PARAMETER        (L = 2, N = 2, M = 0)
      PARAMETER        (LMMAX  = M + L,
     /                  LLN1   = 2*L + N + 1,
     /                  LLM    = 2*L + M,
     /                  LLMNN2 = LLM+2*LLN1,
     /                  LWA    = 5*LLN1**2/2 + LLM*LLN1 + 35*LLN1
     /                           + 9*LLM + 150,
     /                  LKWA   = LLN1 + 25,
     /                  LLOGWA = 2*LLM + 10)
      INTEGER           ME, MAXFUN, MAXIT, IPRINT, MAXNM, IOUT, IFAIL,
     /                  KWA
      DOUBLE PRECISION  X, FUNC, RES, GRAD, U, XL, XU, ACC,
     /                  ACCQP, RESSIZ, RHOB, WA, EPS, T, Y, W
      DIMENSION         X(LLN1), FUNC(LMMAX), GRAD(LMMAX,N),
     /                  U(LLMNN2), XL(LLN1), XU(LLN1),
     /                  WA(LWA), KWA(LKWA), LOGWA(LLOGWA)
      LOGICAL           LOGWA
      EXTERNAL          QL
C
C   set parameters
C
      ME     = 0
      ACC    = 1.0D-8
      ACCQP  = ACC
      RESSIZ = 0.0D0
      RHOB   = 0.0D0
      MAXFUN = 20
      MAXIT  = 100
      MAXNM  = 0
      IPRINT = 2
      IOUT   = 6
      IFAIL  = 0
C
C   starting values and bounds
C
      X(1)  = -1.2D0
      XL(1) = -1.0D5
      XU(1) =  1.0D5
      X(2)  =  1.0D0
      XL(2) = -1.0D5
      XU(2) =  1.0D5
C
C   execute NLPL1 by reverse communication
C
    1 CONTINUE
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-1)) THEN
```

```
         FUNC(1) = 10.0D0*(X(2) - X(1)**2)
         FUNC(2) = 1.0D0 - X(1)
      ENDIF
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-2)) THEN
         GRAD(1,1) = -20.0D0*X(1)
         GRAD(1,2) =  10.0D0
         GRAD(2,1) = -1.0D0
         GRAD(2,2) =  0.0D0
      ENDIF
C
C   call NLPL1
C
      CALL NLPL1(M, ME, LMMAX, L, N, LLN1, LLMNN2, X, FUNC, RES,
     /            GRAD, U, XL, XU, ACC, ACCQP, RESSIZ, MAXFUN, MAXIT,
     /            MAXNM, RHOB, IPRINT, IOUT, IFAIL, WA, LWA, KWA,
     /            LKWA, LOGWA, LLOGWA, QL)
      IF (IFAIL.LT.0) GOTO 1
C
      STOP
      END
```

The following output should appear on screen:

```
--------------------------------------------------------------------
 START OF THE SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
--------------------------------------------------------------------

 Parameters:
    N      =       6
    M      =       4
    ME     =       0
    MODE   =       0
    ACC    =   0.1000D-07
    ACCQP  =   0.1000D-07
    STPMIN =   0.1000D-07
    MAXFUN =      20
    MAXNM  =       0
    MAXIT  =     100
    IPRINT =       2

 Output in the following order:
    IT    - iteration number
    F     - objective function value
    SCV   - sum of constraint violations
    NA    - number of active constraints
    I     - number of line search iterations
    ALPHA - steplength parameter
    DELTA - additional variable to prevent inconsistency
    KKT   - Karush-Kuhn-Tucker optimality criterion


  IT        F          SCV      NA  I    ALPHA     DELTA     KKT
--------------------------------------------------------------------
   1  0.00000000D+00  0.66D+01    4  0  0.00D+00  0.00D+00  0.79D+01
   2  0.16515464D+01  0.30D+01    3  1  0.10D+01  0.00D+00  0.33D+00
   3  0.13428041D+01  0.95D+00    3  1  0.10D+01  0.00D+00  0.14D+01
   4  0.77912492D+00  0.37D+01    3  2  0.42D+00  0.00D+00  0.12D+01
   5  0.49100689D+00  0.32D+01    3  2  0.37D+00  0.00D+00  0.48D+00
   6  0.14406132D-01  0.23D+01    2  1  0.10D+01  0.00D+00  0.32D-01
   7  0.00000000D+00  0.21D-02    3  1  0.10D+01  0.00D+00  0.52D-09
   8  0.00000000D+00  0.00D+00    4  1  0.10D+01  0.00D+00  0.00D+00
     *** WARNING: Zero search direction at feasible point.
                  The final iterate may not be optimal!

  --- Final Convergence Analysis at Last Iterate ---

    Objective function value:      F(X)  =  0.00000000D+00
    Solution values:               X     =
       0.10000000D+01  0.10000000D+01  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00
    Multiplier values:             U     =
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.10000000D+01  0.10000000D+01
       0.10000000D+01  0.10000000D+01  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
    Constraint values:             G(X)  =
       0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
    Distance from lower bound:     XL-X  =
      -0.10000100D+06 -0.10000100D+06  0.00000000D+00  0.00000000D+00
       0.00000000D+00  0.00000000D+00
    Distance from upper bound:     XU-X  =
       0.99999000D+05  0.99999000D+05  0.10000000D+31  0.10000000D+31
       0.10000000D+31  0.10000000D+31
    Number of function calls:      NFUNC =      10
    Number of gradient calls:      NGRAD =       8
    Number of calls of QP solver:  NQL   =       8


  --- Final Convergence Analysis of NLPL1 ---

    Maximum function value:        RES   =  0.00000000D+00
    Function values:               F(X) =
       0.00000000D+00  0.00000000D+00
    Solution:                    X =
       0.10000000D+01  0.10000000D+01
    Multiplier values:           U =
       0.00000000D+00  0.00000000D+00  0.10000000D+01  0.10000000D+01
    Number of function calls:      NFUNC =  10
    Number of derivative calls:    NGRAD =   8
```

The error message is due to the exact calculation of the optimal solution.

To present a data fitting example, we consider a model function

$$h(x,t) = \frac{x_1(t^2 + x_2 t)}{t^2 + x_3 t + x_4} \quad ,$$

$x = (x_1, \ldots, x_4)^T$. The data are shown in the code below. In addition, we have two equality constraints

$$h(x,t_1) - y_1 = 0 \quad , \quad h(x,t_{11}) - y_{11} = 0 \quad .$$

The code and the corresponding screen output follow.

```
      IMPLICIT          NONE
      INTEGER           N, M, L, LLN1, LLM, LLMNN2, LMMAX, LWA,
     /                  LKWA, LLOGWA
      PARAMETER        (L = 11, N = 4, M = 2)
      PARAMETER        (LMMAX  = M + L,
     /                  LLN1   = N + 2*L + 1,
     /                  LLM    = M + 2*L,
     /                  LLMNN2 = LLM + 2*LLN1,
     /                  LWA    = 5*LLN1**2/2 + LLM*LLN1 + 35*LLN1
     /                           + 9*LLM + 150,
     /                  LKWA   = LLN1 + 15,
     /                  LLOGWA = 2*LLM + 10)
      INTEGER           ME,MAXFUN, MAXIT, IPRINT, MAXNM, IOUT, IFAIL,
     /                  KWA, I, J
      DOUBLE PRECISION  X, FUNC, RES, GRAD, U, XL, XU, ACC,
     /                  ACCQP, RESSIZ, RHOB, WA, EPS, T, Y, W
      DIMENSION         X(LLN1), FUNC(LMMAX), GRAD(LMMAX,N),
     /                  U(LLMNN2), XL(LLN1), XU(LLN1),
     /                  WA(LWA), KWA(LKWA), LOGWA(LLOGWA),
     /                  T(L), Y(L), W(N)
      LOGICAL           LOGWA
      DATA              T/0.0625D0,0.0714D0,0.0823D0,0.1000D0,0.1250D0,
     /                    0.1670D0,0.2500D0,0.5000D0,1.0000D0,2.0000D0,
     /                    4.0000D0/
      DATA              Y/0.0246D0,0.0235D0,0.0323D0,0.0342D0,0.0456D0,
     /                    0.0627D0,0.0844D0,0.1600D0,0.1735D0,0.1947D0,
     /                    0.1957D0/
      EXTERNAL          QL
C
C   set parameters
C
      ME     = M
      ACC    = 1.0D-10
      ACCQP  = 1.0D-14
      RESSIZ = 1.0D0
      RHOB   = 0.0D0
      MAXFUN = 20
      MAXIT  = 100
      MAXNM  = 20
      IPRINT = 2
      IOUT   = 6
      IFAIL  = 0
C
C   starting values and bounds
C
      X(1)  = 0.25D0
      X(2)  = 0.39D0
      X(3)  = 0.415D0
      X(4)  = 0.39D0
      DO I = 1,N
         XL(I) = 0.0D0
         XU(I) = 1.0D5
      ENDDO
C
C   execute NLPL1 in reverse communication
C
    1 CONTINUE
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-1)) THEN
         DO J = 1,L
            CALL H(T(J), Y(J), N, X, FUNC(J))
         ENDDO
         CALL H(T(1), Y(1), N, X, FUNC(L+1))
         CALL H(T(L), Y(L), N, X, FUNC(L+2))
      ENDIF
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-2)) THEN
         DO J = 1,L
            CALL DH(T(J), N ,X, W)
            DO I=1,N
               GRAD(J,I) = W(I)
            ENDDO
         ENDDO
         CALL DH(T(1), N, X, W)
         DO I=1,N
            GRAD(L+1,I) = W(I)
         ENDDO
         CALL DH(T(L), N, X, W)
         DO I=1,N
            GRAD(L+2,I) = W(I)
         ENDDO
      ENDIF
C
C   call NLPL1
C
      CALL NLPL1(M, ME, LMMAX, L, N, LLN1, LLMNN2, X, FUNC, RES,
     /           GRAD, U, XL, XU, ACC, ACCQP, RESSIZ, MAXFUN, MAXIT,
     /           MAXNM, RHOB, IPRINT, IOUT, IFAIL, WA, LWA, KWA,
     /           LKWA, LOGWA, LLOGWA, QL)
      IF (IFAIL.LT.0) GOTO 1
C
C   end of main program
C
      STOP
      END
C
C   data fitting function
C
      SUBROUTINE        H(T, Y, N ,X, F)
      IMPLICIT          NONE
      INTEGER           N
      DOUBLE PRECISION  T, Y, X(N), F
C
      F = X(1)*T*(T + X(2))/(T**2 + X(3)*T + X(4)) - Y
C
      RETURN
      END
C
C   partial derivatives
C
      SUBROUTINE        DH(T, N ,X, DF)
      IMPLICIT          NONE
      INTEGER           N
      DOUBLE PRECISION  T, X(N), DF(N)
C
      DF(1) = T*(T + X(2))/(T**2 + X(3)*T + X(4))
      DF(2) = X(1)*T/(T**2 + X(3)*T + X(4))
      DF(3) = -X(1)*T**2*(T + X(2))/(T**2 + X(3)*T + X(4))**2
      DF(4) = -X(1)*T*(T + X(2))/(T**2 + X(3)*T + X(4))**2
```

```
C
      RETURN
      END

    --------------------------------------------------------------------
     START OF THE SEQUENTIAL QUADRATIC PROGRAMMING ALGORITHM
    --------------------------------------------------------------------

     Parameters:
        N      =      26
        M      =      24
        ME     =       2
        MODE   =       0
        ACC    =   0.1000D-09
        ACCQP  =   0.1000D-13
        STPMIN =   0.1000D-09
        MAXFUN =      20
        MAXNM  =      20
        MAXIT  =     100
        IPRINT =       2

     Output in the following order:
        IT    - iteration number
        F     - objective function value
        SCV   - sum of constraint violations
        NA    - number of active constraints
        I     - number of line search iterations
        ALPHA - steplength parameter
        DELTA - additional variable to prevent inconsistency
        KKT   - Karush-Kuhn-Tucker optimality criterion


      IT         F           SCV      NA  I    ALPHA     DELTA     KKT
    --------------------------------------------------------------------
       1  0.22000000D+02  0.55D-01   24  0  0.00D+00  0.00D+00  0.22D+02
       2  0.46298100D-01  0.39D-01   14  1  0.10D+01  0.00D+00  0.48D-01
       3  0.42617595D-01  0.31D-02   15  1  0.10D+01  0.00D+00  0.34D-02
       4  0.42581346D-01  0.32D-04   14  1  0.10D+01  0.00D+00  0.13D-03
       5  0.42521961D-01  0.11D-04   13  1  0.10D+01  0.00D+00  0.11D-03
       6  0.42436528D-01  0.32D-04   14  1  0.10D+01  0.00D+00  0.46D-03
       7  0.42030896D-01  0.70D-03   14  1  0.10D+01  0.00D+00  0.14D-02
       8  0.41860422D-01  0.20D-03   13  1  0.10D+01  0.00D+00  0.41D-03
       9  0.41796662D-01  0.42D-04   14  1  0.10D+01  0.00D+00  0.37D-03
      10  0.41499749D-01  0.75D-03   14  1  0.10D+01  0.00D+00  0.14D-02
      11  0.41264360D-01  0.57D-03   13  1  0.10D+01  0.00D+00  0.69D-03
      12  0.41224676D-01  0.47D-04   14  1  0.10D+01  0.00D+00  0.67D-04
```

```
13  0.41223394D-01  0.88D-07   15  1  0.10D+01  0.00D+00  0.11D-06
14  0.41223393D-01  0.11D-12   17  1  0.10D+01  0.00D+00  0.15D-12

--- Final Convergence Analysis at Best Iterate ---

  Best result at iteration:      ITER  =      14
  Objective function value:      F(X)  =  0.41223393D-01
  Solution values:               X     =
     0.18402828D+00  0.11994003D+01  0.75456942D+00  0.53893657D+00
     0.00000000D+00  0.00000000D+00  0.36239189D-03  0.56478830D-38
     0.00000000D+00  0.20896387D-02  0.95968093D-38  0.25918363D-01
     0.00000000D+00  0.82307563D-38  0.00000000D+00  0.16530389D-38
     0.44269623D-02  0.32372012D-38  0.40973874D-02  0.13513002D-02
     0.00000000D+00  0.10469247D-37  0.00000000D+00  0.29773494D-02
     0.89998786D-38  0.00000000D+00
  Multiplier values:             U     =
     0.15426936D+01 -0.23508590D+00  0.00000000D+00  0.00000000D+00
     0.10000000D+01  0.00000000D+00  0.00000000D+00  0.10000000D+01
     0.00000000D+00  0.10000000D+01  0.00000000D+00  0.69821839D+00
     0.00000000D+00  0.00000000D+00  0.10000000D+01  0.00000000D+00
     0.10000000D+01  0.10000000D+01  0.00000000D+00  0.76852701D+00
     0.00000000D+00  0.10000000D+01  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.10000000D+01  0.10000000D+01  0.00000000D+00  0.10000000D+01
     0.10000000D+01  0.00000000D+00  0.10000000D+01  0.00000000D+00
     0.10000000D+01  0.30178161D+00  0.10000000D+01  0.10000000D+01
     0.00000000D+00  0.10000000D+01  0.00000000D+00  0.00000000D+00
     0.10000000D+01  0.23147299D+00  0.10000000D+01  0.00000000D+00
     0.10000000D+01  0.10000000D+01  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
  Constraint values:             G(X)  =
    -0.96693487D-14 -0.63560268D-14 -0.96693487D-14  0.44269623D-02
    -0.11831627D-13  0.40973874D-02  0.13513002D-02 -0.17645607D-13
    -0.19900748D-13 -0.20254631D-13  0.29773494D-02 -0.12795320D-13
    -0.63560268D-14  0.96693487D-14  0.10697172D-13  0.36239189D-03
     0.13482271D-13  0.15370517D-13  0.20896387D-02  0.19900748D-13
     0.25918363D-01  0.18257097D-13  0.12795320D-13  0.63560268D-14
  Distance from lower bound:     XL-X  =
    -0.18402828D+00 -0.11994003D+01 -0.75456942D+00 -0.53893657D+00
     0.00000000D+00  0.00000000D+00 -0.36239189D-03 -0.56478830D-38
     0.00000000D+00 -0.20896387D-02 -0.95968093D-38 -0.25918363D-01
```

```
      0.00000000D+00 -0.82307563D-38  0.00000000D+00 -0.16530389D-38
     -0.44269623D-02 -0.32372012D-38 -0.40973874D-02 -0.13513002D-02
      0.00000000D+00 -0.10469247D-37  0.00000000D+00 -0.29773494D-02
     -0.89998786D-38  0.00000000D+00
   Distance from upper bound:     XU-X  =
      0.99999816D+05  0.99998801D+05  0.99999245D+05  0.99999461D+05
      0.10000000D+31  0.10000000D+31  0.10000000D+31  0.10000000D+31
      0.10000000D+31  0.10000000D+31  0.10000000D+31  0.10000000D+31
      0.10000000D+31  0.10000000D+31  0.10000000D+31  0.10000000D+31
      0.10000000D+31  0.10000000D+31  0.10000000D+31  0.10000000D+31
      0.10000000D+31  0.10000000D+31  0.10000000D+31  0.10000000D+31
      0.10000000D+31  0.10000000D+31
   Number of function calls:      NFUNC =      14
   Number of gradient calls:      NGRAD =      14
   Number of calls of QP solver:  NQL   =      14


 --- Final Convergence Analysis of NLPL1 ---

   Maximum function value:        RES   =  0.41223393D-01
   Function values:               F(X) =
     -0.96693487D-14  0.44269623D-02 -0.36239189D-03  0.40973874D-02
      0.13513002D-02 -0.20896387D-02 -0.19900748D-13 -0.25918363D-01
      0.29773494D-02 -0.12795320D-13 -0.63560268D-14
   Solution:                      X =
      0.18402828D+00  0.11994003D+01  0.75456942D+00  0.53893657D+00
   Multiplier values:             U =
      0.15426936D+01 -0.23508590D+00  0.00000000D+00  0.00000000D+00
      0.00000000D+00  0.00000000D+00  0.10000000D+01  0.10000000D+01
      0.00000000D+00  0.10000000D+01
   Constraint values:             G(X) =
     -0.96693487D-14 -0.63560268D-14
   Number of function calls:      NFUNC =  14
   Number of derivative calls:    NGRAD =  14
```

References

[1] Hock W., Schittkowski K. (1981): *Test Examples for Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer

[2] Schittkowski K. (1985/86): *NLPQL: A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500

[3] Schittkowski K. (2002): *Numerical Data Fitting in Dynamical Systems*, Kluwer Academic Publishers, Dordrecht

[4] Schittkowski K. (2002): *EASY-FIT: A software system for data fitting in dynamic systems*, Structural and Multidisciplinary Optimization, Vol. 23, No. 2, 153-169

[5] Schittkowski K. (2003): *QL: A Fortran code for convex quadratic programming - User's guide*, Report, Department of Mathematics, University of Bayreuth

[6] Schittkowski K. (2005): *DFNLP: A Fortran Implementation of an SQP-Gauss-Newton Algorithm - User's Guide, Version 2.0*, Report, Department of Computer Science, University of Bayreuth

[7] Schittkowski K. (2006): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide, version 2.2*, Report, Department of Computer Science, University of Bayreuth

[8] Schittkowski K. (2007): *NLPLSQ: A Fortran implementation of an SQP-Gauss-Newton algorithm for least-squares optimization - user's guide*, Report, Department of Computer Science, University of Bayreuth

[9] Schittkowski K. (2008): *NLPMMX: A Fortran implementation of an SQP-Gauss-Newton algorithm for min-max optimization - user's guide*, Report, Department of Computer Science, University of Bayreuth