NLPLSX: A Fortran Implementation of an SQP Algorithm for Least-Squares Optimization with Very Many Observations
- 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:	July, 2012

Abstract

The Fortran subroutine NLPLSX solves constrained least squares problems, where the sum of squared nonlinear functions is to be minimized. All functions are to be continuously differentiable. By assuming now that the sum is too long preventing the usage of special Gauss-Newton-type algorithms, the problem directly solved by the sequential quadratic programming (SQP) code NLPQLP. The usage of the code is documented, and an illustrative example is presented.

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

1 Introduction

Nonlinear least squares optimization is extremely important in many practical situations. Typical applications are maximum likelihood estimation, nonlinear regression, data fitting, system identification, or parameter estimation, respectively. In these cases, 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 data measured at the same time values. An extensive discussion of data fitting especially in case of dynamical systems is given by Schittkowski [4].

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

$$x \in I\!R^n : \begin{array}{l} \min \ \frac{1}{2} \sum_{i=1}^{l} f_i(x)^2 \\ g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ \ , \\ g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ \ , \\ x_l \leq x \leq x_u \ \ . \end{array} \tag{1}$$

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

In case of data fitting, we have a model function $h(x,t)$ depending in addition on a dynamic variable t, the time, and l pairs data of the form y_i and t_i, $i = 1$, ..., l, leading to

$$f_i(x) = h(x, t_i) - y_i \ \ . \tag{2}$$

In other words, we try to compute a set of model parameters x_1, ..., x_n, so that the distance of the model function at these parameters and the experimental time values from the measured data is as small as possible. The distance is measured in the Euclidean or L_2-norm, respectively.

A large variety of numerical methods is available to solve nonlinear least squares problems, most of them based on a Gauss-Newton approach. But we assume now that so many data points or functions $f_i(x)$ are available, i.e., that l is so large, that available algorithms are either not applicable or too slow.

The following sections contains a complete documentation of the Fortran code and an example implementation.

2 Calling Sequence

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

Usage:

```
    CALL  NLPLSX (      M,      ME,   LMMAX,       L,       N,
   /                LNMAX,    MNN2,       X,    FUNC,     RES,
   /                 GRAD,       U,      XL,      XU,     ACC,
   /                ACCQP,  RESSIZ,  MAXFUN   MAXIT,  MAXNM,
   /                 RHOB,  IPRINT,    IOUT,   IFAIL,      WA,
   /                  LWA,     KWA,    LKWA,  LOGWA,  LLOGWA,
   /                QPSLVE)
```

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.
LNMAX :	Dimensioning parameter, at least two and greater than N + L.
MNN2 :	Dimensioning parameter, must be set to M + 2*N + 2 when calling NLPLSX.
X(NMAX) :	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 NLPLSX 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 sum of squared residuals $f_1(x)^2 + \ldots + f_l(x)^2$.
GRAD(LMMAX, NMAX) :	The array is used to pass gradients of residuals and constraints to NLPLSX 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(LMNN2) : 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(NMAX), XU(NMAX) : 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.

RESSIZ : Dummy, any real variable for compatibility with NLPLSX.

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≥NMAX, or MNN2≠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*N/2 + N*M + 41*N + 10*M + 165 .
KWA(LKWA) :	KWA is an integer working array of length LKWA.
LKWA :	Length of the integer working array KWA. LKWA must be at least N + 30.
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 2*M + 10.
QPSLVE :	External subroutine to solve the quadratic programming subproblem. The calling sequence is

```
      CALL QPSLVE(M,ME,MMAX,N,NMAX,MNN,C,D,A,B,
     /            XL,XU,X,U,EPS,MODE,IOUT,IFAIL,IPRINT,
     /            WAR,LWAR,IWAR,LIWAR)
```

For more details about the choice and dimensions of arguments, see [6].

3 Program Organization

All declarations of real numbers must be done in double precision. Subroutine NLPLSX must be linked with the user-provided main program, the SQP code NLPQLP, and the quadratic programming code QL [6].

NLPLSX 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 NLPLSX, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in the first n positions of a double precision array called X.

2. Compute residual and constraint function values values, and store them in a 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 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 NLPLSX.

5. If NLPLSX 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 NLPLSX again, but do not change IFAIL.

6. If NLPLSX terminates with IFAIL=-2, compute gradient values subject to variables stored in X, and store them in GRAD as indicated above. Then call NLPLSX again without changing IFAIL.

7. If NLPLSX terminates with IFAIL=0, the internal stopping criteria are satisfied. The variable values found in X are considered as a local solution of the least squares 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.

4 Example

We consider a model function

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

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

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

The code and the corresponding screen output follow.

```
      IMPLICIT          NONE
      INTEGER           NMAX, MMAX, LMAX, LMMAX, MNN2MX, LWA, LKWA,
     /                  LLOGWA
      PARAMETER        (NMAX   = 5,
     /                  MMAX   = 2,
     /                  LMAX   = 200000)
      PARAMETER        (LMMAX  = MMAX + LMAX,
     /                  MNN2MX = MMAX + 2*NMAX + 2,
     /                  LWA    = 5*NMAX*NMAX/2 + NMAX*MMAX + 41*NMAX
     /                           + 10*MMAX + 165,
     /                  LKWA   = NMAX + 30,
     /                  LLOGWA = 2*MMAX + 10)
      INTEGER           M, ME, N, MNN2, L, 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(NMAX), FUNC(LMMAX), GRAD(LMMAX,NMAX),
     /                  U(MNN2MX), XL(NMAX), XU(NMAX),
     /                  WA(LWA), KWA(LKWA), LOGWA(LLOGWA),
     /                  T(LMAX), Y(LMAX), W(NMAX), PI
      LOGICAL           LOGWA
      EXTERNAL          QL
C
C   set parameters
C
      M      = MMAX
      ME     = MMAX
      N      = NMAX - 1
      MNN2   = M + N + N + 2
      L      = LMAX
      ACC    = 1.0D-14
      ACCQP  = 1.0D-14
      RESSIZ = 1.0D-4
      RHOB   = 0.0D0
      MAXFUN = 20
      MAXIT  = 100
      MAXNM  = 0
      IPRINT = 2
      IOUT   = 6
      IFAIL  = 0
      PI     = 3.1415D0
      DO  J=1,L
         T(J) = DBLE(J)/DBLE(L)*0.5D0*PI
         Y(J) = DSIN(T(J))
      ENDDO
C
```

```
C   starting values and bounds
C
      DO I = 1,N
         XL(I) = -1.0D5
         X(I)  = 0.0D0
         XU(I) = 1.0D5
      ENDDO
C
C   execute NLPLSX 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 NLPLSX
C
      CALL NLPLSX(M,ME,LMMAX,L,N,NMAX,MNN2,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      =    4
    M      =    2
    ME     =    2
    MODE   =    0
    ACC    =   0.1000D-13
    ACCQP  =   0.1000D-13
    STPMIN =   0.1000D-13
    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	0.49998775D+05	0.10D+01	2	0	0.00D+00	0.10D+01	0.15D+11
2	0.22676464D+05	0.10D+01	2	6	0.10D-04	0.00D+00	0.23D+06
3	0.22405881D+05	0.10D+01	2	4	0.39D-02	0.00D+00	0.38D+12
4	0.59116627D+04	0.14D+01	2	9	0.31D-05	0.00D+00	0.21D+05
5	0.95728313D+03	0.47D+00	2	1	0.10D+01	0.00D+00	0.29D+04
6	0.28031432D+03	0.19D+00	2	1	0.10D+01	0.00D+00	0.67D+03
7	0.16066648D+03	0.87D-01	2	1	0.10D+01	0.00D+00	0.31D+03
8	0.13378749D+03	0.72D-02	2	1	0.10D+01	0.00D+00	0.11D+02
9	0.13919220D+03	0.28D-04	2	1	0.10D+01	0.00D+00	0.35D+00
10	0.13895570D+03	0.90D-07	2	1	0.10D+01	0.00D+00	0.17D+00
11	0.13887425D+03	0.59D-09	2	1	0.10D+01	0.00D+00	0.23D-03
12	0.13887414D+03	0.41D-12	2	1	0.10D+01	0.00D+00	0.84D-07
13	0.13887414D+03	0.30D-15	2	1	0.10D+01	0.00D+00	0.13D-11
14	0.13887414D+03	0.82D-16	2	3	0.10D-01	0.00D+00	0.13D-12
15	0.13887414D+03	0.17D-20	2	1	0.10D+01	0.00D+00	0.23D-16

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

   Objective function value:      F(X)  =  0.13887414D+03
   Approximation of solution:     X     =
      0.18373272D+01 -0.13349605D-04  0.13152201D+01 -0.20427155D-04
   Approximation of multipliers:  U     =
     -0.10225645D+04 -0.26999975D+04  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)  =
      0.16940659D-20  0.00000000D+00
   Distance from lower bound:     XL-X  =
     -0.10000184D+06 -0.10000000D+06 -0.10000132D+06 -0.10000000D+06
   Distance from upper bound:     XU-X  =
      0.99998163D+05  0.10000000D+06  0.99998685D+05  0.10000000D+06
   Number of function calls:      NFUNC =   33
   Number of gradient calls:      NGRAD =   15
   Number of calls of QP solver:  NQL   =   19
```

```
--- Final Convergence Analysis of NLPLSX ---

  Number of observations:        L      = 200000
  Residual:                      RES(X) =  0.13887414D+03
  Approximation of solution:     X =
     0.18373272D+01 -0.13349605D-04  0.13152201D+01 -0.20427155D-04
  Approximation of multipliers:  U =
    -0.10225645D+04 -0.26999975D+04  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
     0.00000000D+00  0.00000000D+00
  Number of function calls:      NFUNC =  33
  Number of derivative calls:    NGRAD =  15
```

References

[1] Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction,* Optimization, Vol. 14, 197-216

[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. (1988): *Solving nonlinear least squares problems by a general purpose SQP-method,* in: Trends in Mathematical Optimization, K.-H. Hoffmann, J.-B. Hiriart-Urruty, C. Lemarechal, J. Zowe eds., International Series of Numerical Mathematics, Vol. 84, Birkhäuser, 295-309

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

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

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

[7] 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

[8] 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

[9] Schittkowski K. (2007): *NLPMMX: A Fortran implementation of a sequential quadratic programming algorithm for solving constrained nonlinear min-max problems - user's guide, version 1.0*, Report, Department of Computer Science, University of Bayreuth

[10] 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

[11] Schittkowski K. (2008): *NLPL1: A Fortran implementation of an SQP-Gauss-Newton algorithm for for minimizing sums of absolute function values - user's guide*, Report, Department of Computer Science, University of Bayreuth