NLPQLY: An Easy-To-Use Fortran Implementation of a Sequential Quadratic Programming Algorithm - 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:	November, 2012

Abstract

The Fortran subroutine NLPQLY simplifies the numerical solution of nonlinear programming problems by calling the standard SQP code NLPQLP, where the calling sequence is simplified as much as possible. A user has to provide objective and constraint function values in the same code which calls NLPQLY. Derivatives are internally approximated by forward differences. The usage of the code is documented and illustrated by an example.

Keywords: SQP, sequential quadratic programming, nonlinear programming, non-monotone line search, numerical algorithm, distributed computing, Fortran codes, easy-to-use

1 Introduction

We consider the general optimization problem to minimize an objective function f under nonlinear equality and inequality constraints,

$$x \in I\!R^n : \begin{array}{ll} \min f(x) & \\ g_j(x) = 0 \,, & j = 1, \ldots, m_e \\ g_j(x) \geq 0 \,, & j = m_e + 1, \ldots, m \\ x_l \leq x \leq x_u & \end{array} \tag{1}$$

where x is an n-dimensional parameter vector. It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $I\!R^n$.

Sequential quadratic programming (SQP) is the standard general purpose method to solve smooth nonlinear optimization problems, at least under the following assumptions:

- The problem is not too large.
- Functions and gradients can be evaluated with sufficiently high precision.
- The problem is smooth and well-scaled.

The code NLPQLP of Schittkowski [16, 27] is a Fortran implementation of a sequential quadratic programming (SQP) algorithm. The design of the numerical algorithm is founded on extensive comparative numerical tests of Schittkowski [9, 13, 11], Schittkowski et al. [28], Hock and Schittkowski [4], and on further theoretical investigations published in [10, 12, 14, 15].

To conduct the numerical tests, a random test problem generator is developed for a major comparative study, see [9]. Two collections with more than 300 academic and real-life test problems are published in Hock and Schittkowski [4] and in Schittkowski [17]. Fortran source codes and a test frame can be downloaded from the home page of the author,

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

see [27] for further details. New features of NLPQLP are the possibility to run the code under a distributed system and to perform uphill steps in certain error situations.

In Section 2, we briefly outline the general mathematical structure of an SQP algorithm. The usage of the Fortran subroutine is documented in Section 3 and Section 4 contains an illustrative example.

2 Sequential Quadratic Programming Methods

Sequential quadratic programming or SQP methods belong to the most powerful nonlinear programming algorithms we know today for solving differentiable nonlinear programming problems of the form (1). The theoretical background is described e.g. in Stoer [30] in form of a review, or in Spellucci [29] in form of an extensive text book. From the more practical point of view, SQP methods are also introduced in the books of Papalambros, Wilde [5] and Edgar, Himmelblau [1]. Their excellent numerical performance is tested and compared with other methods in Schittkowski [9], and since many years they belong to the most frequently used algorithms to solve practical optimization problems.

To facilitate the notation of this section, we assume that upper and lower bounds x_u and x_l are not handled separately, i.e., we consider the somewhat simpler formulation

$$\begin{array}{rl} & \min f(x) \\ x \in I\!R^n: & g_j(x) = 0\ , \quad j = 1, \ldots, m_e \\ & g_j(x) \geq 0\ , \quad j = m_e + 1, \ldots, m \end{array} \tag{2}$$

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on $I\!R^n$.

The basic idea is to formulate and solve a quadratic programming subproblem in each iteration which is obtained by linearizing the constraints and approximating the Lagrangian function

$$L(x,u) := f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{3}$$

quadratically, where $x \in I\!R^n$ is the primal variable and $u = (u_1, \ldots, u_m)^T \in I\!R^m$ the multiplier vector.

To formulate the quadratic programming subproblem, we proceed from given iterates $x_k \in I\!R^n$, an approximation of the solution, $v_k \in I\!R^m$, an approximation of the multipliers, and $B_k \in I\!R^{n \times n}$, an approximation of the Hessian of the Lagrangian function. Then one has to solve the quadratic programming problem

$$\begin{array}{rl} & \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d \\ d \in I\!R^n: & \nabla g_j(x_k)^T d + g_j(x_k) = 0\ , \quad j = 1, \ldots, m_e \\ & \nabla g_j(x_k)^T d + g_j(x_k) \geq 0\ , \quad j = m_e + 1, \ldots, m \end{array} \tag{4}$$

Let d_k be the optimal solution and u_k the corresponding multiplier of this subproblem. A new iterate is obtained by

$$\begin{pmatrix} x_{k+1} \\ v_{k+1} \end{pmatrix} := \begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha_k \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \tag{5}$$

where $\alpha_k \in (0, 1]$ is a suitable steplength parameter.

Although we are able to guarantee that the matrix B_k is positive definite, it is possible that (4) is not solvable due to inconsistent constraints. One possible remedy is to introduce an additional variable $\delta \in I\!R$, leading to a modified quadratic programming problem, see Schittkowski [16] for details.

The steplength parameter α_k is required in (5) to enforce global convergence of the SQP method, i.e., the approximation of a point satisfying the necessary Karush-Kuhn-Tucker optimality conditions when starting from arbitrary initial values, typically a user-provided $x_0 \in I\!R^n$ and $v_0 = 0$, $B_0 = I$. α_k should satisfy at least a sufficient decrease condition of a merit function $\phi_r(\alpha)$ given by

$$\phi_r(\alpha) := \psi_r \left(\begin{pmatrix} x \\ v \end{pmatrix} + \alpha \begin{pmatrix} d \\ u - v \end{pmatrix} \right) \tag{6}$$

with a suitable penalty function $\psi_r(x, v)$. Implemented is the augmented Lagrangian function

$$\psi_r(x, v) := f(x) - \sum_{j \in J} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) - \frac{1}{2} \sum_{j \in K} v_j^2 / r_j \quad , \tag{7}$$

with $J := \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, g_j(x) \leq v_j / r_j\}$ and $K := \{1, \ldots, m\} \setminus J$, cf. Schittkowski [14]. The objective function is *penalized* as soon as an iterate leaves the feasible domain. The corresponding penalty parameters r_j, $j = 1, \ldots, m$ that control the degree of constraint violation, must carefully be chosen to guarantee a descent direction of the merit function, see Schittkowski [14],

$$\phi'_{r_k}(0) = \bigtriangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < 0 \quad . \tag{8}$$

Finally one has to approximate the Hessian matrix of the Lagrangian function in a suitable way. To avoid calculation of second derivatives and to obtain a final superlinear convergence rate, the standard approach is to update B_k by the BFGS quasi-Newton formula, cf. Powell [7] or Stoer [30].

3 Program Documentation

NLPQLY is implemented in form of a Fortran subroutine. The quadratic programming problem is solved by the code QL, an implementation of the primal-dual method of Goldfarb and Idnani [2] going back to Powell [8], see also Schittkowski [25] for more details about implementation and usage. Model functions must be provided by reverse communication. The user has to evaluate function values in the same program which executes NLPQLY, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in the first column of an array X.

2. Compute objective and all constraint function values, store them in a scalar variable F and an array G, respectively.

3. Set IFAIL=0 and execute NLPQLY.

4. If NLPQLY returns with IFAIL¡0, compute objective and constraint function values at variable values found in X, store them in F and G, and call NLPQLY again.

5. If NLPQLY terminates with IFAIL=0, the internal optimality criteria are satisfied. In case of IFAIL>0, an error occurred.

Usage:

```
      CALL  NLPQLY(     M,     ME,  MMAX,     N,      F
     /                  G,     XL,    XU,   ACC,  MAXIT,
     /          IPRINT,  IOUT,  IFAIL,    WA,   LWA,
     /                   KWA,  LKWA,   ACT,  LACT       )
```

Definition of the parameters:

M :	Total number of constraints.
ME :	Number of equality constraints.
N :	Number of optimization variables.
X(N) :	Initially, the double precision array X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate.
F :	The double precision variable F has to contain the objective function value evaluated at X when calling NLPQLY.
G(M) :	The double precision array G has to contain the constraint function values at an iterate X when calling NLPQLY. If M=0, use G(1) to dimension G in the calling routine.
XL(N),XU(N) :	When calling NLPQLY the first time, the double precision arrays XL and XU must contain the lower and upper bounds of the variables, respectively.

ACC : The user has to specify the desired final accuracy (e.g. 1.0D-7). Since derivatives are computed by forward differences, it is not recommended to insert a value smaller than 1.0D-7.

MAXIT : The integer variable has to contain the maximum number of iterations when calling NLPQLY the first time. (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.

Note that constraint and multiplier values are not displayed for N,M>1,000.

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:

< 0 - 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 could not be terminated.

5 - Length of a working array too short.

6 - Dummy.

7 - Search direction close to zero at infeasible iterate.

8 - Starting point violates lower or upper bound.

9 - Wrong input parameter, e.g., IPRINT, IOUT.

10 - Inconsistency in QP, division by zero.

>100 - Error message of QP solver.

WA(LWA) :	WA is a double precision working array of length LWA.
LWA :	Length of WA, has to be at least at least 3*N*N + M*N + 45*N + 12*M + 200.
KWA(LKWA) :	KWA is an integer working array of length LKWA.
LKWA :	Length of KWA, has to be at least N + 27.
ACT(LACT) :	The logical array indicates constraints, which NLPQLY considers to be active at the last computed iterate, i.e., G(J) is active, if and only if ACT(J) is true for J=1,...,M.
LACT :	Length ACT, has to be at least 2*M+10.

There are 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 nonlinear and non-convex constraints are infeasible or not. Thus, the nonlinear programming algorithms will proceed until running in any of the mentioned error situations. In this case, the correctness of the model must be very carefully checked.

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

However, some of the error situations also occur if, because of inaccurate 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.

Important Note: The tolerance for approximating derivatives by a forward difference formula is set to the square root of the machine precision. This might be too small in case of inaccurate function values.

4 Example

To give an example how to organize the code, we consider Rosenbrock's post office problem, i.e., test problem TP37 of Hock and Schittkowski [4],

$$x_1, x_2 \in I\!R: \begin{array}{l} \min\ -x_1 x_2 x_3 \\ x_1 + 2x_2 + 2x_3 \geq 0 \\ 72 - x_1 - 2x_2 - 2x_3 \geq 0 \\ 0 \leq x_1 \leq 42 \\ 0 \leq x_2 \leq 42 \\ 0 \leq x_3 \leq 42 \end{array} \tag{9}$$

The Fortran source code for executing NLPQLP is listed below. Gradients are approximated by forward differences. The function block inserted in the main program can be replaced by a subroutine call.

```
      IMPLICIT          NONE
      INTEGER           N, M, LWA, LKWA, LACTIV
      PARAMETER (       N      = 3,
     /                  M      = 2,
     /                  LWA    = 3*N*N + M*N + 45*N + 12*M + 200,
     /                  LKWA   = N + 27,
     /                  LACTIV = 2*M + 10 )
      INTEGER           KWA(LKWA), ME, MAXIT, IPRINT, IOUT, IFAIL, I,
     /                  NFUNC
      DOUBLE PRECISION  X(N), F, G(M), XL(N), XU(N), WA(LWA), ACC
      LOGICAL           ACTIVE(LACTIV)
C
C   Set some constants and starting values
C
      IOUT   = 6
      ACC    = 1.0D-8
      MAXIT  = 100
      IPRINT = 2
      ME     = 0
      IFAIL  = 0
      NFUNC  = 0
      DO I=1,N
         X(I)  = 10.0D0
         XL(I) = 0.0D0
         XU(I) = 42.0D0
      ENDDO
    1 CONTINUE
C============================================================
```

```
C   This is the main block to compute all function values.
C   The block is executed either for computing a steplength
C   sequentially or for approximating gradients by forward
C   differences.
C
      F    = -X(1)*X(2)*X(3)
      G(1) =  X(1) + 2.0D0*X(2) + 2.0D0*X(3)
      G(2) =  72.0D0 - X(1) - 2.0D0*X(2) - 2.0D0*X(3)
C
C============================================================
      NFUNC = NFUNC + 1
      CALL NLPQLY (     M,     ME,      N,      X,      F,
     /                  G,     XL,     XU,    ACC,  MAXIT,
     /             IPRINT,   IOUT,  IFAIL,     WA,    LWA,
     /                KWA,   LKWA, ACTIVE, LACTIV         )
      IF (IFAIL.LT.0) GOTO 1
C
      WRITE(IOUT,1000) NFUNC
 1000 FORMAT('   *** Number of function calls: ',I3)
C
      STOP
      END
```

5 Conclusions

We present an easy-to-use version of the SQP code NLPQLP, see Schittkowski [27], where a limited set of parameters is passed and where only objective and constraint function are to be provided by the user. Derivatives are evaluated internally by forward differences.

References

[1] Edgar T.F., Himmelblau D.M. (1988): *Optimization of Chemical Processes,* McGraw Hill

[2] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1-33

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

[4] Hock W., Schittkowski K. (1983): *A comparative performance evaluation of 27 nonlinear programming codes*, Computing, Vol. 30, 335-358

[5] Papalambros P.Y., Wilde D.J. (1988): *Principles of Optimal Design,* Cambridge University Press

[6] Powell M.J.D. (1978): *A fast algorithm for nonlinearly constraint optimization calculations,* in: Numerical Analysis, G.A. Watson ed., Lecture Notes in Mathematics, Vol. 630, Springer

[7] Powell M.J.D. (1978): *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in: Nonlinear Programming 3, O.L. Mangasarian, R.R. Meyer, S.M. Robinson eds., Academic Press

[8] Powell M.J.D. (1983): *On the quadratic programming algorithm of Goldfarb and Idnani*. Report DAMTP 1983/Na 19, University of Cambridge, Cambridge

[9] Schittkowski K. (1980): *Nonlinear Programming Codes,* Lecture Notes in Economics and Mathematical Systems, Vol. 183 Springer

[10] Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 1: Convergence analysis*, Numerische Mathematik, Vol. 38, 83-114

[11] Schittkowski K. (1981): *The nonlinear programming method of Wilson, Han and Powell. Part 2: An efficient implementation with linear least squares subproblems*, Numerische Mathematik, Vol. 38, 115-127

[12] Schittkowski K. (1982): *Nonlinear programming methods with linear least squares subproblems*, in: Evaluating Mathematical Programming Techniques, J.M. Mulvey ed., Lecture Notes in Economics and Mathematical Systems, Vol. 199, Springer

[13] Schittkowski K. (1983): *Theory, implementation and test of a nonlinear programming algorithm*, in: Optimization Methods in Structural Design, H. Eschenauer, N. Olhoff eds., Wissenschaftsverlag

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

[15] Schittkowski K. (1985): *On the global convergence of nonlinear programming algorithms*, ASME Journal of Mechanics, Transmissions, and Automation in Design, Vol. 107, 454-458

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

[17] Schittkowski K. (1987a): *More Test Examples for Nonlinear Programming,* Lecture Notes in Economics and Mathematical Systems, Vol. 182, Springer

[18] Schittkowski K. (1987): *New routines in MATH/LIBRARY for nonlinear programming problems*, IMSL Directions, Vol. 4, No. 3

[19] 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

[20] Schittkowski K. (1992): *Solving nonlinear programming problems with very many constraints*, Optimization, Vol. 25, 179-196

[21] Schittkowski K. (1994): *Parameter estimation in systems of nonlinear equations*, Numerische Mathematik, Vol. 68, 129-142

[22] Schittkowski K. (2002): *Test problems for nonlinear programming - user's guide*, Report, Department of Mathematics, University of Bayreuth

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

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

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

[26] Schittkowski K. (2003): *DFNLP: A Fortran implementation of an SQP-Gauss-Newton algorithm - user's guide*, Report, Department of Mathematics, University of Bayreuth, 2003

[27] Schittkowski K. (2009): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide*, Report, Department of Computer Science, University of Bayreuth

[28] Schittkowski K., Zillober C., Zotemantel R. (1994): *Numerical comparison of nonlinear programming algorithms for structural optimization*, Structural Optimization, Vol. 7, No. 1, 1-28

[29] Spellucci P. (1993): *Numerische Verfahren der nichtlinearen Optimierung,* Birkhäuser

[30] Stoer J. (1985): *Foundations of recursive quadratic programming methods for solving nonlinear programs,* in: Computational Mathematical Programming, K. Schittkowski, ed., NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 15, Springer