NLPLSQ: A Fortran Implementation of an SQP-Gauss-Newton Algorithm for Least-Squares Optimization

- 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: March, 2016

Abstract

The Fortran subroutine NLPLSQ solves constrained least squares nonlinear programming problems, where the sum of squared nonlinear functions is to be minimized. It is assumed that all functions are continuously differentiable. By introducing additional variables and nonlinear equality constraints, the problem is transformed into a general smooth nonlinear program subsequently solved by the sequential quadratic programming (SQP) code NLPQLP. It can be shown that typical features of special purpose algorithms are retained, i.e., a combination of a Gauss-Newton and a quasi-Newton search direction. The additionally introduced variables are eliminated in the quadratic programming subproblem, so that calculation time is not increased significantly. Some comparative numerical results are included, the usage of the code is documented, and two illustrative examples are presented.

Keywords: least squares optimization, data fitting, Gauss-Newton method, 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 measured data at the same time values. An extensive discussion of data fitting especially in case of dynamical systems is given by Schittkowski [30].

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.

Although many nonlinear least squares programs were developed in the past, see Hiebert [13] for an overview and numerical comparison, only very few programs were written for the nonlinearly constrained case, see, e.g., Holt and Fletcher [15], Lindström [18], Mahdavi-Amiri and Bartels [20], or Fernanda et al. [7]. However, the implementation of one of these or any similar special purpose code requires a large amount of theoretical and numerical analysis.

In this paper, we consider the question how an existing nonlinear programming code can be used to solve constrained nonlinear least squares problems in an efficient and robust way. We will see that a simple transformation of the model under consideration and subsequent solution by a sequential quadratic programming (SQP) algorithm retains typical features of special purpose methods, i.e., the combination of a Gauß-Newton search direction with a quasi-Newton correction. Numerical test results indicate that the proposed approach is as efficient as special purpose methods, although the required programming effort is negligible provided that an SQP code is available.

In a very similar way, also L_1, L_∞, and min-max problems can be solved efficiently by an SQP code after a suitable transformation, see Schittkowski [30, 33, 35].

The following section describes some basic features of least squares optimization, especially some properties of Gauss-Newton and related methods. The transformation of a least squares problem into a special nonlinear program is described in Section 3. We will discuss how some basic features of special purpose algorithms are retained. The same ideas are extended to the constrained case in Section 4. In Section 5, we summarize some comparative numerical results on a large set of test

problems. Sections 6 to 8 contain a complete documentation of the Fortran code and two example implementations.

2 Least Squares Methods

First, we consider unconstrained least squares problems where we omit constraints and bounds to simplify the notation,

$$\begin{aligned} &\min \ \tfrac{1}{2}\sum_{i=1}^{l} \ f_i(x)^2 \\ &x \in I\!R^n \ . \end{aligned} \tag{2}$$

These problems possess a long history in mathematical programming and are extremely important in practice, particularly in nonlinear data fitting or maximum likelihood estimation, see, e.g., Björck [2] or Dennis [3]. Thus, a large number of mathematical algorithms is available for solving (2).

To understand their basic features, we introduce the notation

$$F(x) = (f_1(x), \ldots, f_l(x))^T$$

and let

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

Then

$$\nabla f(x) = \nabla F(x) F(x) \tag{3}$$

defines the Jacobian of the objective function with $\nabla F(x) = (\nabla f_1(x), \ldots, \nabla f_l(x))$. If we assume now that all functions f_1, ..., f_l are twice continuously differentiable, we get the Hessian matrix of f by

$$\nabla^2 f(x) = \nabla F(x) \nabla F(x)^T + B(x) \ , \tag{4}$$

where

$$B(x) = \sum_{i=1}^{l} f_i(x) \nabla^2 f_i(x) \ . \tag{5}$$

Proceeding from a given iterate x_k, Newton's method can be applied to (2) to get a search direction $d_k \in I\!R^n$ by solving the linear system

$$\nabla^2 f(x_k) d + \nabla f(x_k) = 0$$

or, alternatively,

$$\nabla F(x_k) \nabla F(x_k)^T d + B(x_k) d + \nabla F(x_k) F(x_k) = 0 \ . \tag{6}$$

Assume for a moment that

$$F(x^\star) = (f_1(x^\star), \ldots, f_l(x^\star))^T = 0$$

at an optimal solution $x^\star$. A possible situation is a perfect fit where model function values coincide with experimental data. Because of $B(x^\star) = 0$, we neglect matrix $B(x_k)$ in (6), see also (5). Then (6) defines the so-called normal equations of the linear least squares problem

$$\begin{aligned} &\min \ \|\nabla F(x_k)^T d + F(x_k)\| \\ &d \in I\!R^n \quad . \end{aligned} \tag{7}$$

A new iterate is obtained by $x_{k+1} = x_k + \alpha_k d_k$, where d_k is a solution of (7) and where α_k denotes a suitable steplength parameter. It is obvious that a quadratic convergence rate is achieved when starting sufficiently close to an optimal solution. The above calculation of a search direction is known as the Gauss-Newton method and represents the traditional way to solve nonlinear least squares problems, see Björck [2] for more details. In general, the Gauss-Newton method possesses the attractive feature that it converges quadratically although we only provide first order information.

However, the assumptions of the convergence theorem of Gauss-Newton methods are very strong and cannot be satisfied in real situations. We have to expect difficulties in case of non-zero residuals, rank-deficient Jacobian matrices, non-continuous derivatives, and starting points far away from a solution. Further difficulties arise when trying to solve large residual problems, where $F(x^\star)^T F(x^\star)$ is not sufficiently small, for example relative to $\|\nabla F(x^\star)\|$. Numerous proposals have been made in the past to deal with this situation, and it is outside the scope of this section to give a review of all possible attempts developed in the last 30 years. Only a few remarks are presented to illustrate basic features of the main approaches, for further reviews see Gill, Murray and Wright [10], Ramsin and Wedin [25], or Dennis [4].

A very popular method is known under the name Levenberg-Marquardt algorithm, see Levenberg [16] and Marquardt [21]. The key idea is to replace the Hessian in (6) by a multiple of the identity matrix, say $\lambda_k I$, with a suitable positive factor λ_k. We get a uniquely solvable system of linear equations of the form

$$\nabla F(x_k)\nabla F(x_k)^T d + \lambda_k d + \nabla F(x_k) F(x_k) = 0 \quad .$$

For the choice of λ_k and the relationship to so-called *trust region methods*, see Moré [22].

A more sophisticated idea is to replace $B(x_k)$ in (6) by a quasi-Newton-matrix B_k, see Dennis [3]. But some additional safeguards are necessary to deal with indefinite matrices $\nabla F(x_k)\nabla F(x_k)^T + B_k$ in order to get a descent direction. A modified algorithm is proposed by Gill and Murray [9], where B_k is either a second-order approximation of $B(x_k)$, or a quasi-Newton matrix. In this case, a diagonal matrix is added to $\nabla F(x_k)\nabla F(x_k)^T + B_k$ to obtain a positive definite matrix. Lindström [17] proposes a combination of a Gauss-Newton and a Newton method by using a certain subspace minimization technique.

If, however, the residuals are too large, there is no possibility to exploit the special structure and a general unconstrained minimization algorithm, for example a quasi-Newton method, can be applied as well.

3 The SQP-Gauss-Newton Method

Many efficient special purpose computer programs are available to solve unconstrained nonlinear least squares problems. On the other hand, there exists a very simple approach to combine the valuable properties of Gauss-Newton methods with that of SQP algorithms in a straightforward way with almost no additional efforts. We proceed from an unconstrained least squares problem in the form

$$\begin{array}{l} \min \frac{1}{2} \sum_{i=1}^{l} \; f_i(x)^2 \\ x \in I\!R^n \quad , \end{array} \tag{8}$$

see also (2). Since most nonlinear least squares problems are ill-conditioned, it is not recommended to solve (8) directly by a general nonlinear programming method. But we will see in this section that a simple transformation of the original problem and its subsequent solution by an SQP method retains typical features of a special purpose code and prevents the need to take care of negative eigenvalues of an approximated Hessian matrix as in the case of alternative approaches. The corresponding computer program can be implemented in a few lines provided that a SQP algorithm is available.

The transformation, also described in Schittkowski [29, 30, 33], consists of introducing l additional variables $z = (z_1, \ldots, z_l)^T$ and l additional nonlinear equality constraints of the form

$$f_i(x) - z_i = 0 \; , \quad i = 1, \ldots, l \quad . \tag{9}$$

Then the equivalent transformed problem is

$$(x, z) \in I\!R^{n+l} : \quad \begin{array}{l} \min \; \frac{1}{2} z^T z \\ F(x) - z = 0 \quad , \end{array} \tag{10}$$

$F(x) = (f_1(x), \ldots, f_l(x))^T$. We consider now (10) as a general nonlinear programming problem of the form

$$\bar{x} \in I\!R^{\bar{n}} : \quad \begin{array}{l} \min \bar{f}(\bar{x}) \\ \bar{g}(\bar{x}) = 0 \end{array} \tag{11}$$

with $\bar{n} = n + l$, $\bar{x} = (x, z)$, $\bar{f}(x, z) = \frac{1}{2} z^T z$, $\bar{g}(x, z) = F(x) - z$, and apply an SQP algorithm, see Spellucci [36], Stoer [37], or Schittkowski [27, 30]. The quadratic programming subproblem is of the form

$$\bar{d} \in I\!R^{\bar{n}} : \quad \begin{array}{l} \min \frac{1}{2} \bar{d}^T \bar{B}_k \bar{d} + \nabla \bar{f}(\bar{x}_k)^T \bar{d} \\ \nabla \bar{g}(\bar{x}_k)^T \bar{d} + \bar{g}(\bar{x}_k) = 0 \quad . \end{array} \tag{12}$$

Here, $\bar{x}_k = (x_k, z_k)$ is a given iterate and

$$\bar{B}_k = \left(\begin{array}{ccc} B_k & : & C_k \\ C_k^T & : & D_k \end{array} \right) \tag{13}$$

with $B_k \in I\!R^{n \times n}$, $C_k \in I\!R^{n \times l}$, and $D_k \in I\!R^{l \times l}$, a given approximation of the Hessian of the Lagrangian function $L(\bar{x}, u)$ defined by

$$\begin{aligned} L(\bar{x}, u) &= \bar{f}(\bar{x}) - u^T \bar{g}(\bar{x}) \\ &= \tfrac{1}{2} z^T z - u^T (F(x) - z) \ . \end{aligned}$$

Since

$$\nabla_{\bar{x}} L(\bar{x}, u) = \left(\begin{array}{c} -\nabla F(x) u \\ z + u \end{array} \right)$$

and

$$\nabla^2_{\bar{x}} L(\bar{x}, u) = \left(\begin{array}{ccc} B(x, u) & : & 0 \\ 0 & : & I \end{array} \right)$$

with

$$B(x, u) = -\sum_{i=1}^{l} u_i \nabla^2 f_i(x) \ , \tag{14}$$

it seems to be reasonable to proceed now from a quasi-Newton matrix given by

$$\bar{B}_k = \left(\begin{array}{ccc} B_k & : & 0 \\ 0 & : & I \end{array} \right) \ , \tag{15}$$

where $B_k \in I\!R^{n \times n}$ is a suitable positive definite approximation of $B(x_k, u_k)$. Insertion of this $\bar{B}_k$ into (12) leads to the equivalent quadratic programming subproblem

$$(d, e) \in I\!R^{n+l} : \begin{array}{l} \min \ \tfrac{1}{2} d^T B_k d + \tfrac{1}{2} e^T e + z_k^T e \\ \nabla F(x_k)^T d - e + F(x_k) - z_k = 0 \ , \end{array} \tag{16}$$

where we replaced $\bar{d}$ by (d, e). Some simple calculations show that the solution of the above quadratic programming problem is identified by the linear system

$$\nabla F(x_k) \nabla F(x_k)^T d + B_k d + \nabla F(x_k) F(x_k) = 0 \ . \tag{17}$$

This equation is identical to (6), if $B_k = B(x_k)$, and we obtain a Newton direction for solving the unconstrained least squares problem (8).

Note that $B(x)$ defined by (5) and $B(x)$ defined by (14) coincide at an optimal solution of the least squares problem, since $F(x_k) = -u_k$. Based on the above considerations, an SQP method can be applied to solve (10) directly. The quasi-Newton-matrices $\bar{B}_k$ are always positive definite, and consequently also the matrix B_k defined by (13). Therefore, we omit numerical difficulties imposed by negative eigenvalues as found in the usual approaches for solving least squares problems.

When starting the SQP method, one could proceed from a user-provided initial guess x_0 for the variables and define

$$\begin{aligned} z_0 &= F(x_0) \ , \\ B_0 &= \begin{pmatrix} \mu I & : & 0 \\ 0 & : & I \end{pmatrix} \ , \end{aligned} \tag{18}$$

guaranteeing a feasible starting point $\bar{x}_0$. The choice of B_0 is of the form (15) and allows a user to provide some information on the estimated size of the residuals, if available. If it is known that the final norm $F(x^\star)^T F(x^\star)$ is close to zero at the optimal solution $x^\star$, the user could choose a small μ in (18). At least in the first iterates, the search directions are more similar to a Gauss-Newton direction. Otherwise, a user could define $\mu = 1$, if a large residual is expected.

Under the assumption that $\bar{B}_k$ is decomposed in the form (15) and that $\bar{B}_k$ be updated by the BFGS formula, then $\bar{B}_{k+1}$ is decomposed in the form (15), see Schittkowski [29]. The decomposition (15) is rarely satisfied in practice, but seems to be reasonable, since the intention is to find a $x^\star \in I\!R^n$ with

$$\nabla F(x^\star) F(x^\star) = 0 \ ,$$

and $\nabla F(x_k)^T d_k + F(x_k)$ is a Taylor approximation of $F(x_{k+1})$. Note also that the usual way to derive Newton's method is to assume that the optimality condition is satisfied for a certain linearization of a given iterate x_k, and to use this linearized system for obtaining a new iterate.

Example 3.1 *We consider the banana function*

$$f(x_1, x_2) = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2 \ .$$

When applying the nonlinear programming code NLPQLP of Schittkowski [27, 34], an implementation of a general purpose SQP method, we get the iterates of Table 1 when starting at $x_0 = (-1.2, 1.0)^T$. The objective function is scaled by 0.5 to adjust this factor in the least squares formulation (1). The last column contains an internal stopping condition based on the optimality criterion, in our unconstrained case equal to

$$|\nabla f(x_k) B_k^{-1} \nabla f(x_k)|$$

with a quasi-Newton matrix B_k. We observe a very fast final convergence speed, but a relatively large number of iterations.
The transformation discussed above, leads to the equivalent constrained nonlinear programming problem

$$\begin{aligned} & \min \ {z_1}^2 + {z_2}^2 \\ x_1, x_2, z_1, z_2 : \ & 10(x_2 - {x_1}^2) - z_1 = 0 \ , \\ & 1 - x_1 - z_2 = 0 \ . \end{aligned}$$

NLPQLP computes the results of Table 2, where the second column shows in addition the maximal constraint violation.

k	$f(x_k)$	$s(x_k)$
0	24.20	$0.54 \cdot 10^{5}$
1	12.21	$0.63 \cdot 10^{2}$
2	7.98	$0.74 \cdot 10^{2}$
	...	
35	$0.29 \cdot 10^{-3}$	$0.47 \cdot 10^{-3}$
36	$0.19 \cdot 10^{-4}$	$0.39 \cdot 10^{-4}$
37	$0.12 \cdot 10^{-5}$	$0.25 \cdot 10^{-5}$
38	$0.12 \cdot 10^{-7}$	$0.24 \cdot 10^{-7}$
39	$0.58 \cdot 10^{-12}$	$0.11 \cdot 10^{-11}$
40	$0.21 \cdot 10^{-15}$	$0.42 \cdot 10^{-15}$

Table 1: NLP Formulation of Banana Function

k	$f(x_k)$	$r(x_k)$	$s(x_k)$
0	12.10	0.0	$0.24 \cdot 10^{2}$
1	$0.96 \cdot 10^{-10}$	$0.48 \cdot 10^{2}$	$0.23 \cdot 10^{-4}$
2	$0.81 \cdot 10^{-10}$	$0.40 \cdot 10^{-10}$	$0.16 \cdot 10^{-9}$
3	$0.88 \cdot 10^{-21}$	$0.14 \cdot 10^{-8}$	$0.18 \cdot 10^{-20}$

Table 2: Least Squares Formulation of Banana Function

4 Constrained Least Squares Problems

Now we consider constrained nonlinear least squares problems

$$x \in I\!R^n : \begin{array}{l} \min \ \frac{1}{2} \sum_{i=1}^{l} f_i(x)^2 \\ g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ , \\ x_l \leq x \leq x_u \ . \end{array} \tag{19}$$

A combination of the SQP method with the Gauss-Newton method is proposed by Mahdavi-Amiri [19]. Lindström [18] developed a similar method based on an active set idea leading to a sequence of equality constrained linear least squares problems. A least squares code for linearly constrained problems was published by Hanson and Krogh [12] that is based on a tensor model.

On the other hand, a couple of SQP codes are available for solving general smooth nonlinear programming problems, for example VFO2AD (Powell [23]), NLPQLP (Schittkowski [27, 34]), NPSOL (Gill, Murray, Saunders, Wright [11]), or DONLP2 (Spellucci [36]). Since most nonlinear least squares problems are ill-conditioned, it is not recommended to solve (19) directly by a general nonlinear programming method as shown in the previous section. The same transformation used before can

be extended to solve also constrained problems. The subsequent solution by an SQP method retains typical features of a special purpose code and is easily implemented.

As outlined in the previous section, we introduce l additional variables $z = (z_1, \ldots, z_l)^T$ and l additional nonlinear equality constraints of the form

$$f_i(x) - z_i = 0 \ ,$$

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

$$\begin{aligned} & \min \ \tfrac{1}{2} z^T z \\ & f_i(x) - z_i = 0 \ , \quad i = 1, \ldots, l \ , \\ (x,z) \in I\!R^{n+l} : \ & g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ & g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ , \\ & x_l \leq x \leq x_n \ . \end{aligned} \tag{20}$$

In this case, the quadratic programming subproblem has the form

$$\begin{aligned} & \min \ \tfrac{1}{2} (d^T, e^T) \bar{B}_k \begin{pmatrix} d \\ e \end{pmatrix} + z_k^T e \\ & \nabla f_i(x_k)^T d - e + f_i(x_k) - z_i^k = 0 \ , \quad i = 1, \ldots, l \ , \\ (d,e) \in I\!R^{n+l} : \ & \nabla g_j(x_k)^T d + g_j(x_k) = 0 \ , \quad j = 1, \ldots, m_e \ , \\ & \nabla g_j(x_k)^T d + g_j(x_k) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ , \\ & x_l - x_k \leq d \leq x_u - x_k \ . \end{aligned} \tag{21}$$

It is possible to simplify the problem by substituting

$$e = \nabla F(x_k)^T d + F(x_k) - z_k \ ,$$

so that the quadratic programming subproblem depends on only n variables and m linear constraints. This is an important observation from the numerical point of view, since the computational effort to solve (21) reduces from the order of $(n+l)^3$ to n^3, and the remaining computations in the outer SQP frame are on the order of $(n+l)^2$. Therefore, the computational work involved in the proposed least squares algorithm is comparable to the numerical efforts required by special purpose methods, at least if the number l of observations is not too large.

When implementing the above proposal, one has to be aware that the quadratic programming subproblem is sometimes expanded by an additional variable δ, so that some safeguards are required. Except for this limitation, the proposed transformation (20) is independent from the variant of the SQP method used, so that available codes can be used in the form of a *black box*.

In principle, one could use the starting points proposed by (18). Numerical experience suggests, however, starting from $z_0 = F(x_0)$ only if the constraints are satisfied at x_0,

$$\begin{aligned} g_j(x_0) &= 0\ , \quad j = 1, \ldots, m_e\ , \\ g_j(x_0) &\geq 0\ , \quad j = m_e + 1, \ldots, m\ . \end{aligned}$$

In all other cases, it is proposed to proceed from $z_0 = 0$.

A final remark concerns the theoretical convergence of the algorithm. Since the original problem is transformed into a general nonlinear programming problem, we can apply all convergence results known for SQP methods. If an augmented Lagrangian function is preferred for the merit function, a global convergence theorem is found in Schittkowski [26]. The theorem states that when starting from an arbitrary initial value, a Karush-Kuhn-Tucker point is approximated, i.e., a point satisfying the necessary optimality conditions. If, on the other hand, an iterate is sufficiently close to an optimal solution and if the steplength is 1, then the convergence speed of the algorithm is superlinear, see Powell [24] for example. This remark explains the fast final convergence rate one observes in practice.

The assumptions are standard and are required by any special purpose algorithm in one or another form. But in our case, we do not need any regularity conditions for the function f_1, ..., f_l, i.e., an assumption that the matrix $\nabla F(x_k)$ is of full rank, to adapt the mentioned convergence results to the least squares case. The reason is found in the special form of the quadratic programming subproblem (21), since the first l constraints are linearly independent and are also independent of the remaining restrictions.

5 Performance Evaluation

Since the proposed transformation does not depend on constraints, we consider now the unconstrained least squares problem

$$x \in I\!R^n : \quad \begin{array}{l} \min \frac{1}{2} \sum_{i=1}^{l} f_i(x)^2 \\ x_l \leq x \leq x_u \quad . \end{array} \tag{22}$$

We assume that all functions $f_i(x)$, $i = 1, \ldots, l$, are continuously differentiable. Efficient and reliable least squares algorithms were implemented mainly in the 1960s and 1970s, see for example Fraley [8] for a review. An early comparative study of 13 codes was published by Bard [1]. In most cases, mathematical algorithms are based on the Gauss-Newton method, see Section 2. When developing and testing new implementations, the authors used standard test problems, which have been collected from literature and which do not possess a data fitting structure in most cases, see Dennis et al. [5], Hock and Schittkowski [14], or Schittkowski [28].

Our intention is to present a comparative study of some least squares codes, when applied to a set of data fitting problems. Test examples are given in form of analytical functions to avoid additional side effects introduced by round-off or integration errors as, e.g., in the case of dynamical systems. We proceed from a subset of the parameter estimation problems listed in Schittkowski [30], a set of 143 least squares functions. More details, in particular the corresponding model functions, data, and results, are found in the database of the software system EASY-FIT [30, 31], which can be downloaded from the home page of the author[1]. Derivatives are evaluated by the automatic differentiation tool PCOMP, see Dobmann et al. [6]. The following least squares routines are executed to solve the test problems mentioned before:

NLPLSQ: By transforming the original problem into a general nonlinear programming problem in a special way, typical features of a Gauss-Newton and quasi-Newton least squares method are retained, as outlined in the previous sections. The resulting optimization problem is solved by a standard sequential quadratic programming code called NLPQLP, see Schittkowski [27, 34].

NLSNIP: The code is a special purpose implementation for solving constrained nonlinear least squares problems by a combination of Gauss-Newton, Newton, and quasi-Newton techniques, see Lindström [17, 18].

DN2GB: The subroutine is a frequently used unconstrained least squares algorithm developed by Dennis et al. [5]. The mathematical method is also based on a combined Gauss-Newton and quasi-Newton approach.

All algorithms are capable of taking upper and lower bounds of the variables into account, but only NLPLSQ and NLSNIP are able to solve also constrained problems.

[1] http://www.klaus-schittkowski.de/

code	*succ*	n_f	n_g
NLPLSQ	94.4 %	30.2	19.6
NLSNIP	87.4 %	26.5	17.0
DN2GB	93.0 %	27.1	19.2

Table 3: Performance Results for Explicit Test Problems

The optimization routines are executed with the same set of input parameters, although we know that in one or another case, these tolerances can be adapted to a special situation leading to better individual results. Termination tolerance for NLPLSQ is 10^{-10}. DN2GB is executed with tolerances 10^{-9} and 10^{-7} for the relative function and variable convergence. NLSNIP uses a tolerance of 10^{-10} for the relative termination criteria and 10^{-8} for the absolute stopping condition. The total number of iterations is bounded by 1,000 for all three algorithms.

In some situations, an algorithm is unable to stop at the same optimal solution obtained by the other ones. There are many possible reasons, for example termination at a local solution, internal instabilities, or round-off errors. Thus, we need a decision when an optimization run is considered to be a successful one or not. We claim that successful termination is obtained if the total residual norm differs at most by 1 % from the best value obtained by all three algorithms, or, in case of a problem with zero residuals, is less than 10^{-7}. The percentage of successful runs is listed in Table 3, where the corresponding column is denoted by *succ*.

Comparative performance data are evaluated only for those test problems which are successfully solved by all three algorithms, altogether 95 problems. The corresponding mean values for number of function and gradient evaluations are denoted by n_f and n_g and are also shown in Table 3.

Although the number of test examples is too low to obtain statistically relevant results, we get the impression that the codes DN2GB and NLSNIP behave best with respect to efficiency. NLPLSQ and DN2GB are somewhat more reliable than the others subject to convergence towards a global solution. However, none of the codes tested is able to solve all problems within the required accuracy.

6 Calling Sequence

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

Usage:

```
    CALL  NLPLSQ (      M,      ME,   LMMAX,       L,       N,
   /                LNMAX,   LMNN2,       X,    FUNC,     RES,
   /                 GRAD,       U,      XL,      XU,     ACC,
   /                ACCQP,  RESSIZ,  MAXFUN   MAXIT,  MAXNM,
   /                  RHO,  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 M + L.
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.
LMNN2 :	Dimensioning parameter, must be set to M + 2*N + 3*L + 2 when calling NLPLSQ.
X(LNMAX) :	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 NLPLSQ 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, LNMAX) : The array is used to pass gradients of residuals and constraints to NLPLSQ 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(LNMAX), XU(LNMAX) : 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 : The user must indicate a non-negative guess for the approximate size of the least squares residual, i.e., a low positive real number if the residual is supposed to be small, and a large one in the order of 1 if the residual is supposed to be large.

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). If MAXNM=0, monotone line search is performed. MAXNM should not be greater than 50.

RHO : Parameter for initializing a restart in case of IFAIL=2 by setting the BFGS-update matrix to RHO*I, where I denotes the identity matrix. The number of restarts is bounded by MAXFUN. No restart is performed if RHO 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+L>LMMAX, N+L≥LNMAX, or LMNN2≠M+N+N+3*L+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 LNMAX*LNMAX + 27*LNMAX + 8*LMMAX + 150 + MAX0(5*LNMAX*LNMAX/2 + 17*LNMAX + M*LNMAX + 3*M + 20,(N+1)*L)
KWA(LKWA) :	KWA is an integer working array of length LKWA.
LKWA :	Length of the integer working array KWA. LKWA must be at least LNMAX + 40.
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*LMMAX+10.

If $M > 0$ and the starting point is feasible or if RESIZ<SQRT(ACC), starting values for auxiliary variables are set internally to initial residual values found in FUNC.

NLPLSQ must be linked with the calling routine of the user, the SQP code NLPQLP, and the quadratic programming code QL.

7 Program Organization

NLPLSQ is implemented in form of a Fortran subroutine, where all declarations of real numbers must be done in double precision. Model functions and gradients are passed by reverse communication. The user has to provide functions and gradients in the same program which executes NLPLSQ, 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 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 NLPQLP.

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

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

7. If NLPQLP 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.

8 Examples

To give a simple example how to organize the code in case of two explicitly given functions, we consider again Rosenbrock's banana function, see Example 3.1 or test problem TP1 of Hock and Schittkowski [14],

$$x_1, x_2 \in I\!R: \ \min 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{23}$$

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

```
      IMPLICIT          NONE
      INTEGER           L, N, LNMAX, M, LMMAX, LMNN2, LWA, LKWA, LLOGWA
      PARAMETER        (L = 2, N = 2, M = 0, LNMAX = N+L+1, LMMAX = L)
      PARAMETER        (LWA = LNMAX*LNMAX + 27*LNMAX + 8*LMMAX + 150
     /                       + MAX0(5*LNMAX*LNMAX/2 + 17*LNMAX
     /                                + M*LNMAX + 3*M + 20,(N+1)*L),
     /                  LMNN2  = M + 2*N + 3*L + 2,
     /                  LKWA   = LNMAX + 40,
     /                  LLOGWA = 2*(L+M) + 10)
      INTEGER           ME, MAXFUN, MAXIT, IPRINT, MAXNM, IOUT, IFAIL,
     /                  KWA
      DOUBLE PRECISION  X, FUNC, RES, GRAD, U, XL, XU, ACC, ACCQP,
     /                  RESSIZ, RHO, WA
      DIMENSION         X(LNMAX), FUNC(LMMAX), GRAD(LMMAX,LNMAX),
     /                  U(LMNN2), XL(LNMAX), XU(LNMAX),
     /                  WA(LWA), KWA(LKWA), LOGWA(LLOGWA)
      LOGICAL           LOGWA
C
C   set parameters
C
      ME     = 0
      ACC    = 1.0D-18
      ACCQP  = ACC
      RESSIZ = ACC
      RHO    = 1.0D2
      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 NLPLSQ in 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 NLPLSQ
C
      CALL NLPLSQ (      M,     ME,  LMMAX,      L,      N,
     /               LNMAX,  LMNN2,      X,   FUNC,    RES,
     /                GRAD,      U,     XL,     XU,    ACC,
     /               ACCQP, RESSIZ, MAXFUN,  MAXIT,  MAXNM,
     /                 RHO, IPRINT,   IOUT,  IFAIL,     WA,
     /                 LWA,    KWA,   LKWA,  LOGWA,  LLOGWA)
      IF (IFAIL.LT.0) GOTO 1
C
C   end of program
C
      STOP
      END
```

The following output should appear on screen:

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

  Parameters:
     N      =       4
     M      =       2
     ME     =       2
     MODE   =       1
```

```
   ACC    =   0.1000D-17
   ACCQP  =   0.1000D-13
   STPMIN =   0.1000D-14
   RHO    =   0.0000D+00
   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  0.12100000D+02  0.00D+00    2  0  0.00D+00  0.00D+00  0.24D+02
   2  0.54314750D-25  0.48D+02    2  1  0.10D+01  0.00D+00  0.47D-12
   3  0.18785906D-25  0.97D-14    2  1  0.10D+01  0.00D+00  0.38D-25

   Objective function value:      F(X)  =  0.18785906D-25
   Solution values:               X     =
      0.10000000D+01  0.10000000D+01 -0.96837752D-14  0.19359245D-12
   Distances from lower bounds:   X-XL  =
      0.10000100D+06  0.10000100D+06  0.10000000D+31  0.10000000D+31
   Distances from upper bounds:   XU-X  =
      0.99999000D+05  0.99999000D+05  0.10000000D+31  0.10000000D+31
   Multipliers for lower bounds:  U     =
      0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
   Multipliers for upper bounds:  U     =
      0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
   Constraint values:             G(X)  =
      0.96837752D-14  0.30445895D-16
   Multipliers for constraints:   U     =
      0.77448571D-27 -0.11116847D-25
   Number of function calls:      NFUNC =      3
   Number of gradient calls:      NGRAD =      3
   Number of calls of QP solver:  NQL   =      3

  --- Final Convergence Analysis of NLPLSQ ---
```

```
 Sum of squared functions:      RES(X) =  0.37489826D-25
 Function values:               F(X) =
    0.00000000D+00  0.19362290D-12
 Solution:                      X =
    0.10000000D+01  0.10000000D+01
 Multiplier values:             U =
    0.00000000D+00  0.00000000D+00  0.00000000D+00  0.00000000D+00
 Constraint values:             G(X) =

 Number of function calls:      NFUNC =   3
 Number of derivative calls:    NGRAD =   3
```

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

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

$x = (x_1, \ldots, x_4)^T$. 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           L, N, LNMAX, M, LMMAX, LMNN2, LWA, LKWA, LLOGWA
      PARAMETER        (L = 11, N = 4, M = 2, LNMAX = N+L+1,
     /                  LMMAX = M+L)
      PARAMETER        (LWA = LNMAX*LNMAX + 25*LNMAX + 7*LMMAX + 150
     /                       + MAX0(5*LNMAX*LNMAX/2 + 17*LNMAX
     /                                 + M*LNMAX + 3*M + 20,(N+1)*L),
     /                  LMNN2  = M + 2*N + 3*L + 2,
     /                  LKWA = LNMAX + 40,
     /                  LLOGWA = 2*(L+M) + 10)
      INTEGER           ME, MAXFUN, MAXIT, IPRINT, MAXNM, IOUT, IFAIL,
     /                  KWA, I, J
      DOUBLE PRECISION  X, FUNC, RES, GRAD, U, XL, XU, ACC,
     /                  ACCQP, RESSIZ, RHO, WA, T, Y, W
      DIMENSION         X(LNMAX), FUNC(LMMAX), GRAD(LMMAX,LNMAX),
     /                  U(LMNN2), XL(LNMAX), XU(LNMAX),
     /                  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/
C
C   set parameters
C
      ME     = 2
      ACC    = 1.0D-13
      ACCQP  = 1.0D-14
      RESSIZ = 1.0D-4
      RHO    = 1.0D2
      MAXFUN = 20
      MAXIT  = 100
      MAXNM  = 10
      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 NLPLSQ 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 NLPLSQ
C
       CALL NLPLSQ (      M,     ME,    L+M,      L,      N,
     /               L+N+1,  LMNN2,      X,   FUNC,    RES,
     /                GRAD,      U,     XL,     XU,    ACC,
     /               ACCQP, RESSIZ, MAXFUN,  MAXIT,  MAXNM,
     /                 RHO, IPRINT,   IOUT,  IFAIL,     WA,
     /                 LWA,    KWA,   LKWA,  LOGWA,  LLOGWA)
       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
                  NLPQLP, Version 4.04 (Apr 2013)
 ------------------------------------------------------------------------

 Parameters:
    N      =      15
    M      =      13
    ME     =      13
    MODE   =       1
    ACC    =   0.1000D-12
    ACCQP  =   0.1000D-13
    STPMIN =   0.1000D-14
    RHO    =   0.1000D+03
    MAXFUN =      20
    MAXNM  =      10
    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  0.25D+00   13  0  0.00D+00  0.00D+00  0.11D-02
   2  0.42502103D-04  0.11D+00   13  2  0.47D+00  0.00D+00  0.51D-03
   3  0.20847571D-03  0.10D-01   13  1  0.10D+01  0.00D+00  0.51D-04
   4  0.20644006D-03  0.18D-03   13  1  0.10D+01  0.00D+00  0.11D-05
   5  0.20647850D-03  0.13D-04   13  1  0.10D+01  0.00D+00  0.95D-07
   6  0.20648135D-03  0.23D-05   13  1  0.10D+01  0.00D+00  0.19D-07
   7  0.20648571D-03  0.71D-11   13  1  0.10D+01  0.00D+00  0.49D-13

    Objective function value:      F(X)  =  0.20648571D-03
    Solution values:               X     =
```

```
    0.19226325D+00  0.40401714D+00  0.27497963D+00  0.20678888D+00
   -0.19718224D-22  0.46890274D-02  0.27982019D-03  0.54681102D-02
    0.39112775D-02  0.26393845D-02  0.85962230D-02 -0.13764508D-01
    0.86748133D-02 -0.36381032D-03  0.26469780D-22
  Distances from lower bounds:   X-XL  =
    0.19226325D+00  0.40401714D+00  0.27497963D+00  0.20678888D+00
    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
  Distances from upper bounds:   XU-X  =
    0.99999808D+05  0.99999596D+05  0.99999725D+05  0.99999793D+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
  Multipliers for lower bounds:  U     =
    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
  Multipliers for upper bounds:  U     =
    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)  =
   -0.49789686D-12 -0.55888801D-12 -0.62875356D-12 -0.73018067D-12
   -0.84732828D-12 -0.97836192D-12 -0.10497679D-11 -0.75764048D-12
   -0.34565580D-12 -0.11539576D-12 -0.26173508D-13 -0.49789686D-12
   -0.26173508D-13
  Multipliers for constraints:   U     =
    0.10302965D-24 -0.46890274D-02 -0.27982020D-03 -0.54681102D-02
   -0.39112775D-02 -0.26393845D-02 -0.85962230D-02  0.13764508D-01
   -0.86748129D-02  0.36381057D-03 -0.56723318D-25  0.26628371D-01
    0.18673631D-02
  Number of function calls:      NFUNC =      8
  Number of gradient calls:      NGRAD =      7
  Number of calls of QP solver:  NQL   =      7


--- Final Convergence Analysis of NLPLSQ ---

  Sum of squared functions:      RES(X) =  0.41297141D-03
  Function values:               F(X) =
   -0.49789686D-12  0.46890274D-02  0.27982019D-03  0.54681102D-02
    0.39112774D-02  0.26393845D-02  0.85962230D-02 -0.13764508D-01
    0.86748133D-02 -0.36381032D-03 -0.26173508D-13
```

```
Solution:                        X =
   0.19226325D+00  0.40401714D+00  0.27497963D+00  0.20678888D+00
Multiplier values:               U =
   0.26628371D-01  0.18673631D-02  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) =
  -0.49789686D-12 -0.26173508D-13
Number of function calls:        NFUNC =   8
Number of derivative calls:      NGRAD =   7
```

9 Summary

We presented a modification of a Gauss-Newton method with the goal to apply an available *black-box* SQP solver and to retain the excellent convergence properties of Gauss-Newton-type algorithms. The idea is to introduce additional variables and nonlinear equality constraints and to solve the transformed problem by an SQP method. The method is outlined, some comparative performance results are obtained, and the usage of the code is documented.

References

[1] Bard Y. (1970): *Comparison of gradient methods for the solution of nonlinear parameter estimation problems*, SIAM Journal on Numerical Analysis, Vol. 7, 157-186

[2] Björck A. (1990): *Least Squares Methods,* Elsevier, Amsterdam

[3] Dennis J.E. (1973): *Some computational techniques for the nonlinear least squares problem*, in: *Numerical Solution of Systems of Nonlinear Algebraic Equations* , G.D. Byrne, C.A. Hall eds., Academic Press, London, New York

[4] Dennis J.E. (1977): *Nonlinear least squares*, in: *The State of the Art in Numerical Analysis* , D. Jacobs ed., Academic Press, London, New York

[5] Dennis J.E.jr., Gay D.M., Welsch R.E. (1981): *An adaptive nonlinear least-squares algorithm,* ACM Transactions on Mathematical Software, Vol. 7, No. 3, 348-368

[6] Dobmann M., Liepelt M., Schittkowski K. (1995): *Algorithm 746: PCOMP: A Fortran code for automatic differentiation,* ACM Transactions on Mathematical Software, Vol. 21, No. 3, 233-266

[7] Fernanda M., Costa P., Fernandes M.G.P. (2005): *A primal-dual interior-point algorithm for nonlinear least squares constrained problems*, TOP, Vol. 13, 1863-8279

[8] Fraley C. (1988): *Software performance on nonlinear least-squares problems*, Technical Report SOL 88-17, Dept. of Operations Research, Stanford University, Stanford, CA 94305-4022, USA

[9] Gill P.E., Murray W. (1978): *Algorithms for the solution of the non-linear least-squares problem*, CIAM Journal on Numerical Analysis, Vol. 15, 977-992

[10] Gill P.E., Murray W., Wright M.H. (1981): *Practical Optimization*, Academic Press, London, New York, Toronto, Sydney, San Francisco

[11] Gill P.E., Murray W., Saunders M., Wright M.H. (1983): *User's Guide for SQL/NPSOL: A Fortran package for nonlinear programming*, Report SOL 83-12, Dept. of Operations Research, Standford University, California

[12] Hanson R.J., Frogh F.T. (1992): *A quadratic-tensor model algorithm for nonlinear least-squares problems with linear constraints,* ACM Transactions on Mathematical Software, Vol. 18, No. 2, 115-133

[13] Hiebert K. (1979): *A comparison of nonlinear least squares software*, Sandia Technical Report SAND 79-0483, Sandia National Laboratories, Albuquerque, New Mexico

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

[15] Holt J.N., Fletcher R. (1979): *An algorithm for constrained non-linear least-squares*, IMA Journal on Applied Mathematics, Vol. 23, 449-463

[16] Levenberg K. (1944): *A method for the solution of certain problems in least squares,* Quarterly of Applied Mathematics, Vol. 2, 164-168

[17] Lindström P. (1982): *A stabilized Gauß-Newton algorithm for unconstrained least squares problems*, Report UMINF-102.82, Institute of Information Processing, University of Umea, Umea, Sweden

[18] Lindström P. (1983): *A general purpose algorithm for nonlinear least squares problems with nonlinear constraints*, Report UMINF-103.83, Institute of Information Processing, University of Umea, Umea, Sweden

[19] Mahdavi-Amiri N. (1981): *Generally constrained nonlinear least squares and generating nonlinear programming test problems: Algorithmic approach*, Dissertation, The John Hopkins University, Baltimore, Maryland, USA

[20] Mahdavi-Amiri N., Bartels R.H. (1989): *Constrained nonlinear least squares: an exact penalty approach with projected structured quasi-Newton updates*, ACM Transactions on Mathematical Software (TOMS), Vol. 15, 220-242

[21] Marquardt D. (1963): *An algorithm for least-squares estimation of nonlinear parameters,* SIAM Journal on Applied Mathematics, Vol. 11, 431-441

[22] Moré J.J. (1977): *The Levenberg-Marquardt algorithm: implementation and theory,* in: Numerical Analysis, G. Watson ed., Lecture Notes in Mathematics, Vol. 630, Springer, Berlin

[23] 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, Berlin

[24] 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, New York, London

[25] Ramsin H., Wedin P.A. (1977): *A comparison of some algorithms for the nonlinear least squares problem*, Nordisk Tidstr. Informationsbehandlung (BIT), Vol. 17, 72-90

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

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

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

[29] 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

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

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

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

[33] 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

[34] 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

[35] 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

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

[37] 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