NLPQLP: A Fortran Implementation of a Sequential Quadratic Programming Algorithm with Distributed and Non-Monotone Line Search

- User's Guide, Version 4.2 -

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, 2015

Abstract

The Fortran subroutine NLPQLP solves smooth nonlinear programming problems by a sequential quadratic programming (SQP) algorithm. This version is specifically tuned to run under distributed systems controlled by an input parameter. In case of computational errors as for example caused by inaccurate function or gradient evaluations, a non-monotone line search is activated. Numerical results are included which show that in case of noisy function values, a significant improvement of the performance is achieved compared to the version with monotone line search. Further stabilization is obtained by performing internal restarts in case of errors when computing the search direction due to inaccurate derivatives. The new version of NLPQLP successfully solves more than 90 % of our 306 test examples subject to a stopping tolerance of 10^{-7}, although at most two digits in function values are correct in the worst case and although numerical differentiation leads to additional truncation errors. In addition, automated initial and periodic scaling with restarts is implemented. 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 code

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 original code NLPQL of Schittkowski [49] is a Fortran implementation of a sequential quadratic programming (SQP) algorithm. The numerical algorithm is based on extensive comparative numerical tests, see Schittkowski [42, 46, 44], Schittkowski et al. [60], Hock and Schittkowski [26], and on further theoretical investigations published in [43, 45, 47, 48].

The algorithm is extended to solve also nonlinear least squares problems efficiently, see [51] and [53], and to handle problems with very many constraints, see [56]. Extremely noisy problems can be efficiently solved by a non-monotone line search, see [59] or Dai and Schittkowski [13]. Moreover, the code is extended to take advantage of parallel execution of function values, see [58].

To conduct the numerical tests, a random test problem generator is developed for a major comparative study, see [42]. Two collections with together 306 test problems are published in Hock and Schittkowski [26] and in Schittkowski [50]. Fortran source codes and a test frame can be downloaded from the home page of the author,

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

Many of them became part of the Cute test problem collection of Bongartz et al. [8]. About 80 test problems based on a Finite Element formulation are collected for a comparative evaluation in Schittkowski et al. [60]. A set of 1,300 least squares test problems solved by an extension of the code NLPQL to retain typical features of a Gauss-Newton algorithm, is described in [53]. Also these problems can be downloaded from the home page of the author together with an interactive software system called EASY-FIT, see [54].

Moreover, there exist hundreds of commercial and academic applications of NLPQL, for example

1. mechanical structural optimization, see Schittkowski, Zillober, Zotemantel [60] and Kneppe, Krammer, Winkler [29],
2. data fitting and optimal control of transdermal pharmaceutical systems, see Boderke, Schittkowski, Wolf [3] or Blatt, Schittkowski [7],
3. computation of optimal feed rates for tubular reactors, see Birk, Liepelt, Schittkowski, and Vogel [6],
4. food drying in a convection oven, see Frias, Oliveira, and Schittkowski [16],
5. optimal design of horn radiators for satellite communication, see Hartwanger, Schittkowski, and Wolf [24],
6. receptor-ligand binding studies, see Schittkowski [52],
7. optimal design of surface acoustic wave filters for signal processing, see Bünner, Schittkowski, and van de Braak [9].

Previous and present versions of NLPQLP are part of commercial libraries, modeling systems, or optimization systems like

- ANSYS/POPT (CAD-FEM, Grafing) for structural optimization,
- DesignXplorer (ANSYS, Canonsburg) for structural design optimization,
- STRUREL (RCP, Munich) for reliability analysis,
- TEMPO (OECD Reactor Project, Halden) for control of power plants,
- Microwave Office Suit (Applied Wave Research, El Segundo) for electronic design,
- MOOROPT (Marintek, Trondheim) for the design of mooring systems,
- iSIGHT (Simulia/Dassault) for multi-disciplinary CAE,
- POINTER (Synaps, Atlanta) for design automation,
- EXCITE (AVL, Graz) for non-linear dynamics of power units,
- ModeFRONTIER (ESTECO, Trieste) for integrated multi-objective and multi-disciplinary design optimization,

- TOMLAB/MathLab (Tomlab Optimization, Västeras, Sweden) for general nonlinear programming, least squares optimization, data fitting in dynamical systems,
- EASY-FIT (Schittkowski, Bayreuth) for data fitting in dynamical systems,
- OptiSLang (DYNARDO, Weimar), for structural design optimization,
- AMESim (IMAGINE, Roanne), for multidisciplinary system design,
- LMS OPTIMUS (NOESIS, Leuven, Belgium) for multi-disciplinary CAE,
- RADIOSS/M-OPT (MECALOG/Altair, Antony, France) for multi-disciplinary CAE,
- CHEMASIM (BASF, Ludwigshafen) for the design of chemical reactors.

Customers include, among many others, Airbus, AMD, Aramco, Astrium, BASF, Bayer, Bell Labs, BMW, Chevron Research, DLR, Dow Chemical, DuPont, EADS, EMCOSS, ENSIGC, EPCOS, ESA-ESOC, Eurocopter, Fantoft Prosess, General Electric, Hoechst, Hidroelectrica Espanola, IABG, IBM, Institute for Energy Technology Halden, KFZ Karlsruhe, Kongsberg Maritime, Lockheed Martin, Loral Space Systems, MAN, Markov Processes, Marintek, MTU, NASA Langley, NASA Ames, Nevesbu, National Airspace Laboratory, Norsk Hydro Research, Norwegian Computing Center, Norwegian Defense Agency, OECD Halden, Philips, Polysar, ProSim, Rolls-Royce, Shell, Siemens, Sintef, Solar Turbines, Statoil, TNO, Transpower, USAF Research Lab, Wright R & D Center and in addition dozens of academic research institutions all over the world.

The general availability of parallel computers and in particular of distributed computing in networks motivates a careful redesign of the original implementation NLPQL to allow simultaneous function evaluations. The resulting extensions are implemented and the code is called NLPQLP. An additional input parameter l is introduced for the number of parallel machines, that is the number of function calls to be executed simultaneously. In case of $l = 1$, NLPQLP is more or less identical to NLPQL besides of additional changes of the code. Otherwise, the line search procedure is modified to allow parallel function calls, which can also be applied for approximating gradients by difference formulae. The mathematical background is outlined, in particular the modification of the line search algorithm to retain convergence under parallel systems. It must be emphasized that distributed computation of function values is only simulated throughout the paper. It is up to the user to adopt the code to a particular parallel environment.

However, SQP methods are quite sensitive subject to round-off or any other errors in function and especially gradient values. If objective or constraint functions cannot be computed within machine accuracy or if the accuracy by which gradients are approximated is above the termination tolerance, the code could break down typically with the error message IFAIL=4. In this situation, the line search cannot be terminated within a given number of iterations and the algorithm is stopped.

All new versions since 2.0 makes use of non-monotone line search in the error situation described above. The idea is to replace the reference value of the line search termination check, $\psi_{r_k}(x_k, v_k)$, by

$$\max\{\psi_{r_j}(x_j, v_j) : j = k - p, \ldots, k\} \ ,$$

where $\psi_r(x, v)$ is a merit function and p a given parameter. The general idea is not new and for example described in Dai [12], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [19], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [5], Deng et al. [14], Grippo et al. [20, 21], Ke and Han [27], Ke et al. [28], Lucidi et al. [31], Panier and Tits [35], Raydan [40], and Toint [62, 63]. However, there is a basic difference in the methodology: Our goal is to allow monotone line searches as long as they terminate successfully, and to apply a non-monotone one only in a special error situation.

Despite of strong analytical results, SQP methods do not always terminate successfully. Besides of the difficulties leading to the usage of non-monotone line search, it might happen that the search direction as computed from a certain quadratic programming subproblem, is not a downhill direction of the merit function needed to perform a line search. Possible reasons are again severe errors in function and especially gradient evaluations, or a violated regularity condition concerning linear independency of gradients of active constraints (LICQ). In the latter case, the optimization problem is not modeled in a suitable way to solve it directly by an SQP method. Our new version performs an automated restart as soon as a corresponding error message appears. The BFGS quasi-Newton matrix is reset to a multiple of the identity matrix and the matrix update procedure starts from there.

Scaling is an extremely important issue and an efficient procedure is difficult to derive in the general case without knowing too much about the numerical structure of the optimization problem. If requested by the user, the first BFGS update is started from a multiple of the identity matrix, which takes into account information from the solution of the initial quadratic programming subproblem. This restart can be repeated periodically with successively adapted scaling parameters.

In Section 2 we outline the general mathematical structure of an SQP algorithm, the non-monotone line search, and the modifications to run the code under distributed systems. Section 3 contains some numerical results obtained for a set of 306 standard test problems of the collections published in Hock and Schittkowski [26] and in Schittkowski [50]. They show the sensitivity of the new version with respect to the number of parallel machines and the influence of gradient approximations under uncertainty. Moreover, we test the non-monotone line search versus the monotone one, and generate noisy test problems by adding random errors to function values and by inaccurate gradient approximations. This situation appears frequently in practical environments, where complex simulation codes prevent accurate responses and where gradients can only be computed by a difference formula. The usage of the Fortran subroutine is documented in Section 4 and Section 5 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 [61] and an excellent review is given by Boggs and Tolle [4]. From the more practical point of view, SQP methods are also introduced in the books of Papalambros, Wilde [36] and Edgar, Himmelblau [15]. Their excellent numerical performance is tested and compared with other methods in Schittkowski [42], 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

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

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 $C_k \in I\!R^{n \times n}$, an approximation of the Hessian of the Lagrangian function. Then one has to solve the quadratic programming problem

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

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 C_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 [49] 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 $v_0 = 0$, $C_0 = I$. 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

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

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 [47]. 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 [47] or Wolfe [64] in a more general setting, i.e., to get

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

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 C_k by the BFGS quasi-Newton formula, cf. Powell [38] or Stoer [61].

The implementation of a line search algorithm is a critical issue when implementing a nonlinear programming algorithm, and has significant effect on the overall efficiency of the resulting code. On the one hand we need a line search to stabilize the algorithm, on the other hand it is not desirable to waste too many function calls. Moreover, the behavior of the merit function becomes irregular in case of constrained optimization because of very steep slopes at the border caused by large penalty terms. Even the implementation is more complex than shown above, if linear constraints and bounds of the variables are to be satisfied during the line search.

Usually, the steplength parameter α_k is chosen to satisfy the Armijo [1] condition

$$\phi_r(\sigma\beta^i) \leq \phi_r(0) + \sigma\beta^i\mu\phi'_r(0) \ \ , \tag{9}$$

see for example Ortega and Rheinboldt [34]. The constants are from the ranges $0 < \mu < 0.5$, $0 < \beta < 1$, and $0 < \sigma \le 1$. We start with $i = 0$ and increase i until (9) is satisfied for the first time, say at i_k. Then the desired steplength is $\alpha_k = \sigma \beta^{i_k}$.

Fortunately, SQP methods are quite robust and accept the steplength one in the neighborhood of a solution. Typically the test parameter μ for the Armijo-type sufficient descent property (9) is very small. Nevertheless the choice of the reduction parameter β must be adopted to the actual slope of the merit function. If β is too small, the line search terminates very fast, but on the other hand the resulting stepsizes are usually too small leading to a higher number of outer iterations. On the other hand, a larger value close to one requires too many function calls during the line search. Thus, we need some kind of compromise, which is obtained by first applying a polynomial interpolation, typically a quadratic one, and use (9) only as a stopping criterion. Since $\phi_r(0)$, $\phi_r'(0)$, and $\phi_r(\alpha_i)$ are given, α_i the actual iterate of the line search procedure, we easily get the minimizer of the quadratic interpolation. We accept then the maximum of this value and the Armijo parameter as a new iterate, as shown by the subsequent code fragment implemented in NLPQLP.

Algorithm 2.1 *Let β, μ with $0 < \beta < 1$, $0 < \mu < 0.5$ be given.*

```
Start: $\alpha_0 := 1$
For $i = 0, 1, 2, \ldots$ do:
  1) If $\phi_r(\alpha_i) < \phi_r(0) + \mu\, \alpha_i\, \phi_r'(0)$, then stop.
  2) Compute $\bar{\alpha}_i := \dfrac{0.5\, \alpha_i^2\, \phi_r'(0)}{\alpha_i \phi_r'(0) - \phi_r(\alpha_i) + \phi_r(0)}$.
  3) Let $\alpha_{i+1} := \max(\beta\, \alpha_i, \bar{\alpha}_i)$.
```

Corresponding convergence results are found in Schittkowski [47]. $\bar{\alpha}_i$ is the minimizer of the quadratic interpolation, and we use the Armijo descent property for checking termination. Step 3 is required to avoid irregular values, since the minimizer of the quadratic interpolation could be outside of the feasible domain $(0, 1]$. The search algorithm is implemented in NLPQLP together with additional safeguards, for example to prevent violation of bounds. Algorithm 4.1 assumes that $\phi_r(1)$ is known before calling the procedure, i.e., that the corresponding function values are given. We have to stop the algorithm, if sufficient descent is not observed after a certain number of iterations, say 10. If the tested stepsize falls below machine precision or the accuracy by which model function values are computed, the merit function cannot decrease further.

To outline the new approach, let us assume that functions can be computed simultaneously on l different machines. Then l test values $\alpha_i = \beta^{i-1}$ with $\beta = \epsilon^{1/(l-1)}$ are selected, $i = 1, \ldots, l$, where ϵ is a guess for the machine precision. Next we require l parallel function calls to get the corresponding model function values. The first α_i satisfying a sufficient descent property (9), say for $i = i_k$, is accepted as the new steplength to set the subsequent iterate by $\alpha_k := \alpha_{i_k}$. One has to be sure that existing convergence results of the SQP algorithm are not violated.

The proposed parallel line search will work efficiently, if the number of parallel machines l is sufficiently large, and works as follows, where we omit the iteration index k.

Algorithm 2.2 *Let β, μ with $0 < \beta < 1$, $0 < \mu < 0.5$ be given.*

```
Start: For $\alpha_i = \beta^i$ compute $\phi_r(\alpha_i)$ for $i = 0, \ldots, l-1$.
For $i = 0, 1, 2, \ldots$ do:
    If $\phi_r(\alpha_i) < \phi_r(0) + \mu\ \alpha_i\ \phi_r'(0)$, then stop.
```

To precalculate l candidates in parallel at log-distributed points between a small tolerance $\alpha = \tau$ and $\alpha = 1$, $0 < \tau << 1$, we propose $\beta = \tau^{1/(l-1)}$.

In a typical situation we suppose that there is a complex application code providing simulation data, for example by an expensive Finite Element calculation in mechanical structural optimization. It is supposed that various instances of the simulation code providing function values, are executable on a series of different machines, so-called slaves, controlled by a master program that executes NLPQLP. By a message passing system, for example PVM, see Geist et al. [17], only very few data need to be transferred from the master to the slaves. Typically only a set of design parameters of length n must to be passed. On return, the master accepts new model responses for objective function and constraints, at most $m + 1$ double precision numbers. All massive numerical calculations and model data, for example the stiffness matrix of a Finite Element model in a mechanical engineering application, remain on the slave processors of the distributed system.

In both situations, i.e., the serial or parallel version, it is still possible that Algorithm 2.1 or Algorithm 2.2 breaks down because to too many iterations. In this case, we proceed from a descent direction of the merit function, but $\phi_r'(0)$ is extremely small. To avoid interruption of the whole iteration process, the idea is to repeat the line search with another stopping criterion. Instead of testing (9), we accept a stepsize α_k as soon as the inequality

$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)<=j<=k} \phi_{r_j}(0) + \alpha_k \mu \phi_{r_k}'(0) \tag{10}$$

is satisfied, where $p(k)$ is a predetermined parameter with $p(k) = \min\{k, p\}$, p a given tolerance. Thus, we allow an increase of the reference value $\phi_{r_{j_k}}(0)$ in a certain error situation, i.e., an increase of the merit function value. To implement the non-monotone line search, we need a queue consisting of merit function values at previous iterates. In case of $k = 0$, the reference value is adapted by a factor greater than 1, i.e., $\phi_{r_{j_k}}(0)$ is replaced by $t\phi_{r_{j_k}}(0)$, $t > 1$. The basic idea to store reference function values and to replace the sufficient descent property by a sufficient 'ascent' property in max-form, is for example described in Dai [12], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [19], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [5], Deng et al. [14], Grippo et al. [20, 21], Ke and Han [27], Ke et al. [28], Lucidi et al. [31], Panier and Tits [35], Raydan [40], and Toint [62, 63]. However, there is a difference in the methodology: Our goal is to allow monotone line searches as long as they terminate successfully, and to apply a non-monotone one only in an error situation.

The final step of an SQP method consists of updating the quasi-Newton matrix C_k, e.g., by the BFGS formula

$$C_{k+1} := C_k + \frac{q_k q_k^T}{p_k^T q_k} - \frac{C_k p_k p_k^T C_k}{p_k^T C_k p_k} \quad , \tag{11}$$

where $q_k := \bigtriangledown_x L(x_{k+1}, u_k) - \bigtriangledown_x L(x_k, u_k)$ and $p_k := x_{k+1} - x_k$. Special safeguards guarantee that $p_k^T q_k > 0$ and that thus all matrices C_k remain positive definite provided that C_0 is positive definite. A possible scaling factor and restart procedure is to replace an actual C_k by $\gamma_k I$ before performing the update (11), where $\gamma_k = \frac{p_k^T q_k}{p_k^T p_k}$ and where I denotes the identity matrix, see for example Liu and Nocedal [30]. Scaled restarts are recommended, if, e.g., the convergence turns out to become extremely slow.

3 Performance Evaluation

3.1 The Test Environment

Our numerical tests use the 306 academic and real-life test problems published in Hock and Schittkowski [26] and in Schittkowski [50]. Part of them are also available in the Cute library, see Bongartz et. al [8], and their usage is described in Schittkowski [57].

Since analytical derivatives are not available for all problems, we approximate them numerically. The test examples are provided with exact solutions, either known from analytical precalculations *by hand* or from the best numerical data found so far.

First we need a criterion to decide whether the result of a test run is considered as a successful return or not. Let $\epsilon > 0$ be a tolerance for defining the relative accuracy, x_k the final iterate of a test run, and $x^\star$ the supposed exact solution known from the test problem collection. Then we call the output a successful return, if the relative error in the objective function is less than ϵ and if the maximum constraint violation is less than ϵ^2, i.e., if

$$f(x_k) - f(x^\star) < \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) \neq 0$$

or

$$f(x_k) < \epsilon \ , \ \text{if } f(x^\star) = 0$$

and

$$r(x_k) = \|g(x_k)^-\|_\infty < \epsilon^2 \ ,$$

where $\| \dots \|_\infty$ denotes the maximum norm and $g_j(x_k)^- = \min(0, g_j(x_k))$, $j > m_e$, and $g_j(x_k)^- = g_j(x_k)$ otherwise.

We take into account that a code returns a solution with a better function value than the known one, subject to the error tolerance of the allowed constraint violation. However, there is still the possibility that an algorithm terminates at a local solution different from the known one. Thus, we call a test run a successful one, if in addition to the above decision the internal termination conditions are satisfied subject to a reasonably small tolerance (*IFAIL=0*), and if

$$f(x_k) - f(x^\star) \geq \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) \neq 0$$

or

$$f(x_k) \geq \epsilon \ , \ \text{if } f(x^\star) = 0$$

and

$$r(x_k) < \epsilon^2 \ .$$

For our numerical tests, we use $\epsilon = 0.01$ to determine a successful return, i.e., we require a final accuracy of one per cent. Note that in all cases, NLPQLP is called with a termination tolerance of 10^{-7}.

If gradients are not available in analytical form, they must be approximated in a suitable way. The three most popular difference formulae are the following ones:

1. Forward differences:
$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{\eta_i} \Big(f(x + \eta_i e_i) - f(x) \Big) \qquad (12)$$

2. Two-sided differences:
$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{2\eta_i} \Big(f(x + \eta_i e_i) - f(x - \eta_i e_i \Big) \qquad (13)$$

3. Fourth-order formula:
$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\eta_i} \Big(2f(x - 2\eta_i e_i) - 16f(x - \eta_i e_i) + 16f(x + \eta_i e_i) - 2f(x + 2\eta_i e_i) \Big) \quad (14)$$

Here $\eta_i = \eta \max(10^{-5}, |x_i|)$ and e_i is the i-th unit vector, $i = 1, \dots, n$. The tolerance η depends on the difference formula and is set to $\eta = {\eta_m}^{1/2}$ for forward differences, $\eta = {\eta_m}^{1/3}$ for two-sided differences, and $\eta = (\eta_m/72)^{1/4}$ for fourth-order formulae. η_m is a guess for the accuracy by which function values are computed, i.e., either machine accuracy in case of analytical formulae or an estimate of the noise level in function computations. In a similar way, derivatives of constraints are computed.

The Fortran implementation of the SQP method introduced in the previous section, is called NLPQLP. The code represents the most recent version of NLPQL which is frequently used in academic and commercial institutions. NLPQLP is prepared to run also under distributed systems, but behaves in exactly the same way as the serial version, if the number of simulated processors is set to one. Functions and gradients must be provided by reverse communication and the quadratic programming subproblems are solved by the primal-dual method of Goldfarb and Idnani [18] based on numerically stable orthogonal decompositions. NLPQLP is executed with termination accuracy ACC=10^{-7} as mentioned already above, and a maximum number of iterations *MAXIT=500*.

In the subsequent tables, we use the notation

n_{succ}	-	number of successful test runs (according to above definition)
n_{func}	-	average number of function evaluations
n_{grad}	-	average number of gradient evaluations or iterations, respectively
$f(x)$	-	final objective function value
$r(x)$	-	final constraint violation
i_{fail}	-	failure code

To get n_{func} or n_{grad}, we count each evaluation of a whole set of function or gradient values, respectively, for a given iterate x_k, also in the case of several simulated processors, $l > 1$. However, additional function evaluations needed for gradient approximations, are not counted for n_{func}. Their average number is n_{func} for forward differences, $2 \times n_{func}$ for two-sided differences, and $4 \times n_{func}$ for fourth-order formulae. One gradient computation corresponds to one iteration of the SQP method.

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 11.0, 64 bit, under Windows 7 and Intel(R) Core(TM) i7-2720 CPU, 2.2 GHz, with 8 GB RAM.

3.2 Testing Distributed Function Calls

First we investigate the question, how parallel line searches influence the overall performance. Table 1 shows the number of successful test runs and the average number of iterations or gradient evaluations, n_{it}, for an increasing number of simulated parallel calls of model functions denoted by n_p. The forward difference formula (12) is used for gradient approximations and non-monotone line search is applied with a queue size of $p = 40$. Calculation time is about one second for solving all 306 test problems without random perturbations.

$n_p = 1$ corresponds to the serial case, when Algorithm 2.1 is applied to the line search consisting of a quadratic interpolation combined with an Armijo-type bisection strategy and a non-monotone stopping criterion.

n_p	n_{succ}	n_{grad}	l	n_{succ}	n_{grad}
1	306	20	8	305	28
3	228	74	9	306	23
4	289	141	10	306	23
5	303	80	15	304	20
6	305	42	20	305	26
7	305	33	50	305	19

Table 1: Performance Results for Parallel Line Search

In all other cases, $n_p > 1$ simultaneous function evaluations are made according to Algorithm 2.2. To get a reliable and robust line search, one should use at least seven parallel processors. No significant improvements are observed, if we evaluate more than ten functions in parallel. We also conclude that at least 5 processors should be available.

The most promising possibility to exploit a parallel system architecture occurs, when gradients cannot be calculated analytically, but have to be approximated numerically, for example by forward differences, two-sided differences, or even higher order methods. Then we need at least n additional function calls, where n is the number of optimization variables, or a suitable multiple of n.

3.3 Testing NLPQLP under Random Noise

For our numerical tests, we use the two-sided difference formula (13). To test the stability of the SQP code, we add randomly generated noise to all function values. Non-monotone line search is applied with a queue length of $p = 40$ in error situations, and the line search calculation by Algorithm 2.1 is used. The BFGS quasi-Newton updates are restarted with ρI if a descent direction cannot be computed, with $\rho = 10^4$.

To compare the different stabilization approaches, we apply three different scenarios how to handle error situations, which would otherwise lead to early termination,

- monotone line search, no restarts ($\rho = 0$, $p = 0$),
- non-monotone line search, no restarts ($\rho = 0$, $p = 40$),
- non-monotone line search and restarts ($\rho = 10^4$, $p = 40$).

The corresponding results shown in Table 2, are evaluated for increasing random perturbations (ϵ_{err}). More precisely, if ν denotes a uniformly distributed random number between 0 and 1, we replace $f(x_k)$ by $f(x_k)(1+\epsilon_{err}(2\nu-1))$ at each iterate x_k. In the same way, restriction functions are perturbed. The tolerance for approximating gradients, η_m, is set to the machine accuracy in case of $\epsilon_{err} = 0$, and to the random noise level otherwise.

The numerical results are surprising and depend heavily on the new non-monotone line search strategy and the additional stabilization procedures. We are able to solve about 98 % of the test examples in case of extremely noisy function values with at most one correct

	$\rho = 0$, $p = 0$			$\rho = 0$, $p = 40$			$\rho = 10^4$, $p = 40$		
ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{succ}	n_{func}	n_{grad}	n_{succ}	n_{func}	n_{grad}
0	303	29	19	304	30	20	306	30	20
10^{-12}	303	29	19	304	29	19	306	31	20
10^{-10}	302	31	20	304	32	20	305	33	20
10^{-8}	294	35	20	303	39	20	306	41	22
10^{-6}	267	38	19	302	58	23	303	57	23
10^{-4}	257	39	17	298	66	22	300	77	24
10^{-2}	198	38	11	283	78	19	297	87	20

Table 2: Test Results for 306 Academic Test Problems

digit in partial derivative values. However, the stabilization process is costly, especially the line search. The more test problems are successfully solved, the more function evaluations are needed.

3.4 Testing NLPQLP on Problems with Slow Convergence

In some situations, the convergence of an SQP method becomes quite slow, e.g., in case of badly scaled variables or functions, inaccurate derivatives, or inaccurate solutions of the quadratic program (4). In these situations, errors in the search direction or the partial derivatives influence the update procedure (11) and the quasi-Newton matrices C_k are getting more and more inaccurate.

Scaled restarts as described in Section 3 are recommended, if convergence turns out to become extremely slow, especially caused by inaccurate partial derivatives. To illustrate the situation, we consider a few test runs where the examples are generated by discretizing a two-dimensional elliptic partial differential equation by the five-star formula, see Maurer and Mittelmann [32, 33]. The original formulation is that of an optimal control problem, and state and control variables are both discretized. The test problems possess a different numerical structure than those used in the previous section, i.e., a large number variables and weakly nonlinear, sparse equality constraints, and are easily scalable to larger sizes. First partial derivatives are available in analytical form.

From a total set of 13 original test cases, we select five problems which could not be solved by NLPQLP as efficiently as expected with standard solution tolerances, especially if we add some noise. Depending on the grid size, in our case 20 in each direction, we get problems with $n = 722$ or $n = 798$ variables, respectively, and $m_e = 361$ or $m_e = 437$ nonlinear equality constraints. There are no nonlinear inequality constraints. Table 3 contains a summary including the best known objective function values subject to an optimal solution vector $x^\star$. The maximum number of iterations is limited by 500, and all other tolerances are thee same as before.

problem	n	m_e	$f(x^\star)$
EX1	722	361	$0.45903 \cdot 10^{-1}$
EX2	722	361	$0.40390 \cdot 10^{-1}$
EX3	722	361	0.11009
EX4	798	437	$0.75833 \cdot 10^{-1}$
EX5	798	437	$0.51376 \cdot 10^{-1}$

Table 3: Elliptic Control Problems

Note that the code NLPQLP is unable to take sparsity into account. With an SQP-IPM solver being able to handle sparsity, it is possible to solve the same test problems successfully with a fine grid leading to 5.000.000 variables and 2.500.000 constraints, see Sachsenberg [41].

Table 4 shows the number of iterations or gradient evaluations, respectively, for three sets of test runs and different noise levels. Since we have analytical derivatives, we add the perturbations to function as we did for the first set of test runs, and to all partial derivative values. For uniformly distributed random numbers ν between 0 and 1, we add $1 + \epsilon_{err}(2\nu - 1)$ to $f(x_k)$ and $\partial f(x_k)/\partial x_i$ for each iterate x_k and $i = 1, \ldots, n$. In the same way, restriction functions and their derivatives are perturbed. We consider three different scenarios defined by $\epsilon_{err} = 0$, $\epsilon_{err} = 10^{-6}$, and $\epsilon_{err} = 10^{-4}$. The obtained objective function values coincide with those of Table 3 to at least seven digits with one exception. For one test run without scaling, but noisy function and gradient values, the upper bound of 500 iterations is reached. But also in this case, four digits of the optimal function value are correct. The queue length for non-monotone line search is set to 40 and the parameter for internal restarts in error cases is set to $\rho = 10^4$.

We apply different strategies for restarting the BFGS update, i.e., C_k,

- no scaling, i.e., $C_0 = I$,
- initial scaling by the Oren-Luenberger procedure, i.e., update started at $\gamma_1 I$,
- C_k is reset if $\gamma_k \leq \sqrt{\epsilon_t}$, where ϵ_t is the termination accuracy,
- C_k is reset every 7th step,
- C_k is reset every 15th step.

For test runs without or only initial scaling, there are only marginal difference between unperturbed and perturbed function and derivative values. On the other hand, adaptive and periodic scaling reduces the number of iterations significantly. The best results are obtained for periodic scaling after 7 iterations. However, the number of test problems is too small and their mathematical structure is too special to retrieve general conclusions.

noise	*scaling*	EX1	EX2	EX3	EX4	EX5
	none	64	109	88	113	212
	initial	64	108	75	108	202
$\epsilon_{err} = 0$	*adaptive*	14	13	14	16	26
	7-step	14	13	14	23	29
	15-step	20	20	21	25	38
	none	64	109	88	110	385
	initial	64	108	83	115	313
$\epsilon_{err} = 10^{-6}$	*adaptive*	14	17	18	60	35
	7-step	17	13	15	45	31
	15-step	20	20	29	26	39
	none	74	111	104	178	500
	initial	63	116	102	171	397
$\epsilon_{err} = 10^{-4}$	*adaptive*	30	81	43	106	42
	7-step	21	32	52	53	34
	15-step	48	30	29	25	57

Table 4: Elliptic Control Problems, Number of Iterations

4 Program Documentation

NLPQLP 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 [18] going back to Powell [39], see also Schittkowski [55] for more details about implementation and usage. Model functions and gradients must be provided by reverse communication. The user has to evaluate function and gradient values in the same program which executes NLPQLP, 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 called X.

2. Compute objective and all constraint function values, store them in F(1) and the first column of G, respectively.

3. Compute gradients of objective function and all constraints, and store them in DF and DG, respectively. The J-th row of DG contains the gradient of the J-th constraint, J=1,...,M.

4. Set IFAIL=0 and execute NLPQLP.

5. If NLPQLP returns with IFAIL=-1, compute objective and constraint function values for all variables found in the first NP columns of X, store them in F (first NP

positions) and G (first NP columns), and call NLPQLP again. If it turns out that F or G cannot be evaluated at X, call NLPQLP with IFAIL=-10. The actual step size will become reduced by the factor 0.5 and NLQLP returns with IFAIL=-1.

6. If NLPQLP terminates with IFAIL=-2, compute gradient values with respect to the variables stored in the first column of X, and store them in DF and DG. Only derivatives for active constraints, ACT(J)=.TRUE., need to be computed. Then call NLPQLP again.

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

If analytical derivatives are not available, simultaneous function calls can be used for gradient approximations, for example by forward differences (2N>NP), two-sided differences (4N>NP≥2N), or even higher order formulae (NP≥4N).

Do never start NLPQLP with IFAIL=-10. Stepsize reduction will not be possible and NLPQLP runs into an infinite loop. If ME > 0 or if the actual iterate is not strictly feasible, successive stepsize reduction might not work. Moreover, setting IFAIL=-10 is only allowed in case of NP=1.

Usage:

```
     CALL   NLPQLP(         NP,        M,        ME,    MMAX,        N,
  /                       NMAX,     MNN2,         X,       F,        G,
  /                         DF,       DG,         U,      XL,       XU,
  /                          C,        D,       ACC,   ACCQP,   STPMIN,
  /                     MAXFUN,    MAXIT,    MAXNM,      RHO,   IPRINT,
  /                       MODE,     IOUT,     IFAIL,      WA,      LWA,
  /                        KWA,     LKWA,       ACT,    LACT,      LQL,
  /                     QPSLVE                                         )
```

Definition of the parameters:

NP:	Number of processors, i.e., number of parallel function evaluations to be provided by the calling program. If less than 5 parallel processors are available, it is recommended to apply the serial version by setting NP=1, i.e., NP must be greater than 4 and less than 50.

M :	Total number of constraints.
ME :	Number of equality constraints.
MMAX :	Row dimension of array DG containing Jacobian of constraints. MMAX must be at least one and greater or equal to M.
N :	Number of optimization variables.
NMAX :	Row dimension of C. NMAX must be at least two and greater than N.
MNN2 :	Must be equal to M+N+N+2 when calling NLPQLP.
X(NMAX,NP) :	Initially, i.e., when calling NLPQLP with IFAIL=0, the first column of X has to contain starting values. On return, X is replaced by the current iterate. In the driving program, the row dimension of X has to be equal to NMAX. X is internally used to store NP different arguments for which function values are be provided by the calling program, or the final iterate in the first column. Note that in an error situation, the best intermediate iterate is returned.
F(NP) :	On return, F(1) contains the final objective function value. F is used also to store NP different objective function values to be computed from NP sets of arguments stored in X.
G(MMAX,NP) :	On return, the first column of G contains the constraint function values at the final iterate X. In the driving program, the row dimension of G has to be equal to MMAX. G is used internally to store NP different sets of constraint function values to be computed from NP sets of arguments stored in X.
DF(NMAX) :	DF contains the current gradient of the objective function subject to the variable values found in the first column of X. In case of numerical differentiation and a distributed system (NP>1), it is recommended to apply parallel evaluations of F to compute DF.
DG(MMAX,NMAX) :	Input matrix containing the Jacobian matrix of the constraints computed subject to the first column of X, first for the ME equality constraints, then for M-ME inequality constraints (row by row). In the driving program, the row dimension of DG must be equal to MMAX. It is sufficient to determine the gradients subject to the active constraints specified by ACT, see below. In case of numerical differentiation and a distributed system (NP>1), it is recommended to apply parallel evaluations of G to compute DG.

U(MNN2) :	U contains the multipliers with respect to the actual iterate stored in the first column of X. The first M locations contain the multipliers of the M nonlinear constraints, the subsequent N locations the multipliers of the lower bounds, and the final N locations the multipliers of the upper bounds. At an optimal solution, all multipliers with respect to inequality constraints should be nonnegative.
XL(N),XU(N) :	On input, the one-dimensional arrays XL and XU must contain the lower and upper bounds of the variables, respectively.
C(NMAX,NMAX) :	On return, C contains the last computed approximation of the Hessian matrix of the Lagrangian function. C is stored in form of an Cholesky decomposition, is LQL is set to false, see below. In this case, C contains the lower triangular factor of an LDL factorization of the final quasi-Newton matrix (without diagonal elements, which are always one). In the driving program, the row dimension of C has to be equal to NMAX.
D(NMAX) :	The elements of the diagonal matrix of the LDL decomposition of the quasi-Newton matrix are stored in the one-dimensional array D, if LQL is false.
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 needed for 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.
STPMIN :	Minimum steplength in case of NP>1. Recommended is any value in the order of the accuracy by which functions are computed. The value is needed to compute a steplength reduction factor by STPMIN**(1/(NP-1)). STPMIN should not fall below machine accuracy.

MAXFUN :	The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN is only needed in case of NP=1, and must not be greater than 50.
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 performing 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. A value greater than one is recommended. (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
MODE :	The parameter specifies the desired version of NLPQLP. 0 - Normal execution (reverse communication!). 1 - Initial guess for multipliers in U and Hessian of the Lagrangian function in C and D provided. In case of LQL=.TRUE., D is ignored. Otherwise, the lower part of C has to contain the lower triangular factor of an LDL decomposition and D the diagonal part.

2 - Initial scaling (Oren-Luenberger) after first step, BFGS updates started from multiple of identity matrix.

3 - Scaled resets of BFGS matrix, if scaling parameter is less than square root of ACC.

$>$ 3 - Initial and repeated resets of BFGS matrix every MODE steps.

IOUT : Positive 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 contains one the following values:

-2 - Compute new gradient values.

-1 - Compute new function values.

0 - Optimality conditions satisfied.

1 - Stop after MAXIT iterations.

2 - Uphill search direction.

3 - Underflow when computing new BFGS-update matrix.

4 - Line search exceeded MAXFUN iterations.

5 - Length of a working array too short.

6 - False dimensions, M$>$MMAX, N$\geq$NMAX, or MNN2$\neq$M+N+N+2.

7 - Search direction close to zero at infeasible iterate.

8 - Starting point violates lower or upper bound.

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

10 - Inconsistency in QP, division by zero.

11 - More than MAXFUN successive non-evaluable function calls.

$>$100 - Error message of QP solver.

If it turns out that F and G cannot be evaluated at X, call NLPQLP with IFAIL=-10, see above. The actual step will become reduced by the factor 0.5 and NLQLP returns immediately with IFAIL=-1 to request new function values.

NOTE: Might not work in case of ME=0 or non-strictly feasible iterates!

WA(LWA) :	WA is a double precision working array of length LWA. On return, the first N positions contain the best feasible iterate obtained, WA(N+1) the corresponding objective function value, and the subsequent M positions the constraint values. If no intermediate feasible solution exists, WA(N+1) contains a large value, e.g., 1.0D+72.
LWA :	Length of WA, has to be at least at least 23*N+4*M+3*MMAX+NP*(N+M+1)+150. NOTE: The standard QP-solver coming together with NLPQLP (QL) needs additional memory for 3*NMAX*NMAX/2+10*NMAX+MMAX+M+1 double precision numbers.
KWA(LKWA) :	KWA is an integer working array of length LKWA. On return, the first 5 positions contain the following information: KWA(1) - Number of function evaluations. KWA(2) - Number of gradient evaluations. KWA(3) - Iteration count. KWA(4) - Number of QP's solved. KWA(5) - Flag for better feasible, but non-stationary iterate (=1) or not (=0), see below.
LKWA :	Length of KWA, has to be at least 25. NOTE: The standard QP-solver coming together with NLPQLP (QL) needs additional memory for N integer numbers.
ACT(LACT) :	The logical array indicates constraints, which NLPQLP considers to be active at the last computed iterate, i.e., G(J,1) is active, if and only if ACT(J) is true for J=1,...,M.
LACT :	Length of ACT, has to be at least 2*M+10.
LQL :	If LQL is set to true in the calling program, the quadratic programming problem is solved proceeding from a full positive definite quasi-Newton matrix. Otherwise, a Cholesky decomposition (LDL) is performed and updated internally, so that matrix C always consists of the lower triangular factor and D of the diagonal.

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 [55].

In case of IFAIL=-1, the user has to compute objective function and all constraint values subject to the variable values found in the first NP columns of X, and to store them in F and G. Then NLPQLP is called again. In case of IFAIL=-2, one has to compute gradient values subject to the variables stored in the first column of X, and to store them in DF and DG. Only derivatives for active constraints ACTIVE(J)=.TRUE. need to be computed. Then NLPQLP is called again.

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 nonlinear and non-convex constraints are feasible 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 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 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.

NLPQLP returns the best iterate obtained. In case of successful termination (IFAIL=0), this is always the last one. But it might be possible that in an exceptional situation, an intermediate iterate is feasible with a better objective function value than that one of the final iterate, but the KKT optimality conditions are not satisfied. In this case, the better feasible solution is stored at the first n positions of the double precision working array and the corresponding objective function value at position $n + 1$. Moreover, positions $n + 2$ to $n + 1 + m$ contain the constraint values. Note that feasibility is tested by sum of constrained violations tested against ACC.

On successful return with IFAIL=0, KWA(5) is set to zero. If, however, a better feasible objective function value has been found during the first five iterations, then KWA(5) is set to 1, the BFGS-update matrix is set to ρI with $\rho > 0$, where I denotes the identity matrix. The corresponding formal argument of NLPQLP is called RHO. Moreover, the multiplier approximation vector U is set to 0. Thus, an immediate restart under control of the user is possible with MODE=1. Some information is printed on the standard IO channel in case of IPRINT>0. For compatibility reasons with previous versions, RHO replaces TOLNM.

The QP solver is defined in form of an external subroutine to allow a replacement in case of exploiting special sparsity patterns. A typical example is the usage of NLPQLP for solving least squares problems, where artificially introduced equality constraints lead to a Jacobian which consist partially of the identity matrix, see Schittkowski [52, 53].

The internal scaling and restart option is borrowed from limited-memory quasi-Newton methods, see for example Liu and Nocedal [30]. If requested by the user, the quasi-Newton matrix is replaced by a scalar multiple of the identity matrix just before updating. Either an initial scaling or a reset of the whole matrix and computation of a new scaling parameter is performed depending on the input parameter MODE. A scaled restart is recommended, if, e.g., the convergence turns out to become extremely slow.

Note that in case of an infeasible domain of the quadratic programming subproblem, an artificial variable is added to enforce feasibility. Thus, the dimension of some arrays has to be greater than N.

5 Examples

NLPQLP comes with a couple of demo programs by which the following situations are to be illustrated:

file name	comments
NLP_DEMOA.for	easy-to-use, analytical derivatives
NLP_DEMOA.f90	same, but Fortran 90
NLP_DEMOB.for	numerical differentiation
NLP_DEMOC.f90	numerical differentiation and distributed function calls, Fortran 90
NLP_DEMOD.for	warm and cold restarts
NLP_DEMOE.for	simultaneous function and gradient evaluation
NLP_DEMOF.for	active set strategy, numerical derivatives,
NLP_DEMOG.for	active set strategy, numerical derivatives, distributed function calls
NLP_DEMOH.for	initial BFGS matrix and multipliers provided
NLP_DEMOI.for	non-evaluable function calls
NLP_DEMOJ.for	noisy derivatives and n distributed function calls
NLP_DEMOK.for	easy-to-use version with numerical gradients and non-monotone line search
NLP_DEMOL.for	least-squares objective function, analytical derivatives

The first eight examples are different implementations of Rosenbrock's post office problem, i.e., test problem TP37 of Hock and Schittkowski [26],

$$x_1, x_2, x_3 \in I\!R : \begin{array}{l} \min -x_1 x_2 x_3 \\ x_1 + 2x_2 + 2x_3 \geq 0 \\ 72 - x_1 - 2x_2 - 2x_3 \geq 0 \\ 0 \leq x_1 \leq 42 \\ 0 \leq x_2 \leq 42 \\ 0 \leq x_3 \leq 42 \end{array} \qquad (15)$$

A Fortran source code of NLP_DEMOA.for is listed below. The function and derivative blocks of the main program can be replaced by subroutine calls.

```
      IMPLICIT         NONE
      INTEGER          NMAX, MMAX, MNN2X, LWA, LKWA, LACTIV
      PARAMETER (      NMAX   = 4,
     /                 MMAX   = 2,
     /                 MNN2X  = MMAX + NMAX + NMAX + 2,
     /                 LWA    = 1.5*NMAX*NMAX + 34*NMAX + 10*MMAX + 150,
     /                 LKWA   = NMAX + 25,
     /                 LACTIV = 2*MMAX + 10)
      INTEGER          KWA(LKWA), N, ME, M, NP, MNN2, MAXIT, MAXFUN,
     /                 IPRINT, MAXNM, IOUT, MODE, IFAIL, I, J
      DOUBLE PRECISION X(NMAX), F, G(MMAX), DF(NMAX), DG(MMAX,NMAX),
     /                 U(MNN2X), XL(NMAX), XU(NMAX), C(NMAX,NMAX),
     /                 D(NMAX), WA(LWA), ACC, ACCQP, STPMIN, RHO
      LOGICAL          ACTIVE(LACTIV), LQL
      EXTERNAL         QL
C
C   Set some constants and initial values
C
      IOUT   = 6
      ACC    = 1.0D-11
      ACCQP  = 1.0D-12
      STPMIN = 0.0
      MAXIT  = 100
      MAXFUN = 10
      MAXNM  = 0
      RHO    = 0.0D0
      LQL    = .TRUE.
      IPRINT = 2
      N      = 3
      M      = 2
      ME     = 0
      MNN2   = M + N + N + 2
      MODE   = 0
      IFAIL  = 0
      NP     = 1
      DO I=1,N
         X(I)  = 10.0D0
         XL(I) = 0.0D0
         XU(I) = 42.0D0
      ENDDO
    1 CONTINUE
C============================================================
C   This block computes all function values.
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============================================================
      IF (IFAIL.EQ.-1) GOTO 4
    2 CONTINUE
C============================================================
C   This block computes all derivative values.
C
      DF(1)   = -X(2)*X(3)
      DF(2)   = -X(1)*X(3)
      DF(3)   = -X(1)*X(2)
      DG(1,1) =  1.0D0
      DG(1,2) =  2.0D0
      DG(1,3) =  2.0D0
      DG(2,1) = -1.0D0
      DG(2,2) = -2.0D0
      DG(2,3) = -2.0D0
C
C============================================================
    4 CONTINUE
      CALL NLPQLP (    NP,      M,     ME,   MMAX,      N,
     /               NMAX,   MNN2,      X,      F,      G,
     /                 DF,     DG,      U,     XL,     XU,
     /                  C,      D,    ACC,  ACCQP, STPMIN,
     /             MAXFUN,  MAXIT,  MAXNM,    RHO, IPRINT,
     /               MODE,   IOUT,  IFAIL,     WA,    LWA,
     /                KWA,   LKWA, ACTIVE, LACTIV,    LQL,
     /                 QL)
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
C
      STOP
      END
```

The following output should appear on screen:

```
--------------------------------------------------------------------
     Start of the Sequential Quadratic Programming Algorithm
                 NLPQLP, Version 4.0 (July 2012)
--------------------------------------------------------------------

Parameters:
   N      =       3
   M      =       2
   ME     =       0
   MODE   =       0
   ACC    =   0.1000D-10
   ACCQP  =   0.1000D-11
   STPMIN =   0.1000D-14
   RHO    =   0.0000D+00
   MAXFUN =      10
   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 -0.10000000D+04  0.00D+00    2  0  0.00D+00  0.00D+00  0.44D+04
   2 -0.23625000D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.11D+04
   3 -0.32507304D+04  0.36D-14    1  1  0.10D+01  0.00D+00  0.69D+03
   4 -0.34557650D+04  0.71D-14    1  2  0.54D+00  0.00D+00  0.55D+00
   5 -0.34559928D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.15D-01
   6 -0.34560000D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.33D-06
   7 -0.34560000D+04  0.00D+00    1  1  0.10D+01  0.00D+00  0.24D-12

   Objective function value:      F(X)  = -0.34560000D+04
   Solution values:               X     =
      0.24000000D+02  0.12000000D+02  0.12000000D+02
   Distances from lower bounds:   X-XL  =
      0.24000000D+02  0.12000000D+02  0.12000000D+02
   Distances from upper bounds:   XU-X  =
      0.18000000D+02  0.30000000D+02  0.30000000D+02
   Multipliers for lower bounds:  U     =
      0.00000000D+00  0.00000000D+00  0.00000000D+00
   Multipliers for upper bounds:  U     =
      0.00000000D+00  0.00000000D+00  0.00000000D+00
   Constraint values:             G(X)  =
      0.72000000D+02  0.00000000D+00
   Multipliers for constraints:   U     =
      0.00000000D+00  0.14400000D+03
   Number of function calls:      NFUNC =      8
   Number of gradient calls:      NGRAD =      7
   Number of calls of QP solver:  NQL   =      7
```

To illustrate non-evaluable function calls, we consider the following example, where a logarithm can only be evaluated at positive values, see NLP_DEMOI.

$$
x_1, x_2 \in \mathbb{R}: \begin{array}{l} \min 100(x_2 - x_1^2)^2 + (x_1 - 1)^2 \\ x_1 - \log(2 - x_1^2 - x_2^2) \geq 0 \\ 2 - x_1^2 - x_2^2 \geq 0 \\ -2 \leq x_1 \leq 2 \\ -2 \leq x_2 \leq 2 \end{array} \tag{16}
$$

The code executing NLPQLP is listed as follows.

```
      IMPLICIT         NONE
      INTEGER          NMAX, MMAX, MNN2X, LWA, LKWA, LACTIV
      PARAMETER (      NMAX   = 3,
     /                 MMAX   = 2,
     /                 MNN2X  = MMAX + NMAX + NMAX + 2,
     /                 LWA    = 1.5*NMAX*NMAX + 34*NMAX + 10*MMAX + 150,
     /                 LKWA   = NMAX + 25,
     /                 LACTIV = 2*MMAX + 10)
      INTEGER          KWA(LKWA), N, ME, M, MNN2, MAXIT, MAXFUN,
     /                 IPRINT, MAXNM, IOUT, MODE, IFAIL, I
      DOUBLE PRECISION X(NMAX), F(1), G(MMAX), DF(NMAX),
     /                 DG(MMAX,NMAX), U(MNN2X), XL(NMAX), XU(NMAX),
     /                 C(NMAX,NMAX), D(NMAX), WA(LWA), ACC, ACCQP,
     /                 STPMIN, RHO, A, DA1, DA2, EPS
      LOGICAL          ACTIVE(LACTIV), LQL
      EXTERNAL         QL
C
C   Set some constants and initial values
C
      EPS    = 1.0D-7
      IOUT   = 6
      ACC    = 1.0D-14
      ACCQP  = 1.0D-15
      STPMIN = 1.0D-10
      RHO    = 1.0D3
      MAXIT  = 500
      MAXFUN = 20
      MAXNM  = 20
      IPRINT = 2
      MODE   = 0
      N      = 2
      ME     = 0
      M      = 2
      MNN2   = M + N + N + 2
      IFAIL  = 0
      LQL    = .TRUE.
      DO I=1,N
         X(I)  = 0.0D0
         XL(I) = -2.0D0
         XU(I) = 2.0D0
      ENDDO
C
C============================================================
C   This is the main block to compute all function and gradient
C   values.
C
    1 CONTINUE
      A  = 2.0D0 - X(1)**2 - X(2)**2
      IF (A.LT.EPS) THEN
         IFAIL= -10
         GOTO 2
      ENDIF
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-1)) THEN
         F    = 100.0D0*(X(2) - X(1)**2)**2 + (X(1) - 1.0D0)**2
         G(1) = X(1) - DLOG(A)
         G(2) = A
      ENDIF
C
      IF ((IFAIL.EQ.0).OR.(IFAIL.EQ.-2)) THEN
         DF(1)   = -400.0D0*X(1)*(X(2) - X(1)**2)
     /                         + 2.0D0*(X(1) - 1.0D0)
         DF(2)   = 200.0D0*(X(2) - X(1)**2)
         DA1     = -2.0D0*X(1)
         DA2     = -2.0D0*X(2)
         DG(1,1) = 1.0D0 - DA1/A
         DG(1,2) = -DA2/A
         DG(2,1) = DA1
         DG(2,2) = DA2
      ENDIF
C
C============================================================
C
    2 CONTINUE
      CALL NLPQLP (     1,      M,     ME,   MMAX,      N,
     /              NMAX,   MNN2,      X,      F,      G,
     /                DF,     DG,      U,     XL,     XU,
     /                 C,      D,    ACC,  ACCQP, STPMIN,
     /            MAXFUN,  MAXIT,  MAXNM,    RHO, IPRINT,
     /              MODE,   IOUT,  IFAIL,     WA,    LWA,
     /               KWA,   LKWA, ACTIVE, LACTIV,    LQL,
     /                QL)
      IF (IFAIL.LT.0) GOTO 1
C
      STOP
      END
```

Convergence speed is no longer superlinear, since the optimal solution is very close to a singular point.

```
    --------------------------------------------------------------------
        Start of the Sequential Quadratic Programming Algorithm
                   NLPQLP, Version 4.0 (July 2012)
    --------------------------------------------------------------------

    Parameters:
       N      =       2
       M      =       2
       ME     =       0
       MODE   =       0
       ACC    =   0.1000D-13
       ACCQP  =   0.1000D-13
       STPMIN =   0.1000D-09
       RHO    =   0.1000D+04
       MAXFUN =      20
       MAXNM  =      20
       MAXIT  =     500
       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  0.10000000D+01  0.69D+00    2  0  0.00D+00  0.00D+00  0.40D+01
      2  0.80000000D+00  0.47D+00    1  2  0.10D+00  0.00D+00  0.71D+01
      3  0.69132970D+00  0.23D+00    1  2  0.10D+00  0.00D+00  0.26D+01
      4  0.33893007D+00  0.15D+00    1  2  0.33D+00  0.00D+00  0.14D+00
                           .....
     21  0.95514272D-12  0.00D+00    1  1  0.10D+01  0.00D+00  0.32D-11
     22  0.13510433D-12  0.00D+00    1  1  0.10D+01  0.00D+00  0.40D-12
     23  0.31530459D-13  0.00D+00    1  1  0.10D+01  0.00D+00  0.82D-13
     24  0.86289909D-14  0.00D+00    1  1  0.10D+01  0.00D+00  0.26D-13
     25  0.19608380D-14  0.00D+00    1  1  0.10D+01  0.00D+00  0.59D-14


       Objective function value:      F(X)  =  0.19608380D-14
       Solution values:               X     =
          0.99999996D+00  0.99999991D+00
       Distances from lower bounds:   X-XL  =
          0.30000000D+01  0.29999999D+01
       Distances from upper bounds:   XU-X  =
          0.10000000D+01  0.10000001D+01
       Multipliers for lower bounds:  U     =
          0.00000000D+00  0.00000000D+00
       Multipliers for upper bounds:  U     =
          0.00000000D+00  0.00000000D+00
       Constraint values:             G(X)  =
          0.16155229D+02  0.26191935D-06
       Multipliers for constraints:   U     =
          0.00000000D+00  0.71413585D-08
       Number of function calls:      NFUNC =     31
       Number of gradient calls:      NGRAD =     25
       Number of calls of QP solver:  NQL   =     25
```

6 Conclusions

We present a modification of an SQP algorithm designed for execution under a parallel computing environment (SPMD) and where a non-monotone line search is applied in error situations. Numerical results indicate stability and robustness for a set of 306 standard test problems. It is shown that not more than 7 parallel function evaluation per iterations are required for performing a sufficiently accurate line search. Significant performance improvement is achieved by the non-monotone line search especially in case of noisy function values and numerical differentiation, and by restarts in a severe error situation. With the new version of NLPQLP, we are able to solve most examples of our standard set of 306 test problems subject to an termination accuracy 10^{-7} in case of extremely noisy function values with relative accuracy of 1 % and numerical differentiation. In the worst case, at most one digit of a partial derivative value is correct.

References

[1] Armijo L. (1966): *Minimization of functions having Lipschitz continuous first partial derivatives,* Pacific Journal of Mathematics, Vol. 16, 1–3

[2] Barzilai J., Borwein J.M. (1988): *Two-point stepsize gradient methods,* IMA Journal of Numerical Analysis, Vol. 8, 141–148

[3] Boderke P., Schittkowski K., Wolf M., Merkle H.P. (2000): *Modeling of diffusion and concurrent metabolism in cutaneous tissue,* Journal on Theoretical Biology, Vol. 204, No. 3, 393-407

[4] Boggs P.T., Tolle J.W. (1995): *Sequential quadratic programming*, Acta Numerica, Vol. 4, 1 - 51

[5] Bonnans J.F., Panier E., Tits A., Zhou J.L. (1992): *Avoiding the Maratos effect by means of a nonmonotone line search, II: Inequality constrained problems – feasible iterates,* SIAM Journal on Numerical Analysis, Vol. 29, 1187–1202

[6] Birk J., Liepelt M., Schittkowski K., Vogel F. (1999): *Computation of optimal feed rates and operation intervals for tubular reactors,* Journal of Process Control, Vol. 9, 325-336

[7] Blatt M., Schittkowski K. (1998): *Optimal Control of One-Dimensional Partial Differential Equations Applied to Transdermal Diffusion of Substrates*, in: Optimization Techniques and Applications, L. Caccetta, K.L. Teo, P.F. Siew, Y.H. Leung, L.S. Jennings, V. Rehbock eds., School of Mathematics and Statistics, Curtin University of Technology, Perth, Australia, Vol. 1, 81 - 93

[8] Bongartz I., Conn A.R., Gould N., Toint Ph. (1995): *CUTE: Constrained and unconstrained testing environment*, Transactions on Mathematical Software, Vol. 21, No. 1, 123-160

[9] Bünner M.J., Schittkowski K., van de Braak G. (2004): *Optimal design of electronic components by mixed-integer nonlinear programming*, Optimization and Engineering, Vol. 5, 271-294

[10] Dai Y.H., Liao L.Z. (1999): *R-Linear Convergence of the Barzilai and Borwein Gradient Method,* Research Reoport 99-039, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences

[11] Dai Y.H. (2000): *A nonmonotone conjugate gradient algorithm for unconstrained optimization,* Research Report, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences

[12] Dai Y.H. (2002): *On the nonmonotone line search,* Journal of Optimization Theory and Applications, Vol. 112, No. 2, 315–330

[13] Dai Y.H., Schittkowski K. (2008): *A sequential quadratic programming algorithm with non-monotone line search*, Pacific Journal of Optimization, Vol. 4, 335-351

[14] Deng N.Y., Xiao Y., Zhou F.J. (1993): *Nonmonotonic trust-region algorithm,* Journal of Optimization Theory and Applications, Vol. 26, 259–285

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

[16] Frias J.M., Oliveira J.C, Schittkowski K. (2001): *Modelling of maltodextrin DE12 drying process in a convection oven*, Applied Mathematical Modelling, Vol. 24, 449-462

[17] Geist A., Beguelin A., Dongarra J.J., Jiang W., Manchek R., Sunderam V. (1995): *PVM 3.0. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press

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

[19] Grippo L., Lampariello F., Lucidi S. (1986): *A nonmonotone line search technique for Newtons's method,* SIAM Journal on Numerical Analysis, Vol. 23, 707–716

[20] Grippo L., Lampariello F., Lucidi S. (1989): *A truncated Newton method with nonmonotone line search for unconstrained optimization,* Journal of Optimization Theory and Applications, Vol. 60, 401–419

[21] Grippo L., Lampariello F., Lucidi S. (1991): *A class of nonmonotone stabilization methods in unconstrained optimization,* Numerische Mathematik, Vol. 59, 779–805

[22] Han S.-P. (1976): *Superlinearly convergent variable metric algorithms for general nonlinear programming problems* Mathematical Programming, Vol. 11, 263-282

[23] Han S.-P. (1977): *A globally convergent method for nonlinear programming* Journal of Optimization Theory and Applications, Vol. 22, 297–309

[24] Hartwanger C., Schittkowski K., Wolf H. (2000): *Computer aided optimal design of horn radiators for satellite communication*, Engineering Optimization, Vol. 33, 221-244

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

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

[27] Ke X., Han J. (1995): *A nonmonotone trust region algorithm for equality constrained optimization,* Science in China, Vol. 38A, 683–695

[28] Ke X., Liu G., Xu D. (1996): *A nonmonotone trust-region algorithm for unconstrained optimization,* Chinese Science Bulletin, Vol. 41, 197–201

[29] Kneppe G., Krammer J., Winkler E. (1987): *Structural optimization of large scale problems using MBB-LAGRANGE*, Report MBB-S-PUB-305, Messerschmitt-Bölkow-Blohm, Munich

[30] Liu D.C., Nocedal J. (1989): *On the limited memory BFGS method for large scale optimization*, Mathematical Programming, Vol. 45, 503–528

[31] Lucidi S., Rochetich F, Roma M. (1998): *Curvilinear stabilization techniques for truncated Newton methods in large-scale unconstrained optimization,* SIAM Journal on Optimization, Vol. 8, 916–939

[32] Maurer H., Mittelmann H.D. (2000): *Optimization techniques for solving elliptic control problems with control and state constraints: Part 1. Boundary control*, Computational Optimization and Applications, Vol. 16, 29–55

[33] Maurer H., Mittelmann H.D. (2001): *Optimization techniques for solving elliptic control problems with control and state constraints. Part 2: Distributed control*, Computational Optimization and Applications, Vol. 18, 141–160

[34] Ortega J.M., Rheinbold W.C. (1970): *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York-San Francisco-London

[35] Panier E., Tits A. (1991): *Avoiding the Maratos effect by means of a nonmonotone line search, I: General constrained problems,* SIAM Journal on Numerical Analysis, Vol. 28, 1183–1195

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

[37] 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

[38] 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

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

[40] Raydan M. (1997): *The Barzilai and Borwein gradient method for the large-scale unconstrained minimization problem,* SIAM Journal on Optimization, Vol. 7, 26–33

[41] Sachsenberg, B. (2010): *NLPIP: A Forrtan implementation of an SQP Interior Point algorithm for solving large scale nonlinear optimization problems - user's guide*, Report, Department of Computer Science, University of Bayreuth

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

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

[44] 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

[45] 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

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

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

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

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

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

[51] 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

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

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

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

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

[56] Schittkowski K. (2008): *An active set strategy for solving optimization problems with up to 200,000,000 nonlinear constraints*, Applied Numerical Mathematics, Vol. 59, 2999-3007

[57] Schittkowski K. (2008): *An updated set of 306 test problems for nonlinear programming with validated optimal solutions - user's guide*, Report, Department of Computer Science, University of Bayreuth

[58] Schittkowski K. (2010): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search*, Report, Department of Computer Science, University of Bayreuth

[59] K. Schittkowski (2011): *A robust implementation of a sequential quadratic programming algorithm with successive error restoration*, Optimization Letters, Vol. 5, 283-296

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

[61] 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

[62] Toint P.L. (1996): *An assessment of nonmontone line search techniques for unconstrained optimization,* SIAM Journal on Scientific Computing, Vol. 17, 725–739

[63] Toint P.L. (1997): *A nonmonotone trust-region algorithm for nonlinear optimization subject to convex constraints,* Mathematical Programming, Vol. 77, 69–94

[64] Wolfe P. (1969): *Convergence conditions for ascent methods,* SIAM Review, Vol. 11, 226–235

[65] Zhou J.L., Tits A. (1993): *Nonmonotone line search for minimax problems,* Journal of Optimization Theory and Applications, Vol. 76, 455–476