Sequential Quadratic Programming Methods with Distributed and Non-Monotone Line Search

K. Schittkowski *

Abstract

Sequential quadratic programming (SQP) methods are widely used for solving practical optimization problems, especially in structural mechanics. The general structure of SQP methods is briefly introduced and it is shown how these methods can be adapted to distributed computing. However, SQP methods are sensitive subject to errors in function and gradient evaluations. Typically they break down with an error message reporting that the line search cannot be terminated successfully. In these cases, a new non-monotone line search is activated. In case of noisy function values, a drastic improvement of the performance is achieved compared to the version with monotone line search. Numerical results are presented for a set of more than 300 standard test examples.

Keywords: SQP, sequential quadratic programming, nonlinear programming, non-monotone line search, distributed computing

*Department of Computer Science, University of Bayreuth, D-95440 Bayreuth

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 $I\!R^n$. But besides of this we do not suppose any further mathematical structure of the model functions.

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

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

The code NLPQL of Schittkowski [44] is a Fortran implementation of a sequential quadratic programming (SQP) algorithm. The design of the numerical algorithm is founded on extensive comparative numerical tests of Schittkowski [37, 41, 39], Schittkowski et al. [57], Hock and Schittkowski [24], and on further theoretical investigations published in [38, 40, 42, 43]. The algorithm is extended to solve also nonlinear least squares problems efficiently, see [47] or [52], and to handle problems with very many constraints, cf. [48]. To conduct the numerical tests, a random test problem generator is developed for a major comparative study, see [37]. Two collections with more than 300 academic and real-life test problems are published in Hock and Schittkowski [24] and in Schittkowski [45]. Fortran source codes and a test frame can be downloaded from the home page of the author,

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

The test examples are part of the Cute test problem collection of Bongartz et al. [7]. About 80 test problems based on a Finite Element formulation are collected for the comparative evaluation in Schittkowski et al. [57]. A set of 1,170 least squares test problems solved by an extension of the code NLPQL to retain typical features of a Gauss-Newton algorithm, is described in [52]. Also these problems can be downloaded from the home page of the author together with an interactive user interface called EASY-FIT, see [53].

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

1. mechanical structural optimization, see Schittkowski, Zillober, Zotemantel [57] and Kneppe, Krammer, Winkler [28],

2. data fitting and optimal control of transdermal pharmaceutical systems, see Boderke, Schittkowski, Wolf [3] or Blatt, Schittkowski [6],
3. computation of optimal feed rates for tubular reactors, see Birk, Liepelt, Schittkowski, and Vogel [5],
4. food drying in a convection oven, see Frias, Oliveira, and Schittkowski [22],
5. optimal design of horn radiators for satellite communication, see Hartwanger, Schittkowski, and Wolf [21],
6. receptor-ligand binding studies, see Schittkowski [49],
7. optimal design of surface acoustic wave filters for signal processing, see Bünner, Schittkowski, and van de Braak [8].

The general availability of parallel computers and in particular of distributed computing in networks motivates a careful redesign of NLPQL to allow simultaneous function evaluations. The resulting extensions are implemented and the code is called NLPQLP, see Schittkowski [56]. Another 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 identical to NLPQL. 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 the 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 iterative process is stopped.

The new version 2.0 makes use of non-monotone line search in certain error situations. The idea is to replace the reference value of the line search termination check $\psi_{r_k}(x_k, v_k)$ by

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

where $\psi_r(x, v)$ is a merit function and p a given tolerance. The general idea is not new and for example described in Dai [11], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [16], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [4], Deng et al. [12], Grippo et al. [17, 18], Ke and Han [26], Ke et al. [27], Lucidi et al. [29], Panier and Tits [31], Raydan [36], and Toit [60, 61]. 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 an error situation.

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 [24] and in Schittkowski [45]. They show the sensitivity of the new version with respect to the number of parallel machines and the influence of different 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.

2 Sequential Quadratic Programming

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 [59] in form of a review, or in Spellucci [58] in form of an extensive text book. From the more practical point of view, SQP methods are also introduced in the books of Papalambros, Wilde [32] and Edgar, Himmelblau [13]. Their excellent numerical performance was tested and compared with other methods in Schittkowski [37], and since many years they belong to the most frequently used algorithms to solve practical optimization problems.

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

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

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $I\!R^n$. But besides of this we do not suppose any further mathematical structure of the model functions.

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

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

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

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

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

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

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

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

Although we are able to guarantee that the matrix B_k is positive definite, it is possible that (4) is not solvable due to inconsistent constraints. One possible remedy is to introduce an additional variable $\delta \in I\!R$, leading to a modified quadratic programming problem, see Schittkowski [44] for details.

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

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

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

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

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 [42]. The objective function is *penalized* as soon as an iterate leaves the feasible domain. The corresponding penalty parameters r_j, $j = 1$, ..., m that control the degree of constraint violation, must be chosen in a suitable way to guarantee a descent direction of the merit function, see Schittkowski [42] or Wolfe [62] in a more general setting.

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

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 B_k by the BFGS quasi-Newton formula, cf. Powell [34] or Stoer [59].

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 advisable to waste too many function calls. Moreover, the behavior of the merit function becomes irregular in case on constrained optimization, because of very steep slopes at the border caused by the 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.

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

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

see for example Ortega and Rheinboldt [30]. The constants are from the ranges $0 < \mu < 0.5$, $0 < \beta < 1$, and $0 < \sigma \leq 1$. 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, for example $\mu = 0.0001$ in the present implementation of NLPQL. 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 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 applying first 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 get easily the minimizer of the quadratic interpolation. We accept then the maximum of this value or the Armijo parameter as a new iterate, as shown by the subsequent code fragment implemented in NLPQL.

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[42]. $\bar{\alpha}_i$ 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 NLPQL 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., the corresponding function call is made in the calling program. We have to stop the algorithm, if sufficient descent is not observed after a certain number of iterations, say 10. If the tested stepsizes fall 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 for getting the subsequent iterate with $\alpha_k := \alpha_{i_k}$. One has to be sure that existing convergence results of the SQP algorithm are not violated. For an alternative approach based on pattern search, see Hough, Kolda, and Torczon [25].

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 $\beta = \tau^{1/(l-1)}$.

The paradigm of parallelism is SPMD, i.e., Single Program Multiple Data. 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. [14], 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

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

is satisfied, where $p(k)$ is a predetermined parameter with $p(k) = \min\{k, p\}$, p 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 replace 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 not new and for example described in Dai [11], where a general convergence proof for the unconstrained case is presented. The general idea goes back to Grippo, Lampariello, and Lucidi [16], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [4], Deng et al. [12], Grippo et al. [17, 18], Ke and Han [26], Ke et al. [27], Lucidi et al. [29], Panier and Tits [31], Raydan [36], and Toit [60, 61]. 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.

3 Numerical Test Results

3.1 Test Environment and Test Examples

Our numerical tests use the 306 academic and real-life test problems published in Hock and Schittkowski [24] and in Schittkowski [45]. Part of them are also available in the CUTE library, see Bongartz et. al [7], and their usage is described in Schittkowski [51]. The distribution of the dimension parameter n, the number of variables, is shown in Figure 1. We see, for example, that about 270 of 306 test problems have not more than 10 variables. In a similar way, the distribution of the number of constraints is shown in Figure 2.

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 solutions or from the best numerical data found so far. The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 8.0, under Windows XP, and executed on a Pentium IV processor with 2.8 GHz.

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 two test problem collections. Then we call the

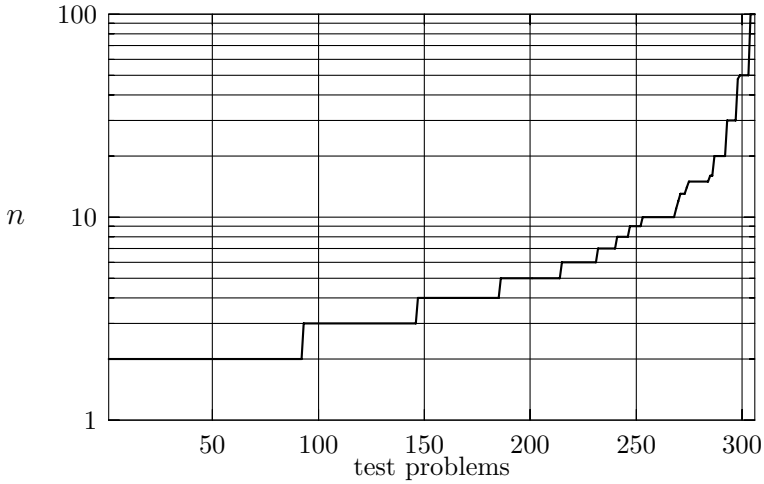

Figure 1: Distribution of Number of Variables.

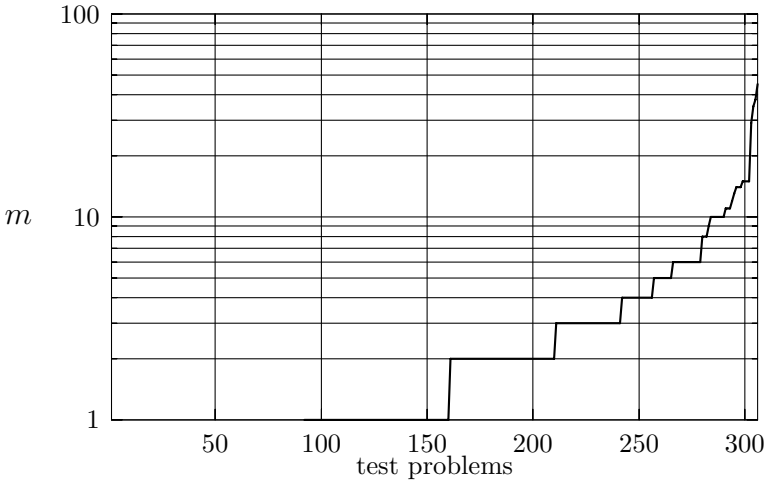

Figure 2: Distribution of Number of Constraints.

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

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

or

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

and

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

where $\| \dots \|_\infty$ denotes the maximum norm and $g_j(x_k)^+ = \max(0, g_j(x_k))$.

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 one. Thus, we call a test run a successful one, if the internal termination conditions are satisfied subject to a reasonably small tolerance (*IFAIL=0*), and if in addition to the above decision,

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

or

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

and

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

For our numerical tests, we use $\epsilon = 0.01$, i.e., we require a final accuracy of one per cent. Gradients are approximated by the following three difference formulae:

1. Forward differences:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{\eta_i} \Big(f(x + \eta_i e_i) - f(x) \Big) \tag{11}$$

2. Two-sided differences:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{2\eta_i} \Big(f(x + \eta_i e_i) - f(x - \eta_i e_i \Big) \tag{12}$$

3. Fourth-order formula:

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{4!\eta_i} \Big(2f(x-2\eta_i e_i) - 16 f(x-\eta_i e_i) + 16 f(x+\eta_i e_i) - 2f(x+2\eta_i e_i) \Big) \tag{13}$$

Here $\eta_i = \eta \max(10^{-5}, |x_i|)$ and e_i is the i-th unit vector, $i = 1, \ldots, n$. 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, see Schittkowski [56]. 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 previous 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 [15] based on numerically stable orthogonal decompositions. NLPQLP is executed with termination accuracy ACC=10^{-8} and a maximum number of iterations *MAXIT=500*.

In the subsequent tables, we use the notation

number of successful test runs	:	n_{succ}
number of false terminations (IFAIL>0)	:	n_{err}
average number of function calls	:	n_{func}
average number of gradient calls	:	n_{grad}
average number of total function calls	:	n_{equ}
total execution time for all test runs in seconds	:	*time*

To get n_{func}, we count each single function call, also in the case of several simulated processors, $l > 1$. However, function evaluations needed for gradient approximations, are not counted. 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.

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, the average number of function calls, and the average number of iterations or gradient evaluations, for an increasing number of simulated parallel calls of model functions denoted by l. The fourth-order formula (13) is used for gradient approximations and non-monotone line search is applied with a queue size of $p = 30$. The calculation time is about 4 *sec* for a series of 306 test runs.

$l = 1$ corresponds to the sequential case, when Algorithm 2.1 is applied for the line search, consisting of a quadratic interpolation combined with an Armijo-type bisection strategy. Since we need at least one function evaluation for the subsequent iterate, we observe that the average number of additional function evaluations needed for the line search, is about two.

l	n_{succ}	n_{func}	n_{grad}
1	306	40	22
3	242	416	130
4	276	423	104
5	293	330	66
6	302	244	39
7	303	241	33
8	299	272	32
9	302	242	26
10	303	277	26
15	302	329	22
20	302	463	23
50	303	1,054	21

Table 1: Performance Results for Parallel Line Search

In all other cases, $l > 1$ simultaneous function evaluations are made according to Algorithm 2.2. Thus, the total number of function calls n_{func} is quite big in Table 1. If, however, the number of parallel machines is sufficiently large in a practical situation, we need only one simultaneous function evaluation in each step of the SQP algorithm. To get a reliable and robust line search, we need at least 5 parallel processors. No significant improvements are observed, if we have more than 10 parallel function evaluations.

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 Gradient Approximations by Difference Formulae under Random Noise

For our numerical tests, we use the three different difference formulae mentioned before, see (11), (12), and (13). To test the stability of these formulae, we add some randomly generated noise to each function value. Non-monotone line search is applied with a queue size of $p = 30$, and the serial line search calculation by Algorithm 2.1 is required.

Tables 2 to 4 show the corresponding results for the different procedures under consideration, and for increasing random perturbations (ϵ_{err}). We report the number of successful runs (n_{succ}) only, since the average number of iterations is more or less the same in all cases. 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.

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	306	37	22	363
10^{-12}	302	56	25	392
10^{-10}	305	67	26	438
10^{-8}	301	86	26	471
10^{-6}	297	142	36	870
10^{-4}	287	147	29	469
10^{-2}	255	191	27	432

Table 2: Test Results for Forward Differences

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	306	40	22	674
10^{-12}	301	40	21	640
10^{-10}	301	101	23	715
10^{-8}	300	60	24	771
10^{-6}	301	73	23	726
10^{-4}	287	102	24	721
10^{-2}	251	170	26	716

Table 3: Test Results for Two-sided Differences

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	306	40	22	1,306
10^{-12}	305	32	20	1,171
10^{-10}	301	42	23	1,264
10^{-8}	301	50	22	1,343
10^{-6}	301	74	25	1,439
10^{-4}	287	101	26	1,642
10^{-2}	255	137	25	1,274

Table 4: Test Results for Fourth-order Formula

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	304	35	22	363
10^{-12}	304	39	22	363
10^{-10}	299	50	25	410
10^{-8}	283	64	25	441
10^{-6}	260	82	28	555
10^{-4}	202	104	29	545
10^{-2}	117	157	29	374

Table 5: Test Results for Forward Differences (Monotone Line Search)

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	306	37	22	670
10^{-12}	300	35	23	632
10^{-10}	300	36	22	646
10^{-8}	295	41	21	649
10^{-6}	280	55	22	658
10^{-4}	242	72	23	681
10^{-2}	136	134	29	751

Table 6: Test Results for Two-sided Differences (Monotone Line Search)

The results are quite surprising and depend heavily on the new non-monotone line search strategy. There are no significant differences in the number of test problems solved between the three different formulae despite of the increasing theoretical approximation orders. Moreover, we are able to solve about 80 % of the test examples in case of extremely noisy function values with at most two correct digits. If we take the number of equivalent function calls into account, we conclude that forward differences are more efficient than higher order formulae.

3.4 Testing Non-Monotone Line Search under Noise

First of all,, we repeat the same set of test runs made in the previous section, but now with the standard monotone line search. The results are summarized in Tables 5 to 7. In this case, we get a slight improvement of the number of successful test runs with increased order of the difference formula. However, the we can solve at most 50 % of all problems successfully in case of the fourth-order formula compared to 80 % in case of forward differences and non-monotone line search.

To investigate the situation in more detail, we proceed from the fourth-order formula and list now the number of successful returns, n_{succ}, and the number of test runs where NLPQLP terminated because of an error message

ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{equ}
0	306	37	22	1,308
10^{-12}	300	31	20	1,169
10^{-10}	298	34	21	1,206
10^{-8}	299	40	21	1,226
10^{-6}	283	50	22	1,316
10^{-4}	246	68	24	1,469
10^{-2}	156	115	28	1,295

Table 7: Test Results for Fourth-order Formula (Monotone Line Search)

	monotone		non-monotone	
ϵ_{err}	n_{succ}	n_{err}	n_{succ}	n_{err}
0	306	10	308	5
10^{-12}	300	17	305	8
10^{-10}	298	33	301	12
10^{-8}	299	53	301	11
10^{-6}	283	99	299	19
10^{-4}	246	157	285	34
10^{-2}	156	315	251	65

Table 8: Successful and Non-Successful Returns

IFAIL>0, n_{err}. The results are listed in Table 8. They clearly indicate the advantage of non-monotone line searches over the monotone ones. Robustness and stability of the SQP method are significantly increased especially in case of large noise in the function evaluations.

A further series of test runs concerns the situation that a user fixes the tolerance η for gradient approximations, e.g., to $\eta = 10^{-7}$. This is a unlikely worst-case scenario and should only happen in a a situation, where a *black-box* derivative calculation is used and where a user is not aware of the accuracy by which derivatives are approximated. Whereas nearly all test runs break down with error messages for the monotone line search and large random perturbations, the non-monotone line search is still able to find to terminate in at least 30 % NLPQLP calls, see Table 9.

4 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 6 parallel function evaluation per iterations are required for conducting

	monotone		non-monotone	
ϵ_{err}	n_{succ}	n_{err}	n_{succ}	n_{err}
0	306	7	307	6
10^{-12}	304	28	303	10
10^{-10}	292	66	302	11
10^{-8}	210	134	286	28
10^{-6}	71	248	188	124
10^{-4}	23	295	119	196
10^{-2}	23	295	116	198

Table 9: Successful and Non-Successful Returns, $\eta = 10^{-7}$

the line search. Significant performance improvement is achieved by the non-monotone line search especially in case of noisy function values and numerical differentiation. There are no differences in the number of test problems solved between forward differences, two-sided differences, and fourth-order formula, even not in case of severe random perturbations. With the new non-monotone line search, we are able to solve about 80 % of the test examples in case of extremely noisy function values with at most two correct digits and forward differences for derivative calculations.

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] 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

[5] 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

[6] 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

[7] 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

[8] Bünner M.J., Schittkowski K., van de Braak G. (2002): *Optimal design of surface acoustic wave filters for signal processing by mixed-integer nonlinear programming*, submitted for publication

[9] 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

[10] 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

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

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

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

[14] 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

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

[16] 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

[17] 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

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

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

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

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

[22] Frias J.M., Oliveira J.C, Schittkowski K. (2001): *Modelling of maltodextrin DE12 drying process in a convection oven*, to appear: Applied Mathematical Modelling

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

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

[25] Hough P.D., Kolda T.G., Torczon V.J. (2001): *Asynchronous parallel pattern search for nonlinear optimization*, to appear: SIAM J. Scientific Computing

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

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

[28] 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

[29] 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

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

[31] 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

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

[33] 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

[34] 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

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

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

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

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

[39] 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

[40] 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

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

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

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

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

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

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

[47] 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

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

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

[50] Schittkowski K. (2001): *NLPQLP: A new Fortran implementation of a sequential quadratic programming algorithm - user's guide, version 1.6*, Report, Department of Mathematics, University of Bayreuth

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

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

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

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

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

[56] Schittkowski K. (2004): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and nonmonotone line search*, Report, Department of Computer Science, University of Bayreuth

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

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

[59] 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

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

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

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

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