A Robust Implementation of a Sequential Quadratic Programming Algorithm with Successive Error Restoration

Address: Prof. K. Schittkowski
Department of Computer Science
University of Bayreuth
D - 95440 Bayreuth

Phone: (+49) 921 557750

E-mail: klaus.schittkowski@uni-bayreuth.de

Date: January, 2010

Abstract

We consider sequential quadratic programming (SQP) methods for solving constrained nonlinear programming problems. It is generally believed that SQP methods are sensitive to the accuracy by which partial derivatives are provided. One reason is that differences of gradients of the Lagrangian function are used for updating a quasi-Newton matrix, e.g., by the BFGS formula. The purpose of this paper is to show by numerical experimentation that the method can be stabilized substantially. Even in case of large random errors leading to partial derivatives with at most one correct digit, termination subject to an accuracy of 10^{-7} can be achieved, at least in 90 % of 306 problems of a standard test suite. The algorithm is stabilized by a non-monotone line search and, in addition, by internal and external restarts in case of errors when computing the search direction due to inaccurate derivatives. Additional safeguards are implemented to overcome the situation that an intermediate feasible iterate might get an objective function value smaller than the final stationary point. In addition, we show how initial and periodic scaled restarts improve the efficiency in a situation with very slow convergence.

Keywords: SQP, sequential quadratic programming, nonlinear programming, non-monotone line search, restart, line search, numerical algorithm,

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 paradigm that function and gradient values can be evaluated with sufficiently high precision, see Schittkowski [23, 24] based on academic and Schittkowski et al. [34] based on structural mechanical engineering test problems.

SQP methods can be extended to solve also nonlinear least squares problems efficiently, see Schittkowski [28, 29], to run under distributed systems by using a parallel line search, see Schittkowski [33], or to handle problems with very many constraints, see Schittkowski [31]. Meanwhile, there exist hundreds of commercial and academic applications of SQP codes.

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 with the error message. To avoid termination and to continue the iteration, one possibility is to make use of non-monotone line search. 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 parameter. The general idea is described in Dai and Schittkowski [5], where a convergence proof for the constrained case is presented.

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 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. We propose to perform 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 anything about the internal numerical structure of the optimization problem. If requested by the user, the first BFGS update can be 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.

For our numerical tests, we use two collections with 306 test problems, see Hock and Schittkowski [11] and in Schittkowski [27]. 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. [3].

In Section 2 we outline the general mathematical structure of an SQP algorithm, especially the non-monotone line search. Section 3 contains some numerical results showing the sensitivity of an SQP algorithm with respect to gradient approximations under uncertainty. We test and compare 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 main result is that even in case of large random errors leading to partial derivatives with at most one correct digit, termination subject to an accuracy of 10^{-7} can be achieved, at least in 90 % of 306 problems of a standard test suite.

2 Sequential Quadratic Programming Methods

Sequential quadratic programming (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 [36] in form of a review, or in Spellucci [35] in form of an extensive text book. Their excellent numerical performance is tested and compared with other methods in Schittkowski [23], and since many years they belong to the most frequently used algorithms to solve practical optimization problems.

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

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

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

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

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

quadratically, where $x \in \mathbb{R}^n$ is the primal variable and $u = (u_1, \ldots, u_m)^T \in \mathbb{R}^m$ the dual variable or the vector of Lagrange multipliers, respectively.

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

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

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

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

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 $\delta \in \mathbb{R}$, leading to a modified quadratic programming problem, see Schittkowski [26] for details.

However, the linear constraints in (4) can become inconsistent even if the original problem (2) is solvable. As in Powell [20], we add an additional variable δ to (4) and solve an $(n+1)$-dimensional subproblem with consistent constraints.

Another numerical drawback of (4) is that gradients of all constraints must be reevaluated in each iteration step. But if x_k is close to the solution, the calculation of gradients of inactive nonlinear constraints is redundant. Given a constant $\varepsilon > 0$, we define the sets

$$\overline{I}_1^{(k)} \doteq \{j \in I : \ g_j(x_k) \leq \varepsilon \text{ or } v_k^{(j)} > 0\} \ , \quad \overline{I}_2^{(k)} \doteq I \setminus \overline{I}_1^{(k)} \ , \tag{6}$$

with $v_k = (v_k^{(1)}, \ldots, v_k^{(m)})^T$ and $I \doteq \{j : m_e < j \leq m\}$, and solve the following subproblem

in each step,

$$\begin{array}{rl} \text{minimize} & \frac{1}{2}d^T B_k d + \nabla f(x_k)^T d + \frac{1}{2}\varrho_k \delta^2 \\ d \in I\!R^n, \delta \in [0,1]: & \nabla g_j(x_k)^T d + (1-\delta) g_j(x_k) = 0 \ \ , \ \ j = 1, \ldots, m_e, \\ & \nabla g_j(x_k)^T d + (1-\delta) g_j(x_k) \geq 0 \ \ , \ \ j \in \overline{I}_1^{(k)}, \\ & \nabla g_j(x_{\kappa(k,j)})^T d + g_j(x_k) \geq 0 \ \ , \ j \in \overline{I}_2^{(k)}. \end{array} \tag{7}$$

The indices $\kappa(k,j) \leq k$ denote previous iterates where the corresponding gradient has been evaluated the last time. We start with $\overline{I}_1^{(0)} \doteq I$ and $\overline{I}_2^{(0)} \doteq \emptyset$ and reevaluate constraint gradients in subsequent iterations only if the constraint belongs to the active set $\overline{I}_1^{(k)}$. The remaining rows of the Jacobian matrix remain filled with previously computed gradients.

We denote by (d_k, u_k) the solution of (7), where u_k is the multiplier vector, and by δ_k the additional variable to prevent inconsistent linear constraints. Under the linear independency constraint qualification (LICQ), it is easy to see that $\delta_k < 1$. B_k is a positive-definite approximation of the Hessian of the Lagrange function. For the global convergence analysis, any choice of B_k is appropriate as long as the eigenvalues are bounded away from zero.

However, to guarantee a superlinear convergence rate, we update B_k by the BFGS quasi-Newton method together with a stabilization to guarantee positive definite matrices, see Powell [20, 21]. The penalty parameter ϱ_k is required to reduce the perturbation of the search direction by the additional variable δ as much as possible. A suitable choice is given in Schittkowski [24].

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

$$\phi_r(\alpha) \doteq \psi_r \left(\begin{pmatrix} x \\ v \end{pmatrix} + \alpha \begin{pmatrix} d \\ u - v \end{pmatrix} \right) \tag{8}$$

with a suitable penalty function $\psi_r(x, v)$. One possibility is to apply the augmented Lagrangian function

$$\psi_r(x,v) \doteq f(x) - \sum_{j \in J} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) - \frac{1}{2} \sum_{j \in K} v_j^2 / r_j \ \ , \tag{9}$$

with $J \doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, g_j(x) \leq v_j / r_j\}$ and $K \doteq \{1, \ldots, m\} \setminus J$, cf. Schittkowski [24]. 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 be carefully chosen to guarantee a descent direction

of the merit function, see Schittkowski [24] or Wolfe [39] in a more general setting, i.e., to get

$$\phi'_{r_k}(0) = \bigtriangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < 0 \quad . \tag{10}$$

The implementation of a line search algorithm is a crucial 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 a certain Armijo [1] condition, i.e., a sufficient descent condition of the merit function (9) which guarantees convergence to a stationary point. However, to take the curvature of the merit function into account, we need some kind of compromise between a polynomial interpolation, typically a quadratic one, and a reduction of the stepsize by a given factor, until as a stopping criterion is reached. 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.

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

Start: $\alpha_0 \doteq 1$

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 $\bar{\alpha}_i \doteq \dfrac{0.5\ \alpha_i^2\ \phi'_r(0)}{\alpha_i \phi'_r(0) - \phi_r(\alpha_i) + \phi_r(0)}$.

3) Let $\alpha_{i+1} \doteq \max(\beta\ \alpha_i, \bar{\alpha}_i)$.

The algorithm goes back to Powell [20] and corresponding convergence results are found in Schittkowski [24]. $\bar{\alpha}_i$ 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 $(0, 1]$. Additional safeguards are required, for example to prevent violation of bounds. Algorithm 2.1 assumes that $\phi_r(1)$ is known before calling the procedure, i.e., that the corresponding function values are given. We stop the algorithm, if sufficient descent is not observed after a certain number of iterations. If the tested stepsize falls

below machine precision or the accuracy by which model function values are computed, the merit function cannot decrease further.

It is possible that Algorithm 2.1 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

$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)<=j<=k} \phi_{r_j}(0) + \alpha_k \mu \phi'_{r_k}(0) \tag{11}$$

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 described in Dai [4] and Dai and Schittkowski [5], where convergence proofs are presented. The general idea goes back to Grippo, Lampariello, and Lucidi [8], and was extended to constrained optimization and trust region methods in a series of subsequent papers, see Bonnans et al. [2], Deng et al. [6], Grippo et al. [9, 10], Ke and Han [12], Ke et al. [13], Lucidi et al. [15], Panier and Tits [19], Raydan [22], and Toint [37, 38].

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 [21] or Stoer [36],

$$B_{k+1} \doteq B_k + \frac{q_k q_k^T}{p_k^T q_k} - \frac{B_k p_k p_k^T B_k}{p_k^T B_k p_k} \quad , \tag{12}$$

where $q_k \doteq \bigtriangledown_x L(x_{k+1}, u_k) - \bigtriangledown_x L(x_k, u_k)$ and $p_k \doteq x_{k+1} - x_k$. Special safeguards guarantee that $p_k^T q_k > 0$ and that thus all matrices B_k remain positive definite provided that B_0 is positive definite. A possible scaling factor and restart procedure is to replace an actual B_k by $\gamma_k I$ before performing the upate (12), where $\gamma_k = p_k^T q_k / p_k^T p_k$ and where I denotes the identity matrix, see for example Liu and Nocedal [14]. Scaled restarts are recommended, if, e.g., the convergence turns out to become extremely slow.

3 Test Results

Our numerical tests use the 306 academic and real-life test problems published in Hock and Schittkowski [11] and in Schittkowski [27]. They their usage is described in Schitt-

kowski [32]. The test examples are provided with exact solutions, either known from analytical investigations *by hand* or from the best numerical data found so far.

The Fortran implementation of the SQP method introduced in the previous section, is called NLPQLP, see Schittkowski [33]. The code is frequently used at 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 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 [7] based on numerically stable orthogonal decompositions, see Schittkowski [30]. NLPQLP is executed with termination accuracy 10^{-7} and a maximum number of 500 iterations.

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

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

or

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

and

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

where $\| \ldots \|_\infty$ 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 $x^\star$ 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 and if

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

or

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

and

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

For our numerical tests, we use $\epsilon = 0.01$ to determine a successful return, i.e., we require a final accuracy of one per cent.

Since analytical derivatives are not available for all problems, we approximate them numerically. Two popular difference formulae are forward differences

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

and two-sided differences,

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{2\eta_i}\Big(f(x+\eta_i e_i) - f(x-\eta_i e_i\Big) \tag{14}$$

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, and $\eta = {\eta_m}^{1/3}$ for two-sided differences. 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.

Higher order formulae are available, but require too many additional function evaluations for making them applicable in complex simulation systems. Moreover, they often do not yield better numerical results, at least for our numerical tests.

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

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. However, additional function evaluations needed for gradient approximations, are not counted for n_{func}. Their average number is n_{func} for the forward difference or $2 \times n_{func}$ for the two-sided difference formulae. One gradient computation corresponds to one iteration of the SQP method.

For our numerical tests, we use the two-sided difference formula (14). To test the stability of the SQP code, we add randomly generated noise to each function value. Non-monotone line search is applied with a queue length of $p = 30$ 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 $\rho = 10^4$.

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 10.1, EM64T, under Windows XP64 and Dual Core AMD Opteron Processor 265, 1.81 GHz, with 8 GB of RAM.

To compare the different stabilization approaches, we apply three different scenarios,

Table 1	-	monotone line search, no restarts
Table 2	-	non-monotone line search, no restarts
Table 3	-	non-monotone line search and restarts

The corresponding results are evaluated for increasing random perturbations (ϵ_{err}). More precisely, if ρ denotes a uniformly distributed random number between 0 and 1, we

ϵ_{err}	n_{succ}	n_{func}	n_{grad}
0	304	35	22
10^{-12}	303	36	22
10^{-10}	297	40	23
10^{-8}	293	43	23
10^{-6}	280	57	24
10^{-4}	243	74	26
10^{-2}	133	133	32

Table 1: Test Results for Monotone Line Search without Restarts

ϵ_{err}	n_{succ}	n_{func}	n_{grad}
0	306	38	22
10^{-12}	303	37	23
10^{-10}	300	43	25
10^{-8}	300	53	24
10^{-6}	295	71	25
10^{-4}	268	116	30
10^{-2}	204	185	33

Table 2: Test Results for Non-Monotone Line Search without Restarts

replace $f(x_k)$ by $f(x_k)(1 + \epsilon_{err}(2\rho - 1))$ for 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.

External restarts within the test program are performed if a local solution is found with a function value higher than a feasible iterate with a better objective function value. This happens due to the non-monotone line search for three problems in case of perturbation by $\epsilon_{err} = 10^{-4}$ and for 27 problems in case of $\epsilon_{err} = 10^{-2}$. The situation is reported by NLPQLP, and a restart with a multiple of the identity matrix is performed.

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 90 % of the test examples in case of extremely noisy function values with at most one correct digit in partial derivative values.

However, the stabilization process is costly. The more test problems are successfully solved, the more iterations, especially function evaluations, are needed. The additional line search iterations and internal or external restarts cost about 20 times as many function and about three times as many gradient evaluations in the worst case.

In some situations, the convergence of an SQP method becomes quite slow for many reasons, e.g., badly scaled variables or functions, inaccurate derivatives, or inaccurate

ϵ_{err}	n_{succ}	n_{func}	n_{grad}
0	306	38	22
10^{-12}	305	39	23
10^{-10}	300	47	24
10^{-8}	303	83	27
10^{-6}	302	105	30
10^{-4}	297	318	43
10^{-2}	279	647	64

Table 3: Test Results for Non-Monotone Line Search and Restarts

solutions of the quadratic program (4). In these situations, errors in the search direction or the partial derivatives influence the update procedure (12) and the quasi-Newton matrices B_k are getting more and more inaccurate.

A frequently proposed remedy is to restart the update algorithm by a replacing the actual matrix B_k by the initial matrix B_0 or any similar one, if more information is available. One possibility is to multiply a special scaling factor with the identity matrix, i.e., to let $B_k \doteq \gamma_k I$ for selected iterates k, where $\gamma_k \doteq p_k^T q_k / p_k^T p_k$ and where I denotes the identity matrix, see for example Liu and Nocedal [14]. Here, $q_k \doteq \bigtriangledown_x L(x_{k+1}, u_k) - \bigtriangledown_x L(x_k, u_k)$ and $p_k \doteq x_{k+1} - x_k$.

Scaled restarts are recommended, if the convergence turns out to become extremely slow. To illustrate the situation, we consider a few test runs, where the examples are generated by discretizing a two-dimensional elliptic partial differential equation, see Maurer and Mittelmann [16, 17]. The original formulation is that of an optimal control problem and state and control variables are discretized.

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. 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. They are obtained by applying the five-star formula to approximate second partial derivatives.

Tables 4 to 6 contain numerical results first for standard tolerances with termination accuracy 10^{-7}. For the results of the the first table, we execute NLPQLP without scaling by starting the BFGS updated with the identity matrix, see Table 4. Results for scaled initial matrices are reported in Table 5, and in addition after each seven steps, Table 6. The total number of iterations is limited to 200. Besides of dimensions and numbers of function and gradient evaluations, we also report the final objective function value $f(x)$ and corresponding constraint violation $r(x)$.

We observe a significant speedup for scaled restarts every seven iterations. Initial scaling alone and more than seven restarts do not lead to better results.

problem	n	m_e	n_{func}	n_{grad}	$f(x)$	$r(x)$
EX 1	722	361	64	64	0.45903100E-1	0.33E-10
EX 2	722	361	109	109	0.40390974E-1	0.22E-8
EX 3	722	361	88	88	0.11009561E+0	0.49E-9
EX 4	798	437	113	113	0.75833416E-1	0.12E-9
EX 5	798	437	200	200	0.51376012E-1	0.60E-5

Table 4: Elliptic Control Problem, Test Results without Restart

problem	n	m_e	n_{func}	n_{grad}	$f(x)$	$r(x)$
EX 1	722	361	64	64	0.45903100E-1	0.64E-11
EX 2	722	361	108	108	0.40390974E-1	0.21E-8
EX 3	722	361	75	75	0.11009568E+0	0.46E-9
EX 4	798	437	108	108	0.75833417E-1	0.63E-9
EX 5	798	437	200	200	0.51369466E-1	0.14E-6

Table 5: Elliptic Control Problem, Test Results with Initial Scaling

problem	n m_e	n_{func}	n_{grad}	$f(x)$	$r(x)$	
EX 1	722	361	20	20	0.45903160E-1	0.19E-7
EX 2	722	361	21	21	0.40390974E-1	0.79E-7
EX 3	722	361	41	41	0.11009561E+0	0.23E-9
EX 4	798	437	58	58	0.75833423E-1	0.44E-8
EX 5	798	437	112	112	0.51365403E-1	0.15E-12

Table 6: Elliptic Control Problem, Test Results with Scaled Restarts Every 7th Step

4 Conclusions

We present a modification of an SQP algorithm designed for execution under distributed computing (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. 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 severe error situations. We are able to solve about 90 % of a standard set of 306 test examples 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] 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

[3] 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

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

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

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

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

[8] 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

[9] 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

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

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

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

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

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

[15] 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

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

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

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

[19] 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

[20] 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

[21] 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

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

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

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

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

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

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

[28] 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

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

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

[31] 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

[32] 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

[33] Schittkowski K. (2009): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user's guide, version 3.0*, Report, Department of Computer Science, University of Bayreuth

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

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

[36] 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

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

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

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