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

- 1st Revision -

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: May, 2010

Abstract

We consider sequential quadratic programming (SQP) methods for solving constrained nonlinear programming problems. It is generally believed that these 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. The algorithm applies non-monotone line search and internal and external restarts in case of errors due to inaccurate derivatives while computing the search direction. 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 in 90 % of 306 problems of a standard test suite. On the other hand, the original version with monotone line search and without restarts solves only 30 % of these problems under the same test environment. In addition, we show how initial and periodic scaled restarts improve the efficiency in situations with slow convergence.

Keywords: SQP; sequential quadratic programming; nonlinear programming; line search; restart; numerical algorithm; noisy functions

1 Introduction

We consider the general optimization problem to minimize an objective function 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 [18, 19] based on academic and Schittkowski et al. [28] based on structural mechanical engineering test problems.

SQP methods can be extended to solve also nonlinear least squares problems efficiently, see Schittkowski [22, 23], to run under distributed systems by using a parallel line search, see Schittkowski [27], or to handle problems with very many constraints, see Schittkowski [25]. A combination of an SQP and IPM approach is implemented by Sachsenberg [17] to solve very large and sparse problems. 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

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

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. This is an important assumption to be able 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.

Restarting procedures are not new and are for example discussed by Gill and Leonard [6] in the framework of limited-memory updates. See also an early paper of Nocedal [13] for a modified BFGS update. Restarts are also implemented in existing software, see e.g. the code `do2nlp` of Spellucci [29].

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. It is possible, for example, to start the BFGS update procedure from a multiple of the identity matrix, which takes into account information from the previous and the actual iterates. 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 [9] and in Schittkowski [21]. 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 by forward differences 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 in 90 % of the test runs.

Another set of test problems is derived from fully discretized semi-linear elliptic control problems, see Maurer and Mittelmann [11, 12]. They possess a different numerical structure, i.e., a large number variables and weakly nonlinear equality constraints, and are easily scalable to larger size. We use a subset of them with only 722 and 798 variables, since our SQP code is unable to exploit sparsity. Some test results for up to 5.000.000 variables and 2.500.000 constraints are presented in Sachsenberg [17]. The resulting finite dimensional nonlinear programs require a relatively large number of iterations until successful termination. We show how different automated scaling procedures accelerate the convergence drastically.

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 [30] or in Boggs and Tolle [**?**]. Their excellent numerical performance is tested and compared with other methods in Schittkowski [18], 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 $I\!R^n$.

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

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

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

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 we solve the quadratic programming problem

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

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

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

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

Although we are able to guarantee that the matrix B_k is positive definite, it is possible that (4) is not solvable due to inconsistent constraints. As proposed by Powell [14],

one possible remedy is to introduce an additional variable to (4) leading to a modified quadratic programming problem, see Schittkowski [20] for details. Given a constant $\varepsilon > 0$, we define the sets

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

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,

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

We denote by (d_k, u_k) the solution of the quadratic program (4), 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 $\delta_k < 1$. For the global convergence analysis, any choice of B_k is appropriate as long as the eigenvalues are bounded away from zero. (7) could further be modified by an active set strategy, see Schittkowski [20], but this is not relevant for the purpose of this paper. 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 [14, 15]. 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 [19], which guarantees the sufficient descent of the search direction and prevents undue increase.

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 investigated by Rockafellar [16],

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

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 [19] and Rockafellar [16]. The objective function is *penalized* as soon as an

iterate leaves the feasible domain. The corresponding penalty parameters r_j, $j = 1, \ldots, m$, which control the degree of constraint violation, must be carefully chosen to guarantee a descent direction of the merit function, see Schittkowski [19] or Wolfe [33] 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. 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: $\alpha_0 = 1$

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 [14] and corresponding convergence results are found in Schittkowski [19]. $\bar{\alpha}_i$ 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 $\phi_r'(0)$ becomes positive due to round-off errors in the partial derivatives or 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. Thus, 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 $p(k) \doteq \min\{k, p\}$, 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, e.g., Toint [31, 32].

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 [15] or Stoer [30],

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

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. Formula (12) is extremely sensitive subject to the accuracy by which partial derivatives are provided. Worst of all, we use them only for computing the difference vector q_k, by which we introduce additional truncation errors.

A frequently proposed remedy is to restart the BFGS-update algorithm (12) by replacing the actual matrix B_k by a scalar multiple of 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 $B_k = \gamma_k I$ for selected iterates k, where

$$\gamma_k \doteq \frac{p_k^T q_k}{p_k^T p_k} \tag{13}$$

and where I denotes the identity matrix, see for example Liu and Nocedal [10].

3 Numerical Results for 306 Test Problems under Random Noise

Our numerical tests use the 306 academic and real-life test problems published in Hock and Schittkowski [9] and in Schittkowski [21]. The usage of the corresponding Fortran codes is documented in Schittkowski [26]. The test examples are provided with 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 [27]. 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 [24]. 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

$$f(x_k) - f(x^\star) < \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) \neq 0 \tag{14}$$

or

$$f(x_k) < \epsilon \ , \ \text{if } f(x^\star) = 0 \tag{15}$$

and

$$r(x_k) \doteq \|g(x_k)^-\|_\infty < \epsilon^2 \ , \tag{16}$$

where $\|\ldots\|_\infty$ denotes the maximum norm and $g_j(x_k)^- \doteq \min(0, g_j(x_k))$, $j > m_e$, and $g_j(x_k)^- \doteq g_j(x_k)$ 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 the internal termination conditions are satisfied subject to a reasonably small tolerance and if

$$f(x_k) - f(x^\star) \geq \epsilon |f(x^\star)| \ , \ \text{if } f(x^\star) \neq 0 \tag{17}$$

or

$$f(x_k) \geq \epsilon \ , \ \text{if } f(x^\star) = 0 \tag{18}$$

and if (16) holds.

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

Since analytical derivatives are not available for all problems, partial derivatives are approximated by forward differences,

$$\frac{\partial}{\partial x_i} f(x) \approx \frac{1}{\eta_i}\Big(f(x + \eta_i e_i) - f(x)\Big) \ . \tag{19}$$

Here, $\eta_i \doteq \eta \max(10^{-5}, |x_i|)$ and e_i is the i-th unit vector, $i = 1, \ldots, n$. The tolerance η is set to $\eta \doteq {\eta_m}^{1/2}$, where 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 approximated.

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

In the subsequent table, 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, the additional function evaluations needed for gradient approximations are not counted for n_{func}. Their average number is n_{func} for the forward difference formula. One gradient computation corresponds to one iteration of the SQP method.

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 1, are evaluated for increasing random perturbations (ϵ_{err}). More precisely, if ν denotes a uniformly distributed random number

	$\rho = 0$, $p = 0$			$\rho = 0$, $p = 40$			$\rho = 10^4$, $p = 40$		
ϵ_{err}	n_{succ}	n_{func}	n_{grad}	n_{succ}	n_{func}	n_{grad}	n_{succ}	n_{func}	n_{grad}
0	306	36	22	306	37	22	306	37	22
10^{-12}	304	40	23	306	66	26	306	66	26
10^{-10}	301	45	24	305	72	28	306	72	29
10^{-8}	276	58	26	302	103	31	304	120	32
10^{-6}	248	77	30	295	167	40	302	222	49
10^{-4}	178	97	30	273	220	43	295	379	62
10^{-2}	92	163	36	224	308	48	279	630	78

Table 1: Test Results for 306 Academic Test Problems

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 Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 11.0, 64 bit, under Windows 7 and Intel(R) Core(TM) i7 CPU 860, 2.8 GHz, with 8 GB RAM.

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.

4 Numerical Results for Semi-Linear Elliptic Control 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 (12) and the quasi-Newton matrices B_k are getting more and more inaccurate.

Scaled restarts as described in Section 2, see (13), 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 [11, 12]. 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,

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 2: Elliptic Control Problems

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 2 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 the same as before.

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 [17].

Table 3 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 2 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., B_k,

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 3: Elliptic Control Problems, Number of Iterations

- no scaling, i.e., $B_0 = I$,
- initial scaling, i.e., the update procedure is started at $B_1 = \gamma_1 I$, see (13),
- B_k is reset if $\gamma_k \leq \sqrt{\epsilon_t}$, where ϵ_t is the termination accuracy,
- B_k is reset every 7th step,
- B_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.

5 Conclusions

We present a modification of an SQP algorithm designed for solving nonlinear programming problems with noisy function values. A non-monotone line search is applied which allows an intermediate increase of a merit function. Scaled restarts are performed in error situations, i.e., in situations which would otherwise lead to false terminations. Numerical results indicate extreme stability and robustness on a set of 306 standard test problems. We are able to solve about 90 % of a standard set of 306 test examples in case of extremely noisy function values with relative accuracy of 1 % and numerical differentiation

by forward differences. In the worst case, only one digit of a partial derivative value is correct. On the other hand, the original version with monotone line search and without restarts solves only 30 % of these problems under the same test environment. For all test runs, we use a termination accuracy of 10^{-7} which is not adapted to the noise level.

In real-life applications with complex simulation procedures, perturbed function values are not unusual. Analytical gradients are often not available and a forward difference formula is applied to approximate partial derivatives. However, the efficiency of SQP methods depends more on the accuracy of partial derivatives than on the accuracy by which function values are computed. One reason is the updating procedure of an internal quasi-Newton method which requires computation of differences of Lagrangian gradients at neighbored iterates.

Another difficulty is that the convergence of an SQP algorithm is sometimes slow, i.e., the code needs a large number of iterations until termination. There are many possible reasons, but in most situations badly scaled problem functions or variables lead to long iteration cycles. It is often recommended to perform a restart as soon as slow termination is observed. To test this situation, the previously used set of 306 test examples is not appropriate and we use a small set of relatively large problems obtained by discretizing semi-linear elliptic control problems. Our numerical results indicate that scaled adaptive or periodic restarts with a short cycle length, say between 7 and 15, are appropriate to improve convergence speed significantly.

Our presented numerical results depend on the setting of parameters, by which we control the test environment, e.g., restart scaling (ρ), queue length (p), optimality check (ϵ), ..., and the execution of the SQP code, e.g., termination accuracy (ϵ_t), maximum number of iterations (500). Much more parameter values are fixed inside the SQP code. An appropriate choice influences numerical tests on the academic level, but also the practical usage. As certain values might increase reliability (n_{succ}) and, vice versa, decrease efficiency (n_{grad}), other values might have the opposite effect. It is not possible to find the 'best' combination of all settings, since they strongly depend on the numerical structure of the underlying optimization problem. Even for the underlying two test problem sets, one would need a large number of additional numerical results. Their complete documentation would break the limitations of a publication, and very likely not lead to any further insight. To a certain extend, the reader should accept that there have been initial experiments by which a suitable test frame was fixed.

References

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

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

[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] Gill P.E., Leonard M.W. (2003): *Limited-memory reduced-Hessian methods for unconstrained optimization*, SIAM Journal on Optimization, Vol. 14, 380–401

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

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

[11] 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, Vol. 16, 29–55

[12] 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, Vol. 18, 141–160

[13] Nocedal J. (1980): *Updating quasi-Newton matrices with limited storage*, Mathematics of Computation, Vol. 35, 773–782

[14] 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

[15] 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

[16] Rockafellar T. (1974): *Augmented Lagrangian multiplier functions and duality in nonconvex programming*, SIAM Journal of Control, Vol. 12, 268-285

[17] 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

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

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

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

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

[22] 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

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

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

[25] 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

[26] 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

[27] 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

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

[29] Spellucci P.: *DONLP2 users guide*, Technical University at Darmstadt, Department of Mathematics, Darmstadt, Germany

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

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

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

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