Optim Lett (2015) 9:1271–1282
DOI 10.1007/s11590-015-0863-x

ORIGINAL PAPER

A combined SQP–IPM algorithm for solving large-scale nonlinear optimization problems

Björn Sachsenberg · Klaus Schittkowski

Received: 7 April 2014 / Accepted: 10 February 2015 / Published online: 27 February 2015

Abstract We consider a combined IPM–SQP method to solve smooth nonlinear optimization problems, which may possess a large number of variables and a sparse Jacobian matrix of the constraints. Basically, the algorithm is a sequential quadratic programming (SQP) method, where the quadratic programming subproblem is solved by a primal-dual interior point method (IPM). A special feature of the algorithm is that the quadratic programming subproblem does not need to become exactly solved. To solve large optimization problems, either a limited-memory BFGS update to approximate the Hessian of the Lagrangian function is applied or the user specifies the Hessian by himself. Numerical results are presented for the 306 small and dense Hock-Schittkowski problems, for 13 large semi-linear elliptic control problems after a suitable discretization, and for 35 examples of the CUTEr test problem collection with more than 5000 variables.

Keywords Nonlinear programming · Large-scale optimization · SQP methods · IPM methods · Numerical tests

1 Introduction

We consider the nonlinear programming problem to minimize an objective function subject to inequality constraints,

$$x \in I\!R^n : \begin{array}{ll} \min & f(x) \\ & g(x) \leq 0, \end{array} \tag{1}$$

B. Sachsenberg · K. Schittkowski (✉)
Department of Computer Science, University of Bayreuth, 95440 Bayreuth, Germany
e-mail: klaus.schittkowski@uni-bayreuth.de

where x is an n-dimensional parameter vector. It is assumed that the functions $f(x)$ and $g(x) = (g_1(x), \ldots, g_m(x))^T$ are twice continuously differentiable on the $\mathbb{R}^n$. To illustrate the underlying mathematical algorithm, we omit equality constraints and upper and lower bounds of variables to facilitate the notation.

The basic idea is to mix a sequential quadratic programming (SQP) and an interior point method (IPM) for nonlinear programming. In an outer loop, a sequence of quadratic programming subproblems is constructed by approximating the Lagrangian function

$$L(x, u) := f(x) + u^T g(x) \tag{2}$$

quadratically and by linearizing the constraints. The resulting quadratic programming subproblem (QP)

$$d \in \mathbb{R}^n : \begin{array}{l} \min \ \frac{1}{2} d^T H(x_k, u_k) d + \nabla f(x_k)^T d \\ g(x_k) + \nabla g(x_k) d \le 0 \end{array} \tag{3}$$

is then solved by an interior point solver. A pair (x_k, u_k) denotes the current iterate in the primal-dual space, $H(x_k, u_k)$ the Hessian of the Lagrangian function of (2), i.e., $H(x_k, u_k) = \nabla_{xx} L(x_k, u_k)$ or a suitable approximation, and $\nabla g(x_k)$ is the Jacobian matrix of the vector of constraints. We call $x_k \in \mathbb{R}^n$ the primal and $u_k \in \mathbb{R}^m$ the dual variable or the multiplier vector, respectively. The index k is an iteration index and stands for the k-th step of the optimization algorithm, $k = 0, 1, 2, \ldots$.

Sequential quadratic programming methods are well known, and numerous modifications and extensions have been published on SQP methods. Review papers are given by Boggs and Tolle [2] and Gould and Toint [12]. Most optimization textbooks have chapters on SQP methods, see, for example, see Fletcher [9], Gill, Murray and Wright [10], and Sun and Yuan [25]. A method for solving large scale quadratic programs is introduced in Cafieri et al. [4].

An alternative approach is the interior point method (IPM) developed in the 90's, see, e.g., Griva et al. [14], There are numerous alternative algorithms and implementations available differing especially by their stabilization approaches, by which convergence towards a stationary point can be guaranteed, see, e.g., Byrd, Gilbert, and Nocedal [3], D'Apuzzo et al. [7], or the review paper of Gondzio [11]. The underlying strategy consists of replacing the constrained optimization problem (1) by a simpler one without inequality constraints,

$$x \in \mathbb{R}^n, s \in \mathbb{R}^m : \begin{array}{l} \min \ \ f(x) - \mu \sum_{j=1}^m \log\left(s_j\right) \\ g(x) + s = 0. \end{array} \tag{4}$$

Here, $s = (s_1, \ldots, s_m)^T > 0$ denotes a vector of m slack variables, where the positivity has to be guaranteed separately. The smaller the so-called barrier term μ is, the closer are the solutions of both problems. It is essential to understand that we now have $n+m$ primal variables x and s, and in addition m dual variables $u \in \mathbb{R}^m$, the multipliers of the equality constraints of (4). Since, however, these multiplier approximations are also used to approximate the multipliers of the inequality constraints of (1), we also require that $u > 0$ throughout the algorithm.

By applying Newton's method to the KKT conditions (4), we construct a sequence of systems of linear equations, see, for example, Byrd, Gilbert, and Nocedal [3]. Thus, we solve a so-called primal-dual system of linear equations

$$\begin{pmatrix} H_k & \nabla g(x_k)^T & 0 \\ \nabla g(x_k) & -B_k & I \\ 0 & S_k & U_k \end{pmatrix} \begin{pmatrix} d_k^x \\ d_k^u \\ d_k^s \end{pmatrix} + \begin{pmatrix} \nabla f(x_k) + \nabla g(x_k)^T u_k \\ g(x_k) + s_k - C_k u_k \\ S_k u_k - \mu_k e \end{pmatrix} = 0. \quad (5)$$

Here, k denotes the actual iteration index and x_k, s_k, and u_k are the primal and dual iterates. S_k and U_k are positive diagonal matrices containing the vectors s_k and u_k along the diagonal. $B_k, C_k \in \mathbb{R}^{m\times m}$ are positive diagonal regularization matrices and I denotes the identity matrix. Moreover, we introduce $e = (1, ..., 1)^T \in \mathbb{R}^m$. The barrier term μ_k introduced in (4), is internally adapted and depends now on the iteration index k.

A step length $\alpha_k > 0$ along $d_k = (d_k^x, d_k^s, d_k^u)$ is determined to achieve sufficient decrease of a merit function and to get the next iterate

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ u_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ u_k \end{pmatrix} + \alpha_k \begin{pmatrix} d_k^x \\ d_k^s \\ d_k^u \end{pmatrix}, \quad (6)$$

where $0 < \alpha_k \leq 1$ and where $s_{k+1} > 0$ and $u_{k+1} > 0$ must be guaranteed by recursive reduction of α_k. The matrix H_k in (5) is either the Hessian matrix of the Lagrangian function $L(x_k, u_k)$ or a corresponding quasi-Newton matrix updated in each step. Convergence is measured by evaluating the norm of

$$F(x_k, s_k, u_k) := \begin{pmatrix} \nabla f(x_k) + \nabla g(x_k)^T u_k \\ g(x_k) + s_k \\ S_k u_k \end{pmatrix}. \quad (7)$$

In Sect. 2 we give a brief overview of the algorithm, and some numerical results are summarized in Sect. 3.

2 The combined SQP–IPM algorithm

In our situation, we proceed from an SQP algorithm, where the quadratic programming subproblem (3) is solved by an interior-point method for a fixed k, i.e., we replace (3) by

$$\begin{array}{l} d^x \in \mathbb{R}^n, \\ d^s \in \mathbb{R}^m \end{array} : \begin{array}{l} \min \ \frac{1}{2} d^{x\,T} H(x_k, u_k) d^x + \nabla f(x_k)^T d^x - \mu_k \sum_{j=1}^m \log\left(s_j^k + d_j^s\right) \\ g(x_k) + \nabla g(x_k) d^x + s_k + d^s = 0, \end{array} \quad (8)$$

where d^x is the primal variable, and $s_k + d^s$ is considered as slack variable to facilitate the subsequent notation. d^u denotes now the implicitly defined dual variable of (8). Moreover, we use the notation $s_k = (s_1^k, \ldots, s_m^k)^T$ and $d^s = (d_1^s, \ldots, d_m^s)^T$.

The approximate primal and dual solution returned by an IPM solver depends on an internal iteration index l and is denoted by $d^x_{k,l}$, $d^s_{k,l}$, and the multiplier approximation $d^u_{k,l}$, respectively. They are supposed to converge towards a solution of the quadratic program (3). Note that we do not change the Hessian matrix $H(x_k, u_k)$ during the inner iteration.

Thus, the overall algorithm consists of two nested loops identified by two iteration indices k and l. By x_k, s_k, and u_k we denote the outer iterates of primal, slack, and dual variables, respectively, $k = 0, 1, 2, \ldots$. x_0 is a user-provided starting point and $u_0 > 0$, $s_0 > 0$ are usually set internally by the algorithm. The multiplier and slack variables must satisfy $u_k > 0$ and $s_k > 0$ in all subsequent steps. Correspondingly, $d^x_{k,l}$, $d^s_{k,l}$, and $d^u_{k,l}$ are the iterates of the inner cycle with $s_k + d^s_{k,l} > 0$ and $d^u_{k,l} > 0$, $l = 0, 1, 2, \ldots$. To get an SQP method, the inner loop continues until termination at an optimal solution subject to a small tolerance. The outer loop requires an additional line search along the direction obtained by the inner loop, to converge towards a stationary point.

On the other hand, a user may terminate the inner loop at any time, e.g., by setting a small value for the maximum number of iterations. Thus, it is not required to solve the quadratic programming problem exactly. A possible reason could arise when solving very large optimization problems with relatively fast function and gradient evaluations, to avoid time-consuming linear algebra manipulations of the internal QP solver.

Applying again Newton's method to the KKT optimality conditions of (8), we get a primal-dual system of linear equations in each iteration of the inner loop, formulated now in the primal and the dual space analogously to (5),

$$\begin{pmatrix} H_k & \nabla g(x_k)^T & 0 \\ \nabla g(x_k) & -B_{k,l} & I \\ 0 & D^s_{k,l} & D^u_{k,l} \end{pmatrix} \begin{pmatrix} \Delta d^x_{k,l} \\ \Delta d^u_{k,l} \\ \Delta d^s_{k,l} \end{pmatrix} + \begin{pmatrix} H_k d^x_{k,l} + \nabla f(x_k) + \nabla g(x_k)^T d^u_{k,l} \\ g(x_k) + \nabla g(x_k) d^x_{k,l} + s_k + d^s_{k,l} - C_{k,l} d^u_{k,l} \\ D^s_{k,l} d^u_{k,l} - \mu_{k,l} e \end{pmatrix} = 0. \tag{9}$$

Here, k denotes the outer iteration index, l an inner iteration index, and x_k, s_k, and u_k are the outer iterates. $D^s_{k,l}$, and $D^u_{k,l}$ are positive diagonal matrices containing the vectors $s_k + d^s_{k,l}$ and $d^u_{k,l}$ along the diagonal. $B_{k,l}, C_{k,l} \in I\!R^{m\times m}$ are positive diagonal regularization matrices. Their choice depends on the used merit function to get a descent direction. For the l_2-merit function (16) and the flexible penalty function (18), for example, we adapt the regularization of Chen and Golfarb [5] to our SQP–IPM algorithm,

$$B_{k,l} = C_{k,l} = \frac{\left\| g(x_k) + \nabla g(x_k) d^x_{k,l} + s_k + d^s_{k,l} \right\|_2}{r_k} I. \tag{10}$$

r_k is a penalty parameter updated in the outer cycle to guarantee descent of a merit function. The corresponding update formulae depend on the merit function chosen, and are found in the references.

The barrier term $\mu_{k,l}$ introduced in (9) is internally adapted and depends on the iteration indices k and l. Convergence is obtained if the right-hand side of (9) is below a tolerance $\varepsilon_k > 0$.

After solving (9), we get new iterates

$$\begin{aligned} d^x_{k,l+1} &= d^x_{k,l} + \alpha_{k,l}\Delta d^x_{k,l}, \\ d^s_{k,l+1} &= d^s_{k,l} + \alpha_{k,l}\Delta d^s_{k,l}, \\ d^u_{k,l+1} &= d^u_{k,l} + \alpha_{k,l}\Delta d^u_{k,l}, \end{aligned} \tag{11}$$

where $\alpha_{k,l} \in (0, 1]$ is a step length parameter. To simplify the analysis, we do not distinguish between a primal and a dual step length. By applying the fraction to the boundary rule, we get an $\alpha_{k,l}$ such that the inner iterates satisfy

$$\begin{aligned} d^u_{k,l+1} &\geq (1-\tau)d^u_{k,l}, \\ d^s_{k,l+1} &\geq (1-\tau)d^s_{k,l}, \end{aligned} \tag{12}$$

with, e.g., $\tau = 0.995$.

However, the step length might be still too long and is reduced further to guarantee the descent of a merit function

$$\tilde{\Phi}_{\mu,r}\left(x, s, u, d^x, d^s, d^u\right), \tag{13}$$

where μ is a barrier parameter and r is a penalty parameter which must be carefully chosen to guarantee a sufficient descent property, i.e., at least

$$\begin{aligned} &\tilde{\Phi}_{\mu_{k,l},r_k}\left(x_k, s_k, u_k, d^x_{k,l+1}, d^s_{k,l+1}, d^u_{k,l+1}\right) \\ &\quad \leq \tilde{\Phi}_{\mu_{k,l},r_k}\left(x_k, s_k, u_k, d^x_{k,l}, d^s_{k,l}, d^u_{k,l}\right). \end{aligned} \tag{14}$$

The merit function $\tilde{\Phi}_{\mu,r}$ is closely related to the merit function one has to apply in the outer cycle. Here, the step length parameter α_k is adapted such that a sufficient descent property subject to a merit function $\Phi_{\mu,r}(x, s, u)$ is obtained, i.e., that we are able to find a penalty parameter r_k satisfying (14) and

$$\begin{aligned} &\Phi_{\mu_k,r_k}\left(x_k + \alpha_k d^x_k, s_k + \alpha_k d^s_k, u_k + \alpha_k d^u_k\right) \\ &\quad \leq \Phi_{\mu_k,r_k}\left(x_k, s_k, u_k\right) + \nu\, \alpha_k \nabla_{d_k} \Phi_{\mu_k,r_k}\left(x_k, s_k, u_k\right) \end{aligned} \tag{15}$$

where $0 < \alpha_k \leq 1$ is a sufficiently small stepsize and $\nu > 0$ is a given constant. Note that the inner product on the right-hand side of the inequality is always negative. ∇_{d_k} denotes the the directional derivative along (d^x_k, d^s_k, d^u_k).

To give an example, we consider the the l_2-merit function

$$\Phi_{\mu,r}(x, s, u) := f(x) - \mu \sum_{i=1}^{m} \log s_i + r\, \|g(x) + s\|_2 \tag{16}$$

with $\mu, r \in \mathbb{R}$, see, e.g., Chen and Golfarb [5], and neglect iteration indices for a moment. By replacing $f(x)$ by $\frac{1}{2}d^T Hd + \nabla f(x)^T d$ and $g(x)$ by $g(x) + \nabla g(x)d$ and by using $s + d_s > 0$ as slack variable for (3), we obtain

$$\tilde{\Phi}_{\mu,r}\left(x, s, u, d^x, d^s, d^u\right) = \nabla f(x)^T d^x + \frac{1}{2} d^{xT} H d^x - \mu \sum_{j=1}^{m} \log\left(s_j + d_j^s\right) + r \left\| g(x) + \nabla g(x) d^x + s + d^s \right\|_2, \tag{17}$$

i.e., its counterpart used for solving the quadratic programming subproblem. Another possible merit function is the so-called flexible penalty function of Curtis and Nocedal [6],

$$\Phi_{\mu,r}(x, s, u) = f(x) - \mu \sum_{i=1}^{m} \log s_i + \frac{\rho_1 + \rho_2}{2}\rho_3 + \min \left\{ \begin{matrix} \rho_1(\|g(x) + s\|_2 - \rho_3) \\ \rho_2(\|g(x) + s\|_2 - \rho_3) \end{matrix} \right\}, \tag{18}$$

with $r = (\rho_1, \rho_2, \rho_3)$, $\rho_1 \leq \rho_2$, and $\rho_3 = \|g(x + s)\|_2$.

For $\rho_1 = \rho_2 = r$, (18) and (16) are equivalent. Similar to (17), the counterpart for solving the quadratic programming subproblem is derived. Note that both merit functions do not depend on the multiplier vector $u \in \mathbb{R}^m$ in contrast to, e.g., the augmented Lagrangian merit function used by Schittkowski [21,22], which has also been implemented.

The algorithm is summarized now as follows:

Algorithm 1 Choose starting values x_0, u_0, and s_0 with $u_0 > 0$ and $s_0 > 0$, $\mu_{0,0} > 0$, $r_0 > 0$ and some internal constants. For $k := 0, 1, 2, \ldots$

1. Evaluate function and gradient values at x_k, i.e., $f(x_k)$, $g(x_k, \nabla f(x_k)$, and $\nabla g(x_k)$,
2. Check stopping criteria based on the KKT conditions (7). If satisfied, then return.
3. Compute one or more penalty parameters r_k depending on the merit function under consideration.
4. Choose starting values $d_{k,0}^x$, $d_{k,0}^s > 0$, and $d_{k,0}^u > 0$.
5. For $l := 0, 1, 2, \ldots, l_{max}$ do.
   (a) Determine a barrier parameter $\mu_{k,l}$ and suitable scaling matrices $B_{k,l}$ and $C_{k,l}$, e.g., by (10).
   (b) Solve the primal-dual system of linear equations (9) and determine a step length parameter $\alpha_{k,l} \in (0, 1]$ which satisfies (12) and (14).
   (c) Compute new internal iterates $d_{k,l+1}^x$, $d_{k,l+1}^s$, and $d_{k,l+1}^u$ by (11).
   (d) If the termination criteria for the QP (8) are satisfied, i.e., either the norm of the right-hand side of (9) is sufficiently small or $l = l_{max}$, let $\mu_k := \mu_{k,l}$, $d_k^x := d_{k,l+1}^x$, $d_k^s := d_{k,l+1}^s$, and $d_k^u := d_{k,l+1}^u$ and break the for-loop.
6. Find a step length α_k such that the sufficient decrease property (15) is satisfied, e.g., by successive reduction of $\alpha_k = 1$. If necessary, compute new function values.
7. Set $x_{k+1} := x_k + \alpha_k d_k^x$, $s_{k+1} := s_k + \alpha_k d_k^s$, $u_{k+1} := u_k + \alpha_k d_k^u$ and go to step 1.

Here, $l_{max} > 0$ is a given maximum number of iterations of the inner cycle. In principal, $l_{max} = 1$ leads to an IPM and a very large l_{max}, say 100, to an SQP method.

The larger the number of variables n is, the smaller l_{max} should be chosen. If, on the other hand, function evaluations are very costly or highly nonlinear, is recommended to use a higher number of sub-iterations.

The primal and dual stepsize parameters are always greater than zero and less or equal to one. Note that the feasibility condition (12) and the sufficient descent properties (14) and (15) are always satisfied for a sufficiently small stepsize due to the specific choice of the merit function, the barrier and the penalty parameter, and especially the structure of the primal-dual system (9). This can be achieved, e.g., by successive reduction of α_k until the corresponding inequalities are satisfied.

The size of the primal-dual system (9) can be reduced by eliminating $\Delta d^s_{k,l}$ to get a smaller reduced KKT system

$$\begin{pmatrix} H_k & \nabla g(x_k)^T \\ \nabla g(x_k) & -D^s_{k,l}\left(D^u_{k,l}\right)^{-1} - B_{k,l} \end{pmatrix} \begin{pmatrix} \Delta d^x_{k,l} \\ \Delta d^u_{k,l} \end{pmatrix} + \begin{pmatrix} H_k d^x_{k,l} + \nabla f(x_k) + \nabla g(x_k)^T d^u_{k,l} \\ g(x_k) + \nabla g(x_k) d^x_{k,l} + \mu_{k,l}\left(D^u_{k,l}\right)^{-1} e - C_{k,l} d^u_{k,l} \end{pmatrix} = 0 \tag{19}$$

for determining $\Delta d^x_{k,l}$ and $\Delta d^u_{k,l}$. There are several strategies for updating the barrier parameter $\mu_{k,l}$, e.g., by the Mehrota predictor-corrector method developed originally for linear programming, see Nocedal et al. [18] for the nonlinear programming formulas. In our tests however, we leave the barrier parameter constant in the inner iterations, i.e. $\mu_{k,l+1} = \mu_{k,l}$ for all $k, l \in \mathbb{N}$. In the outer iterations, we set $\mu_{k+1,0} = 0.1\mu_{k,0}$ whenever the error of the KKT conditions is less than $5\mu_{k,0}$.

The matrix H_k in (9) or (19), respectively, could be the Hessian matrix of the Lagrangian function (2), if available. However, to satisfy the sufficient descent properties discussed before and to allow an efficient solution of the system of linear equations (9), $H_k = \nabla_{xx} L(x_k, u_k)$ is modified by adding positive values to all entries along the diagonal, until H_k is positive definite.

Alternatively, it is possible to replace the Hessian matrix of the Lagrangian function by a quasi-Newton matrix. Since, however, standard update methods lead to a fill-in, we apply a limited memory BFGS update, see e.g., Liu and Nocedal [8] or Waltz et al. [27]. The idea is to store only the last p pairs of vectors $\nabla_x L(x_{k+1-i}, u_{k-i}) - \nabla_x L(x_{k-i}, u_{k-i})$ and $x_{k+1-i} - x_{k-i}$ for $i = 0, \ldots, p-1$ with $0 < p \ll n$. These pairs of vectors are used to implicitly construct the matrix at x_{k+1} and u_{k+1}. Instead of storing $O(n^2)$ double precision numbers for a full update, one has to keep only $O(pn)$ numbers in memory.

To illustrate limited memory BFGS updates in short, we omit the iteration index k for simplicity. Now, the matrix has the form $H = \xi I + NMN^T$, where $\xi > 0$ is a scaling factor, N is a $n \times 2p$ matrix, and M is a $2p \times 2p$ matrix. M and N are directly computed from the p stored pairs of vectors and ξ. To solve the linear system of equations (19) efficiently for different right-hand sides, we write the inverse of the matrix in (19) in a more tractable form

$$\left[A + BC^T\right]^{-1} = \left[\begin{pmatrix} \xi I & \nabla g(x)^T \\ \nabla g(x) & -SU^{-1} \end{pmatrix} + \begin{pmatrix} N \\ 0 \end{pmatrix} \begin{pmatrix} MN^T & 0 \end{pmatrix}\right]^{-1}$$
$$= A^{-1} - A^{-1}B(\underbrace{I + C^T A^{-1} B}_{\in \mathbb{R}^{2p \times 2p}})^{-1} C^T A^{-1} \tag{20}$$

by the Sherman-Morrison-Woodbury formula with

$$A := \begin{pmatrix} \xi I & \nabla g(x)^T \\ \nabla g(x) & -SU^{-1} \end{pmatrix}, \quad B := \begin{pmatrix} N \\ 0 \end{pmatrix}, \quad C := \begin{pmatrix} MN^T & 0 \end{pmatrix}.$$

Instead of solving (19), we only have to solve the system $Cz = b$ several times with different right hand sides. Matrix $I + V^T C^{-1} U$ is only of size $2p \times 2p$ and can be inverted at negligible costs.

3 Numerical results

The combined IPM/SQP algorithm outlined in the previous section has been implemented in form of a Fortran subroutine called NLPIP, see Sachsenberg and Schittkowski [19]. The flexible merit function (16) is applied together with the regularization matrices (10). We solve all test problems by the same set of tolerances and parameters, which are internally stored as default values. The number of recursive LM-Quasi-Newton updates is $p = 7$ unless defined separately. The KKT-system (5) or any derived system of linear equations is solved either by LAPACK [1] in case of the small test problems and by PARDISO otherwise, see e.g., Schenk and Gärtner [20]. The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 11.0, 64 bit, under Windows 7 and Intel(R) Core(TM) i7-2720QM CPU, 2.2 GHz, with 8 GB RAM.

3.1 Elliptic optimal control problems with control and state constraints

Maurer and Mittelmann [16,17] published numerical results to compare some large-scale optimization codes on a certain class of test problems obtained by discretizing semi-linear elliptic optimal control problems with control and state constraints. The two-dimensional elliptic equations are discretized by a scalable rectangular grid of size $N = 100$ in x- and y-direction, where the following abbreviations are used in Table 1:

problem	test problem identifier,
n	number of optimization variables ,
m_e	number of equality constraints,
n_{func}	number of function evaluations,
n_{grad}	number of gradient evaluations,
$f(x^\star)$	objective function value at termination point $x^\star$
time	total CPU time in seconds.

Table 1 Test results for semilinear elliptic control problems

Problem	n	m_e	n_{func}	n_{grad}	$f(x^\star)$	Time		
EX 1	10,197	9801	13	13	0.19652520	0.38E−14	0.25E−08	8.1
EX 2	10,197	9801	21	20	0.09669524	0.38E−14	0.88E−08	10.4
EX 3	10,197	9801	12	12	0.32100999	0.33E−14	0.11E−08	7.3
EX 4	10,197	9801	12	12	0.24917886	0.38E−14	0.25E−08	6.7
EX 5	10,593	10,197	15	15	0.55224625	0.23E−09	0.11E−08	8.7
EX 6	10,593	10,197	21	21	0.01507906	0.29E−09	0.10E−08	11.5
EX 7	10,593	10,197	206	80	0.28462160	0.30E−14	0.70E−08	33.9
EX 8	10,593	10,197	208	84	0.21964514	0.31E−14	0.10E−08	30.4
EX 9	19,602	9801	15	15	0.06216417	0.95E−14	0.11E−08	11.0
EX 10	19,602	9801	16	16	0.05645717	0.23E−13	0.15E−08	16.5
EX 11	19,602	9801	15	15	0.11026724	0.66E−14	0.10E−08	9.0
EX 12	19,998	10,197	20	20	0.07806694	0.13E−12	0.13E−08	11.8
EX 13	19,998	10,197	24	24	0.05267357	0.75E−13	0.23E−08	22.0

Table 2 Average test results for 306 Hock-Schittkowski problems

Code	n_{succ}	n_{func}	n_{grad}	Time
NLPIP ($p = 7$)	301	81	28	1.4
NLPIP ($p = 70$)	303	22	17	2.3
NLPQLP	305	24	17	0.6

Problems EX1 to EX8 correspond to examples 5.1–5.8 of Maurer and Mittelmann [16] and problems EX9–EX13 to examples 1–5 of Maurer and Mittelmann [17]. All optimal objective function values shown in Table 1 coincide to those presented by the authors of the papers mentioned, besides of some differences caused by different discretization accuracy.

One function evaluation consists of the computation of one objective function value and all constraint function values. Derivatives are available in analytical form and termination accuracy is set to 10^{-8}. The number of internal iterations is set to $l_{max} = 1$. We observe rapid convergence within a quite low number of iterations which is not effected by doubling the number of variables for EX9 to EX13.

3.2 Small and dense HS-problems

Moreover, we evaluate the performance of NLPIP on the set of 306 small-scale, but highly nonlinear test problems of Hock and Schittkowski [15,23], and compare the results to the SQP solver NLPQLP, a dense implementation of an SQP-method, see Schittkowski [21,22,24]. The latter reference contains a detailed description of the test environment.

Two-sided differences are used for approximating derivatives and termination accuracy is set to 10^{-5}. In Table 2 we present some results for $p = 7$ and $p = 70$. n_{succ}

Table 3 Test results for CUTEr-problems

Problem	n	m_e	n_{func}	n_{grad}	$f(x^\star)$	$g^-(x^\star)$	Time
GILBERT	5000	1	65	32	0.2459468E+04	0.72E−07	2.4
BRAINPC0	6907	6900	159	57	0.3670417E+00	0.31E−09	12.4
BRAINPC1	6907	6900	267	73	0.4175864E−03	0.11E−07	34.0
BRAINPC2	13,807	13,800	128	50	0.4421012E+00	0.10E−07	33.6
BRAINPC3	6907	6900	452	155	0.3637082E+00	0.84E−10	36.4
BRAINPC4	6907	6900	309	82	0.4068723E+00	0.18E−08	17.9
BRAINPC5	6907	6900	209	61	0.3847010E+00	0.15E−09	13.0
BRAINPC6	6907	6900	214	79	0.3750098E+00	0.11E−09	19.2
BRAINPC7	6907	6900	210	70	0.3965791E+00	0.70E−10	14.7
BRAINPC8	6907	6900	549	170	0.7875434E−03	0.17E−07	37.5
BRAINPC9	6907	6900	287	94	0.3533461E+00	0.17E−07	20.2
CAR2	5999	3996	293	156	0.2666083E+01	0.76E−12	27.6
CLNLBEAM	60,003	40,000	27	14	0.3500000E+03	0.25E−19	12.6
CORKSCRW	45,006	30,000	277	263	0.9809597E+02	0.14E−12	132.7
COSHFUN	6001	0	1,943	500	-0.7732327E+00	0.00E+00	36.5
DRUGDIS	6004	4000	148	121	0.4277756E+01	0.44E−10	14.7
DTOC1NA	5998	3996	14	14	0.4138867E+01	0.25E−11	0.9
DTOC1NB	5998	3996	15	14	0.7138849E+01	0.58E−08	0.9
DTOC1NC	5998	3996	15	14	0.3519934E+02	0.53E−10	0.9
DTOC1ND	5998	3996	15	15	0.4760303E+02	0.15E−08	1.0
DTOC2	5998	3996	238	118	0.5086610E+00	0.72E−07	10.9
DTOC5	9999	4999	28	14	0.1535111E+01	0.72E−10	1.2
DTOC6	10,001	5000	63	49	0.1348480E+06	0.87E−07	5.3
JUNKTURN	10,010	7000	1,279	500	0.1719764E−02	0.11E−04	87.4
OPTMASS	60,010	40,004	51	22	-0.1853756E−02	0.15E−15	19.3
ORTHRDM2	8003	4000	29	15	0.3110153E+03	0.10E−08	2.2
ORTHRDS2	5003	2500	46	32	0.7624654E+03	0.83E−07	2.7
ORTHREGA	8197	4096	234	108	0.2264784E+05	0.16E−09	23.6
ORTHREGC	5005	2500	185	78	0.9481285E+02	0.19E−08	8.1
ORTHREGD	5003	2500	29	16	0.7620643E+03	0.33E−10	1.2
ORTHREGE	7506	5000	150	65	0.1087286E+04	0.79E−09	13.5
ORTHRGDM	10,003	5000	31	18	0.1513802E+04	0.34E−09	3.8
ORTHRGDS	5003	2500	55	33	0.7620643E+03	0.23E−08	2.7
READING5	5001	5000	46	28	0.0000000E+00	0.00E+00	1.6
SVANBERG	50,000	0	188	188	0.8362382E+05	0.90E−08	256.0

is the number of successful solutions satisfying KKT conditions subject to a predetermined tolerance of 10^{-8}, and approaching the known optimal solution subject to a relative error of one percent. n_{func} is the number of function evaluations, and n_{grad} is the number of derivative evaluations, in both cases of objective function and all constraint functions simultaneously. Since many of the test problems are highly non-

linear, the number of internal iterations is set to $l_{max} = 100$. In other words, we apply an SQP method in this case.

Called with a larger number of BFGS-updates, NLPIP needs about the same number of iterations and function evaluations compared to NLPQLP. Since the quadratic programming problem is iteratively solved, average computation times are higher than those of NLPQLP, especially due to a large number of limited-memory updates.

3.3 CUTEr collection

A large number of test problems for nonlinear programming has been collected and programmed by Gould, Orban, and Toint [13]. The library is widely used for developing and testing optimization programs, and consists of small- and large-scale problems. Derivatives are available in analytical form. For our purposes, we select 35 test problems with 5000 or more variables. The problems are identified by their internal name. They only possess equality constraints with five exceptions, CAR2 ($m = 4996$), CORKSCRW ($m = 35{,}000$), COSHFUN ($m = 2000$), OPTMASS ($m = 50{,}005$), and SVANBERG ($m = 50{,}000$).

Numerical results are listed in Table 3 for termination accuracy 10^{-8} and $l_{max} = 1$. In two cases, the maximum number of iterations (500), is reached. $g^-(x^\star)$ shows the constraint violation on termination. During two other test runs, the code stops because of more than 20 feasible iterates without reducing the objective function value. Only one iteration is allowed for the solving the quadratic subproblems, i.e., we apply an interior point method.

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. Society for industrial and applied mathematics, Philadelphia (1999)
2. Boggs, P.T., Tolle, J.W.: Sequential quadratic programming. Acta Numer. **4**, 1–51 (1995)
3. Byrd, R.H., Gilbert, J.C., Nocedal, J.: A trust region method based on interior point techniques for nonlinear programming. Math. Progr. **89**, 149–185 (2000)
4. Cafieri, S., D'Apuzzo, M., De Simone, V., di Serafino, D.: On the iterative solution of KKT systems in potential reduction software for large scale quadratic problems. Comput. Optim. Appl. **38**, 27–45 (2007)
5. Chen, L., Goldfarb, D.: An interior-point piecewise linear penalty method for nonlinear programming. Math. Progr. **10**, 1–50 (2009)
6. Curtis, F.E., Nocedal, J.: Flexible penalty functions for nonlinear constrained optimization. J. Numer. Anal. **28**, 335–351 (2008)
7. D'Apuzzo, M., De Simone, V., di Serafino, D.: On mutual impact of numerical linear algebra and large-scale optimization with focus on interior point methods. Comput. Optim. Appl. **45**(2), 283–310 (2010)
8. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large-scale optimization. Math. Progr. **45**, 503–528 (1989)
9. Fletcher, R.: Practical methods of optimization. Wiley, Chichester (1987)
10. Gill, P.E., Murray, W., Wright, M.: Practical optimization. Academic Press, London (1982)
11. Gondzio, J.: Interior point methods 25 years later. Eur. J. Oper. Res. **218**, 587–601 (2012)
12. Gould, N.I.M., Toint, Ph.L.: SQP methods for large-scale nonlinear programming. In: Proceedings of the 19th IFIP TC7 Conference on System Modelling and Optimization: Methods, Theory and Applications, pp 149–178 (1999)

13. Gould, N.I.M., Orban, D., Toint, Ph.L.: General CUTEr documentation, CERFACS Technical Report TR/PA/02/13 (2005)
14. Griva, I., Shanno, D.F., Vanderbei, R.J., Benson, H.Y.: Global convergence of a primal-dual interior-point method for nonlinear programming. Algorithmic Oper. Res. **3**, 12–19 (2008)
15. Hock, W., Schittkowski, K.: A comparative performance evaluation of 27 nonlinear programming codes. Computing **30**, 335–358 (1983)
16. Maurer, H., Mittelmann, H.D.: Optimization techniques for solving elliptic control problems with control and state constraints, part 1: boundary control. Comput. Optim. Appl. **16**, 29–55 (2000)
17. Maurer, H., Mittelmann, H.D.: Optimization techniques for solving elliptic control problems with control and state constraints, part 2: distributed control. Comput. Optim. Appl. **18**, 141–160 (2001)
18. Nocedal, J., Wächter, A., Waltz, R.A.: Adaptive barrier strategies for nonlinear interior methods. SIAM J. Optim. **19**, 1674–1693 (2009)
19. Sachsenberg, B., Schittkowski, K.: NLPIP: a Fortran implementation of an SQP interior point method for solving large-scale nonlinear optimization problems—user's guide. Department of Computer Science, University of Bayreuth, Report (2014)
20. Schenk, O., Gärtner, K.: On fast factorizing pivoting methods for symmetric indefinite systems. Electron. Trans. Numer. Anal. **23**, 158–179 (2006)
21. Schittkowski, K.: On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction. Optimization **14**, 197–216 (1983)
22. Schittkowski, K.: NLPQL: a Fortran subroutine solving constrained nonlinear rogramming problems. Annals of Operations Research **5**, 485–500 (1986)
23. Schittkowski, K.: More test examples for nonlinear programming, lecture notes in economics and mathematical systems, vol. 182. Springer
24. Schittkowski, K.: 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 (2009)
25. Sun, W.Y., Yuan, Y.: Optimization theory and methods: nonlinear programming. Springer, New York (2006)
26. Vanderbei, R.J.: LOQO: An interior point code for quadratic programming. Optim. Methods Softw. **11**, 451–484 (1999)
27. Waltz, R.A., Morales, J.L., Nocedal, J., Orban, D.: An interior algorithm for nonlinear optimization that combines line search and trust region steps. Math. Program. **107**, 391–408 (2006)