# NLPIP: A Fortran Implementation of an SQP-IPM Algorithm for Solving Large-Scale Nonlinear Optimization Problems - User's Guide, Version 2.0 -

**Authors** : **B. Sachsenberg, K. Schittkowski**

*Contact* : K. Schittkowski
Siedlerstr. 3
95488 Eckersdorf
Germany

*Phone*: (+49) 921 32887

*E-mail*: klaus@schittkowski.de

*Web*: www.klaus-schittkowski.de

*Date*: October, 2013

**Abstract**

The Fortran subroutine NLPIP is designed to solve smooth and large-scale nonlinear optimization problems. The underlying algorithm is based on an SQP method, where the quadratic programming subproblem is solved by a primal-dual interior point method. A special feature of the algorithm is that the quadratic programming subproblem does not need to get exactly solved. If the number of iterations is set to one, we obtain a standard interior point method. 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. The Jacobian of the constraints or, if available, the Hessian of the Lagrangian function should be sparse. All necessary operations by which these matrices are accessed through a so-called primal-dual system of equations, are provided by a separate code called LINSLV, which is to be implemented by the user. Interfaces for some existing linear solvers are available. Numerical results are included for the small and dense Hock-Schittkowski problems, for large semi-linear elliptic control problems after a suitable discretization, and for the cute-r test problem collection.

Keywords: large-scale optimization, nonlinear programming, IPM, interior point method, SQP, sequential quadratic programming method, merit function, non-monotone line search, numerical algorithm, Fortran code

# 1 Introduction

We consider the nonlinear programming problem to minimize an objective function under nonlinear equality and inequality constraints,

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

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 twice continuously differentiable on the whole $I\!R^n$.

To illustrate the underlying mathematical algorithm, we omit upper and lower bounds and equality constraints to get, in a somewhat simpler notation, a problem of the form

$$x \in I\!R^n : \begin{array}{ll} \min & f(x) \\ g(x) \leq 0 \ , & \end{array} \tag{2}$$

where $g(x) = (g_1(x), \ldots, g_m(x))^T$.

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

$$L(x, u) := f(x) + u^T g(x) \tag{3}$$

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

$$d \in I\!R^n : \begin{array}{ll} \min & \frac{1}{2} d^T H(x_k, u_k) d + \nabla f(x_k)^T d \\ g(x_k) + \nabla g(x_k) d \leq 0 & \end{array} \tag{4}$$

is then solved by an interior point solver. Here, the pair $(x_k, u_k)$ denotes the current iterate in the primal-dual space, $H(x_k, u_k)$ denotes the Hessian of the Lagrangian function (3), 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 $x_k \in I\!R^n$ the primal and $u_k \in I\!R^m$ 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.

Sequential quadratic programming methods became popular during the late 70's. Since then, there are numerous modifications and extensions have been published on SQP methods. Nice review papers are given by Boggs and Tolle [2] and Gould and Toint [9]. All optimization textbooks have chapters on SQP methods, see, for example, see Fletcher [7], Gill, Murray and

Wright [8], and Sun and Yuan [22]. A widely used Fortran code has been implemented by Schittkowski [21] and is called NLPQLP.

An alternative approach is the interior point method (IPM) developed in the 90's, see, e.g., Griva et al. [11], 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].

The underlying strategy consists of replacing the constrained optimization problem (2) by a simpler one without inequality constraints,

$$x \in I\!R^n, s \in I\!R^m : \begin{array}{ll} \min & f(x) - \mu \sum_{j=1}^m log(s_j) \\ & g(x) + s = 0 \quad . \end{array} \tag{5}$$

Here, $s > 0$ denotes a vector of $m$ slack variables, where the positivity has to be guaranteed separately. The smaller the so-called barrier term $\mu$ 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 $u \in I\!R^m$, the multipliers of the equality constraints of (5). Since, however, these multiplier approximations are also used to approximate the multipliers of the inequality constraints of (2), we also require that $u > 0$ throughout the algorithm.

By constructing a sequence of systems of linear equations, where each one is called a primal-dual system, a series of iterates toward the solution is approximated, see, for example, Byrd, Gilbert, and Nocedal [3]. The primal-dual system is given by

$$\begin{pmatrix} H_k & \nabla g(x_k)^T & 0 \\ \nabla g(x_k) & -B_k & I \\ 0 & S_k & U_k \end{pmatrix} \begin{pmatrix} d_k^x \\ d_k^u \\ d_k^s \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla g(x_k)^T u_k \\ g(x_k) + s_k - C_k u_k \\ S_k u_k - \mu_k e \end{pmatrix} \quad . \tag{6}$$

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. $B_k, C_k \in I\!R^{m \times m}$ are positive diagonal regularization matrices and $I$ denotes the identity matrix. Moreover, we introduce $e = (1, ..., 1)^T \in I\!R^m$. The barrier term $\mu_k$ introduced in (6), is internally adapted and depends 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

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ u_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ u_k \end{pmatrix} + \alpha_k \begin{pmatrix} d_k^x \\ d_k^s \\ d_k^u \end{pmatrix} \tag{7}$$

where $0 < \alpha_k \leq 1$ and where $s_{k+1} > 0$ and $u_{k+1} > 0$ must be guaranteed. The matrix $H_k$ in (6) is either the Hessian matrix of the Lagrangian function $L(x_k, u_k)$ or a corresponding quasi-Newton matrix updated in each step.

In Section 2 we give a brief overview of the algorithm. The corresponding Fortran subroutine is documented in Section 3. Some examples show how to implement an optimization problem, see Section 4. Finally, we present numerical results in Section 5.

## 2 The Algorithm

In our situation, only the quadratic programming subproblem (4) is solved by an interior-point method, i.e., we replace (4) by

$$d^x \in I\!R^n, d^s \in I\!R^m : \begin{array}{l} \min \ \frac{1}{2} d^{xT} H(x_k, u_k) d^x + \nabla f(x_k)^T d^x - \mu \sum_{j=1}^m log(d_j^s) \\ g(x_k) + \nabla g(x_k) d^x + d^s = 0 \ . \end{array} \tag{8}$$

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 $d^u_{k,l}$. They serve as search directions for converging towards a solution of the nonlinear program (2). Note again that we can stop at any iteration $l$ we like, and get a standard IPM by leaving the inner loop after the first step.

Thus, the 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 set by the algorithm. The slack and multiplier variables must satisfy $u_k > 0$ and $s_k > 0$ in all subsequent steps. Correspondingly, $d^x_{k,l}$, $d^s_{k,l} > 0$, and $d^u_{k,l} > 0$ are the iterates of an inner cycle.

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 subproblem exactly. If only one step is performed towards a solution of (4), a standard IPM is obtained. A possible reason could be to solve very large optimization problems with relatively fast function and gradient evaluation times, to avoid time-consuming linear algebra manipulations of the internal QP solver.

The KKT optimality conditions of (8) lead to a primal-dual system of linear equations in each inner loop, formulated now in the primal and the dual space analogously to (6),

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

Here, $k$ denotes the outer iteration index, $l$ an inner iteration index, and $x_k$, $s_k$, and $u_k$ are the outer iterates. $S^d_{k,l}$ and $U^d_{k,l}$ are positive diagonal matrices containing the vectors $d^s_k$ and

$d_k^u$ along the diagonal. $B_{k,l}, C_{k,l} \in I\!R^{m\times m}$ are positive diagonal regularization matrices. The barrier term $\mu_{k,l}$ introduced in (9), is internally adapted and depends now in the iteration indices $k$ and $l$.

By solving (9), we get new iterates

$$\begin{aligned} d^x_{k,l+1} &= d^x_{k,l} + \alpha_{k,l}\Delta d^x_{k,l} \ , \\ d^s_{k,l+1} &= d^s_{k,l} + \alpha_{k,l}\Delta d^s_{k,l} \ , \\ d^u_{k,l+1} &= d^u_{k,l} + \alpha_{k,l}\Delta d^u_{k,l} \ , \end{aligned} \tag{10}$$

where $\alpha_{k,l} \in (0,1]$ is a step length parameter. To simplify the analysis, we only mention that in our implementation we distinguish between a primal and a dual step length. By reducing $\alpha$, we get an $\alpha_{k,l}$ such that the inner iterates satisfy

$$d^u_{k,l+1} = d^u_{k,l} + \alpha_{k,l}\Delta d^u_{k,l} > 0 \ , \quad d^s_{k,l+1} = d^s_{k,l} + \alpha_{k,l}\Delta d^s_{k,l} > 0 \ . \tag{11}$$

However, the search direction might be still too long and is to be reduced further to guarantee the sufficient descent of a merit function

$$\tilde{\Phi}_{\mu,r}(x, s, u, d^x, d^s, d^u) \ , \tag{12}$$

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

$$\tilde{\Phi}_{\mu_{k,l},r_k}(x_k, s_k, u_k, \Delta d^x_{k,l}, \Delta d^s_{k,l}, \Delta d^u_{k,l}) \le \tilde{\Phi}_{\mu_{k,l},r_k}(x_k, s_k, u_k, 0, 0, 0) \ . \tag{13}$$

The merit function $\tilde{\Phi}_{\mu,r}$ is closely related to the merit function one has to apply in the outer cycle, see the example below. In the outer cycle, the step length parameter $\alpha_k$ 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 $r_k$ and, especially, a step size $0 < \alpha_k \le 1$ with

$$\begin{aligned} \Phi_{\mu_k,r_k}(x_k + \alpha_k d^x_k, s_k + \alpha_k d^s_k, u_k + \alpha_k d^u_k) \ &\le \ \Phi_{\mu_k,r_k}(x_k, s_k, u_k) \\ &+ \ \nu\, \alpha_k \nabla\Phi_{\mu_k,r_k}(x_k, s_k, u_k)^T \begin{pmatrix} d^x_k \\ d^s_k \\ d^u_k \end{pmatrix} \end{aligned} \tag{14}$$

where $\nu > 0$ is a given constant. Note that the inner product on the right-hand side of the inequality is always negative.

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

$$\Phi_{\mu,r}(x, s, u) := f(x) - \mu \sum_{i=1}^{m} \log s_i + r\|g(x) + s\|_2 \tag{15}$$

with $\mu, r \in I\!R$, see, e.g., Chen and Golfarb [4], and neglect iteration indices for a moment.. By replacing $f(x)$ by $\frac{1}{2}d^T H d + \nabla f(x)^T d$ and $g(x)$ by $g(x) + \nabla g(x) d$, we obtain

$$\tilde{\Phi}_{\mu,r}(x, s, u, d^x, d^s, d^u) = \nabla f(x)^T d^x + \frac{1}{2} d^{xT} H d^x - \mu \sum_{i=1}^{m} \log(s_i + d_i^s) + r\|g(x) + \nabla g(x) d^x + s + d^s\|_2 \ , \tag{16}$$

i.e., its counterpart used for solving the quadratic programming subproblem, see (12) and (13).

Another merit function is the so-called flexible penalty function of Curtis and Nocedal [5], which is default in NLPIP,

$$\Phi_{\mu,r}(x, s, u) = f(x) - \mu \sum_{i=1}^{m} \log s_i + \frac{r_1 + r_2}{2} r_3 + \min \left\{ \begin{array}{c} r_1(\|g(x) + s\|_2 - r_3), \\ r_2(\|g(x) + s\|_2 - r_3) \end{array} \right\} \tag{17}$$

with $r = (r_1, r_2, r_3)$ and

$$\begin{array}{rcl} r_1 & \leq & r_2 \ , \\ r_3 & = & \|g(x + s\|_2 \ . \end{array}$$

For $r_1 = r_2$, (17) and (16) are equivalent. Similar to (16), the counterpart for solving the quadratic programming subproblem is derived. Note that both merit functions do not depend on the multiplier vector $u \in I\!R^m$ in contrast to the so-called augmented Lagrangian merit functions.

The algorithm can be summarized now as follows:

**Algorithm 2.1** *1. Choose starting values $z_0 = (x_0, u_0, s_0)$ with $u_0 > 0$ and $s_0 > 0$ and some internal constants.*

*2. For $k := 0, 1, 2, \ldots$*

*(a) Check stopping criteria based on the KKT conditions. If satisfied, then return.*

*(b) Choose starting values $d^x_{k,0}$, $d^s_{k,0}$, and $d^u_{k,0}$*

*(c) For $l := 0, 1, 2, \ldots, l_{max}$ do*

*i. Determine a barrier parameter $\mu_{k,l}$ and suitable scaling matrices $B^{k,l}$ and $C^{k,l}$.*

*ii. Solve the primal-dual system of linear equations (9) and determine a step length parameter $\alpha_{k,l} \in (0, 1]$ which satisfy (11) and (13).*

*iii. Compute new internal iterates $d^x_{k,l+1}$, $d^s_{k,l+1}$, and $d^u_{k,l+1}$ by (10).*

*iv. If termination criteria for the QP (4) are satisfied, e.g., either KKT conditions or $l = l_{max}$, let $\mu_k := \mu_{k,l+1}$, $d^x_k := d^x_{k,l+1}$, $d^s_k := d^s_{k,l+1}$, and $d^u_k := d^u_{k,l+1}$ and break for-loop.*

*(d) Find a step length $\alpha_k$ such that the sufficient decrease property (14) is satisfied.*

*(e) Set* $x_{k+1} := x_k + \alpha_k d_k^x$, $s_{k+1} := s_k + \alpha_k d_k^s$, $u_{k+1} := u_k + \alpha_k d_k^u$.

Here, $l_{max} > 0$ is a given maximum number of iterations of the inner cycle. The primal and dual stepsize parameters are always greater than zero and less or equal to one. Note that the feasibility condition (11) and the sufficient descent properties (13) and (14) 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 until the corresponding inequalities are satisfied.

The primal-dual system (9) can be reduced further by exploiting that

$$\Delta d_{k,l}^s = U_{k,l}^{-1}(\mu_{k,l} e - S_{k,l}\Delta d_{k,l}^u) - S_{k,l}\Delta d_{k,l}^u \tag{18}$$

to get a smaller reduced KKT system

$$\begin{pmatrix} H_k & \nabla g(x_k)^T \\ \nabla g(x_k) & -S_{k,l}U_{k,l}^{-1} - B_{k,l} \end{pmatrix} \begin{pmatrix} \Delta d_{k,l}^x \\ \Delta d_{k,l}^u \end{pmatrix} = - \begin{pmatrix} H_k d_{k,l}^x + \nabla f(x_k) + \nabla g(x_k)^T d_{k,l}^u \\ g(x_k) + \nabla g(x_k) d_{k,l}^x + \mu_{k,l} U_{k,l}^{-1} e - C_{k,l} d_{k,l}^u \end{pmatrix} \tag{19}$$

for determining $\Delta d_{k,l}^x$, $\Delta d_{k,l}^u$, and, by (18), $\Delta d_{k,l}^s$. The barrier parameter $\mu_{k,l}$ must be carefully updated, e.g., by the Mehrota predictor-corrector method developed originally for linear programming, see Nocedal et.al [15] for the nonlinear programming formulas.

The matrix $H_k$ in (9) or (19), respectively, could be the Hessian matrix of the Lagrangian function (3), if available. However, to guarantee the sufficient descent properties discussed before and to allow an efficient solution of the system of linear equations (9), $H_k = \nabla_x^2 L(x_k, u_k)$ has to be positive definite for all $k$. Since this property cannot be satisfied in general, a typical modification is to add positive values to the diagonal, i.e., to let $H_k = \nabla_x^2 L(x_k, u_k) + W_k$, where $W_k$ is a positive diagonal matrix with suitable weights. To solve large-scale problems, it must be assumed in this case that $\nabla_x^2 L(x_k, u_k)$ is sparse.

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, it is possible to apply a limited memory BFGS update, see e.g., Liu and Nocedal [6] or Waltz et.al [24].

The standard BFGS method stores the whole matrix which is updated in every iteration by the rule

$$H_{k+1} := H_k + \frac{a_k a_k{}^T}{b_k{}^T a_k} - \frac{H_k b_k b_k{}^T H_k}{b_k{}^T H_k b_k} \tag{20}$$

with

$$\begin{aligned} a_k &:= \nabla_x L(x_{k+1}, u_{k+1}) - \nabla_x L(x_k, u_{k+1}) \\ b_k &:= x_{k+1} - x_k \end{aligned}$$

Usually, we start with a scaled unit matrix for $H_0$ and stabilize the formula by requiring that

$$a_k^T b_k \geq 0.2\, b_k^T H_k b_k \quad ,$$

see Powell [16] or Schittkowski [19].

The idea of the limited memory BFGS update is to store only the last $p$ pairs of vectors

$$\nabla_x L(x_{k+1-i}, u_{k-i}) - \nabla_x L(x_{k-i}, u_{k-i}), \quad x_{k+1-i} - x_{k-i}$$

for $i = 0, \ldots, p-1$ with $0 < p \ll n$. These pairs of vectors are used to implicitly construct an approximation of the Hessian 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 update in short, we omit the iteration index $k$ for simplicity. Now, the BFGS matrix has the form

$$H = \xi I + NMN^T \quad ,$$

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 $\xi$.

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

$$\begin{aligned} & \left[ \begin{pmatrix} \xi I & \nabla g(x)^T \\ \nabla g(x) & -SU^{-1} \end{pmatrix} + \begin{pmatrix} N \\ 0 \end{pmatrix} \begin{pmatrix} MN^T & 0 \end{pmatrix} \right]^{-1} \qquad (21) \\ =: \; & [C + UV^T]^{-1} \\ = \; & C^{-1} - C^{-1}U(\underbrace{I + V^T C^{-1} U}_{\in \mathbb{R}^{2p \times 2p}})^{-1} V^T C^{-1} \end{aligned}$$

by the Sherman-Morrison-Woodbury formula. Instead of solving (19), we only have to solve the system $Cz = b$ several times with different right hand sides. Moreover, $C$ has exactly the same structure as the matrix $H$ in (19) or, in other words, $H_k = \xi I$ in this case. Matrix $I + V^T C^{-1} U$ is only of size $2p \times 2p$ and can be inverted at negligible costs.

# 3 Program Documentation

## 3.1 NLPIP

NLPIP is implemented in form of a FORTRAN subroutine, where function and gradient values are computed within the calling routine of the user and passed to NLPIP by reverse communication. The following rules apply:

1. Choose starting values for the variables to be optimized, and store them in X.
2. Compute objective and all constraint function values, store them in F and G, respectively.
3. Compute gradients of objective function and all constraints, and store them in DF and DG, respectively.

4. Set IFAIL=0 and execute NLPIP.

5. If NLPIP returns with IFAIL=-1, compute objective function and constraint values for all variable values in X, store them in F and G, and call NLPIP again.

6. If NLPIP terminates with IFAIL=-2, compute gradient values with respect to the variable values in X, store them in DF and DG, and call NLPIP again.

7. If NLPIP terminates with IFAIL=0, the internal stopping criteria are satisfied. WORK(1) passes the norm of the KKT vector to the calling program and WORK(2) the maximum constraint violation. In case of IFAIL>0, an error occurred.

The user also has to provide a Function called LINSLV which is responsible for various operations concerning the Jacobian of the constraints and for solving the reduced KKT system , see Section 3.2 for details.

Note that NLPIP calls some LAPACK [1] routines and must be linked to this library.

**Usage:**

```
      CALL  NLPQLP(      M,      ME,     LDG,       N,       X,
     /                   F,       G,      DF,      DG,       Y,
     /                  SL,      XL,      XU,  ACTIVE,       P,
     /                 ACC,   ACCQP,  MAXFUN,   MAXIT,   MNFS,
     /              IPRINT,    IOUT,  IPARAM,   IFAIL,   STEPP,
     /                WORK,   LWORK,   IWORK,  LIWORK           )
```

**Parameter Definition:**

| | |
|---|---|
| M : | Number of all constraints without bounds of variables. |
| ME : | Number of equality constraints. |
| LDG : | Number of non-zero elements of the Jacobian of constraints, DG. |
| N : | Number of optimization variables. |
| X(N) : | When calling NLPIP the first time, X has to contain starting values for the optimal solution. On return, X is replaced by the current iterate. In the driving program, the dimension of X has to be at least N. |

| | |
|---|---|
| F | Objective function value at X. |
| G(M) | Constraint function values at X. In the driving program, the dimension has to be at least M. |
| DF(N) | Gradient of the objective function at X. |
| DG(LDG) | Jacobian of the constraint functions at X. Only non-zero elements are to be passed to subroutine LINSLV. |
| Y(M+2N) | On return, Y contains the multipliers with respect to the current iterate stored in X. The first M entries contain the multipliers of the M constraints given by G, the subsequent N entries contain the multipliers of the lower bounds, and the last N entries the multipliers of the upper bounds. |
| SL(M+2N) | On return, SL contains the slack variables with respect to the current iterate stored in X. The first ME entries are undefined. The subsequent M-ME entries contain the slacks of the inequality constraints as given in G. |
| XL(N) | Lower bounds for X. |
| XU(N) | Upper bounds for X, where a lower bound has to be less than an upper bound. |
| ACTIVE(M+2N) | Logical array to indicate constraints active at the current iterate. The first M entries correspond to the M constraints given in G, the subsequent N entries correspond to the lower bounds and the last N entries to the upper bounds. |
| P | Maximum number of pairs of vectors stored for limited memory BFGS updates. Typical values are in the range from 3 to 20. |
| ACC | The user has to specify the desired termination accuracy. ACC should not be much smaller than the accuracy by which gradients are computed. |
| ACCQP | Accuracy for solving the internal quadratic programming problem. ACCQP should be smaller than ACC and greater than the machine precision. |
| MAXFUN : | The integer variable defines an upper bound for the number of function calls during the line search (e.g. 20). MAXFUN must not exceed 50. |

MAXIT : Maximum number of outer iterations, where one iteration corresponds to one formulation and solution of the quadratic programming subproblem, or, alternatively, one evaluation of gradients (e.g. 100).

MNFS : Maximum number of feasible steps without improvements, where the relative change of objective function values and feasibility is measured by ACC. Must be greater than 1.

IPRINT : Specification of the desired output level.

0 - no output

1 - initial and final messages

2 - one line displayed for each main iteration

3 - additional messages about QP solution

4 - information about QP iterates and line search

5 - additional warnings

IOUT : Positive integer indicating the desired output unit number, i.e., all write-statements start with 'WRITE(IOUT,... '.

IPARAM(20) : Integer parameter vector of length 20 to control execution. Subsequent positions not specified in advance, must get -100, and missing parameters are reserved for later use. Parameter values might get changed internally, so do not modify them between successive iterations.

IPARAM(1): General algorithmic structure, i.e.,

0 - LM-BFGS updates (default)

1 - Hessian matrix of the Lagrangian provided

IPARAM(2): Merit function

1 - L2 penalty

2 - flexible penalty (default)

IPARAM(3): Starting value for penalty updates of merit function, i.e.,

<>0 - exponent, i.e., 10**IPARAM(3), between -10 and 10

0 - 1.0 (default)

IPARAM(4): Exponent of initial barrier parameter $\mu$, i.e., $\mu$=10**IPARAM(4), where -10 <IPARAM(4)< 2 (default: -3)

IPARAM(5): Update of barrier parameter $\mu$

1 - if $c_{KKT} < 5\mu$, then $\mu \leftarrow 0.1\mu$ (default)

2 - if $c_{KKT} < 5\mu$, then $\mu \leftarrow 0.01\mu$

3 - if $c_{KKT} < 5\mu$, then $\mu \leftarrow \min(0.2c_{KKT}, c_{KKT}^{1.5})$

4 - $\mu \leftarrow \min(0.2c_{KKT}, \mu)$

where $c_{KKT}$ denotes the norm of the KKT vector

IPARAM(6): Infinity measure 10**IPARAM(5), where 5<IPARAM(6)< 50 (default: 20)

IPARAM(7): Number of QP iterations, internally adapted (default 10). If set to 1, only one QP step per outer iteration will be made corresponding to a standard interior point method. In this case, however, the number of QP iterations might become increased if a descent direction is not achieved.

IPARAM(8): Step-length reduction factor, i.e., the line search will reduce the step-length by 1/IPARAM(8) until sufficient decrease property of the merit function is achieved (default 5). The parameter is in the range from 2 to 20.

The subsequent tolerances are passed to QPSLV:

IPARAM(11): Predictor-corrector method

1 - enable predictor step (default)

2 - use equal step lengths

4 - always use central step

8 - enable corrector step

IPARAM(12): Minimum $\mu$ value:

1 - ACC/100

2 - $\mu/100$

3 - $mu^2$

4 - predetermined constant ($10^{-8}$, default)

IPARAM(13): Update rule for sigma:

1 - constant (default)

2 - minimize violation of KKT condition

IPARAM(14): Regularization strategy (default 18):

0 - no regularization

1 - standard regularization
2 - $\|g(x) + s\|$
3 - $\lambda - y$
4 - $(g(x) + s)^2$
5 - $\|g(x) + s\|^2$
8 - regularize inequality constraints
16 - regularize right hand side
32 - regularize by vector instead of a single scalar
64 - automatic choice of regularization parameter
128 - constant regularization

IPARAM(17): Starting values for slack variable SL, multiplier Y, and internal search direction DL (default 1):

0 - predetermined in the calling program
1 - set internally to reduce complementary gap
2 - same rules as specified by Vanderbei [23]

The remaining positions are not yet used and should not be changed.

IFAIL : The parameter shows the reason for terminating a solution process. Initially, IFAIL must be set to zero. On return, IFAIL contains one the following values:

-2 - Compute new gradient values and call NLPIP again.
-1 - Compute new function values and call NLPIP again.
0 - Optimality conditions are satisfied.
1 - Stop after MAXIT iterations.
2 - No descent direction found for merit function.
3 - Local infeasibility or MFCQ failure detected.
4 - Line search failed, e.g., due to inaccurate derivatives.
5 - Length of a working array too short.
6 - False dimensions, M>LDG or N<0.
7 - Wrong entry in parameter field IPARAM.
8 - Starting point violates lower or upper bound.
9 - Wrong input parameter, e.g., N, M, IPRINT, IOUT, etc.
10 - Objective or constraint function unbounded.

| | |
|---|---|
| | 11 - NaN found in model function values.<br>12 - Termination due to more than MNFS steps without improvements of feasible function values.<br>>100 - Error in LINSLV, more details with IPRINT>0. |
| STEPP : | Double precision scalar allowing reduction of step size after the first iteration. Must not become less than 0 or greater than 1 and has to be 1 on start. |
| WORK(LWORK) : | Double precision working array of length LWORK. On return, WORK(1) contains the final norm of the KKT vector. |
| LWORK : | Length of WORK, has to be at least at least<br>32*N + 20*M + 6*P*N + 2*P*M + 8*P*P + 2*P + 100. |
| IWORK(LIWORK) : | Integer working array of length LIWORK. |
| LIWORK : | Length of integer work array, must be at least 3*P + 100. |

Note that depending on the error message, more information is displayed or written to the output channel in case of IPRINT>0.

Some of the termination reasons are enforced by insufficient accuracy of gradient values, in particular if computed by a difference formula. If we assume that all functions and gradients are computed within machine precision and that the implementation is correct, there remain the following possibilities that could cause an error message:

- The termination parameter ACC is too small, so that the numerical algorithm plays around with round-off errors without being able to improve the solution. Especially the approximation of the Hessian matrix of the Lagrangian function becomes unstable in this case. A straightforward remedy is to restart the optimization cycle again with a larger stopping tolerance.

- The constraints are contradicting, i.e. the set of feasible solutions is empty. There is no way to find out whether a general nonlinear and non-convex set possesses a feasible point or not. Thus, the nonlinear programming algorithms will proceed until running in any of the mentioned error situations. In this case, the correctness of the model must be very carefully checked.

- Constraints are feasible, but some of them there are degenerate, for example if some of the constraints are redundant. One should know that SQP algorithms assume the satisfaction of the so-called constraint qualification, i.e., that gradients of active constraints are linearly independent at each iterate and in a neighbourhood of an optimal solution. In this situation, it is recommended to check the formulation of the model constraints.

However, some of the error situations also occur if, especially due to inaccurate gradients, the solution of the quadratic programming subproblem or the primal-dual system (9), respectively, does not yield a descent direction for the underlying merit function. In this case, one should

try to improve the accuracy of function evaluations, scale the model functions in a proper way, or start the algorithm from other initial values.

A possible reason for reducing the step size could be to prevent non-computable function values, e.g., if the function evaluation in the calling program contains a logarithm to be taken for a negative variable value. STEPP must be set ot one when calling NLPIP the first time.

NLPIP must be linked to to the calling routine of the user, subroutine LINSLV, see below, the LAPACK [1] numerical algebra library, and a linear equation solver, e.g., PARDISO. LAPACK may be downloaded from the Netlib repository, see netlib.org. We recommend to use a LAPACK library of your Fortran compiler, if available. In addition, the code must be linked to the calling routine of the user,

## 3.2 LINSLV

Subroutine LINSLV serves to organize sparsity patterns of the Jacobian of the constraints and, in case of IPARAM(1)=1, of the Hessian matrix of the Lagrangian function. All accesses to the matrix of the primal dual system (19), i.e., the specification of sparsity patterns, the factorization, the solution of the system of equations subject to different right-hand sides, etc., are involved and must be provided by the user.

**Usage:**

CALL LINSLV(MODE, M, LDG, NHRS, N, IP, DG, A, B)

Note that some arrays serve for in- and output.

**Parameter Definition:**

| | |
|---|---|
| MODE : | Selection of one of the operations listed below. |
| M : | Total number of constraints. |
| LDG : | Length of array DG containing non-zero elements of the Jacobian of constraints. |
| NRHS : | Number of right-hand sides for which a solution of (9) or (21) is requested. |
| N : | Number of optimization variables. |
| IP : | Integer depending on MODE, also used to export error codes, reported as 100+IP by NLPIP. |

| | |
|---|---|
| DG(LDG) | Jacobian of the constraint functions at X. The data can be of any format as provided by the user and must match the definition of DG as passed to NLPIP. |
| A(MAX(N,M)) | In- and output array depending on MODE. |
| B(N+M,NRHS) | In- and output array depending on MODE. |

To illustrate the internal operations depending on the value of MODE, let $u \in I\!R^n$, $v \in I\!R^m$, $\sigma \in \{-1, 1\}$, and $z, b \in I\!R^{n+m}$.

**MODE=1:** Multiply transpose of the Jacobian matrix with a vector,

$$\sigma \ \nabla g(x)^T v + u \quad , \tag{22}$$

where $\sigma := IP$, $u := A$ and $v := B$. In simpler notation, compute $IP \cdot DG^T \cdot B + A$ and store the result in $A$, where IP=+1 or IP=-1.

**MODE=2:** Multiply Jacobian matrix with a vector,

$$\sigma \ \nabla g(x) u + v \quad , \tag{23}$$

where $\sigma := IP$, $v := A$ and $u := B$. In an equivalent notation, compute $IP \cdot DG \cdot B + A$ and store the result in $A$, where IP=+1 or IP=-1.

**MODE=3:** Multiply $i$-th row of the Jacobian matrix with a vector,

$$\nabla g_i(x) u \quad , \tag{24}$$

where $i := IP$ and $u := B$, is compute the product of the $IP$'th row of $DG$ times $B$ and store the result in $A(1)$.

**MODE=4:** Factorize the reduced KKT matrix (19)

$$\begin{pmatrix} H(x,u) + D_A & \nabla g(x)^T \\ \nabla g(x) & -D_B \end{pmatrix} z = b \tag{25}$$

where $D_A := diag(A)$ and $D_B := diag(B)$ $(= SU^{-1})$ are positive diagonal matrices. If IPARAM(1)=1, then $H(x,u)$ is the Hessian matrix of the Lagrangian function, i.e., $H(x,u) = \nabla_{xx} L(x,u)$, see (3), otherwise $H(x,u) = 0$. $diag(A)$ and $diag(B)$ are diagonal matrices, where the diagonal elements are the coefficients of the vectors $A$ and $B$, respectively. Factorization data must be stored internally depending on the applied linear algebra solver.

**MODE=5:** Same as for MODE=4, but only the diagonal entries are changed and an available factorization generated by a previous call of LINSLV with MODE=4 may be reused.

**MODE=6:** Solve the linear system (25)

$$\begin{pmatrix} H(x,y)+D_A & \nabla g(x)^T \\ \nabla g(x) & -D_B \end{pmatrix} z = b \tag{26}$$

subject to $z$, where $H(x,y)$ is either the Hessian matrix of the Lagrangian function, if IPARAM(1)=1, or zero, if IPARAM(1)=0. $D_A$ and $D_B$ are positive diagonal matrices. An available factorization generated by a previous call of LINSLV with MODE=4 may be reused.

**MODE=7** Same as for MODE=6, but solve the linear system with NRHS multiple right-hand sides all stored in $B$.

**MODE=8** Compute $H(x,u)\cdot B$ and store the result in $A$, where $H(x,u)$ is the user provided Hessian. This mode is executed only if IPARAM(1)=1.

When factorizing the reduced KKT system, one hat to take into account that the underlying interior point method generates entries of $D_B$ tending to zero, while others tend to infinity. Also some of the entries of $D_A$ may tend to infinity. The coefficients of both $D_A$ and $D_B$ are all nonnegative.

# 4 Example

To give an example, we consider the following problem,

$$x \in I\!R^3: \quad \begin{aligned} &\min \quad -x_1x_2x_3 \\ &g_1(x) = -x_1 - 2x_2 - 2x_3 \le 0 \ , \\ &g_2(x) = -72 + x_1 + 2x_2 + 2x_3 \le 0 \ , \\ &0 \le x_i \le 100, \ i = 1,2,3 \end{aligned}$$

The Fortran source code calling NLPIP is listed below, where the initial penalty parameter is set to exp(9). Gradients are given in analytical form. We use a dense Jacobian and solve the KKT system with PARDISO, see e.g., Schenk and Gärtner [17], as part of Intel's MKL. Other examples that also demonstrate the use of sparse matrices, are shipped together with the NLPIP code.

```
      PROGRAM          NLPIP_DEMOB
      IMPLICIT         NONE
      INTEGER          N, M, LDG, MNN2, LWA, LKWA, P
      PARAMETER (      N     = 3,
     /                 M     = 2,
     /                 P     = 7,
```

```
     /                 LDG   = 6,
     /                 MNN2  = M + N + N + 2,
     /                 LWA   = 32*N + 20*M + 6*P*N + 2*P*M
     /                                     + 8*P*P + 2*P + 100,
     /                 LKWA  = 3*P + 100)
      INTEGER          KWA(LKWA), IFAIL, IPRINT, IOUT, MAXFUN, MAXIT,
     /                 IPARAM(20), I
      DOUBLE PRECISION X(N), ACC, ACCQP, STEPP, F, G(M), DF(N), DG(LDG),
     /                 Y(MNN2), SL(MNN2), XL(N), XU(N), WA(LWA)
      LOGICAL          ACTIVE(MNN2)
C
C   Set some constants and initial values
C
      IFAIL  = 0
      IPRINT = 2
      IOUT   = 6
      MAXFUN = 20
      MAXIT  = 200
      ACC    = 1.0D-10
      ACCQP  = 1.0D-16
      STEPP  = 1.0D0
      DO I=1,N
         X(I)  = 10.0D0
         XL(I) = 0.0D0
         XU(I) = 42.0D0
      END DO
      DO I=1,20
         IPARAM(I) = -100
      END DO
    1 CONTINUE
C============================================================
C   This block computes all function values.
C
      F    = -X(1)*X(2)*X(3)
      G(1) = -X(1) - 2.0D0*X(2) - 2.0D0*X(3)
      G(2) = -72.0D0 + X(1) + 2.0D0*X(2) + 2.0D0*X(3)
C
C============================================================
      IF (IFAIL.EQ.-1) GOTO 4
    2 CONTINUE
C============================================================
C   This block computes all derivative values.
C
      DF(1) = -X(2)*X(3)
```

```
      DF(2) = -X(1)*X(3)
      DF(3) = -X(1)*X(2)
      DG(1) = -1.0D0
      DG(2) =  1.0D0
      DG(3) = -2.0D0
      DG(4) =  2.0D0
      DG(5) = -2.0D0
      DG(6) =  2.0D0
C
C=============================================================
    4 CONTINUE
      CALL NLPIP (      M,      0,    LDG,      N,      X,
     /                   F,      G,     DF,     DG,      Y,
     /                  SL,     XL,     XU, ACTIVE,      P,
     /                 ACC,  ACCQP, MAXFUN,  MAXIT, IPRINT,
     /                IOUT, IPARAM,  IFAIL,  STEPP,     WA,
     /                 LWA,    KWA,   LKWA                )
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
C
      STOP
      END
C
C ************************************************************
C *   LINSLV interface routine based on PARDISO
C ************************************************************
C
      SUBROUTINE LINSLV( MODE, M, LDG, NRHS, N, IP, DG, A, B)
      IMPLICIT           NONE
      INTEGER            MODE, M, LDG, N, NRHS, IP
      DOUBLE PRECISION   DG, A, B
      DIMENSION          DG(LDG), A(M*N+M+N), B((M+N)*NRHS)
C
C   Local parameters
C
      DOUBLE PRECISION   W, SUM
      INTEGER            IW
      DIMENSION          W(100), IW(100)
      INTEGER            I, J, NM
      COMMON            /RLINSLV/ W
      COMMON            /ILINSLV/ IW
C
C   Switch
C
```

```
      NM = N + M
      GOTO (1,2,3,4,5,6,6), MODE
C=============================================================
C   Compute A = IP*DG'*B + A
C
    1 CONTINUE
      DO I=1,N
         SUM = 0.0D0
         DO J=1,M
            SUM = SUM + DG(M*(I-1)+J)*B(J)
         END DO
         A(I) = DBLE(IP)*SUM + A(I)
      END DO
      RETURN
C=============================================================
C   Compute A = IP*DG*B + A
    2 CONTINUE
      DO J=1,M
         SUM = 0.0D0
         DO I=1,N
            SUM = SUM + DG(M*(I-1)+J)*B(I)
         END DO
         A(J) = DBLE(IP)*SUM + A(J)
      END DO
      RETURN
C=============================================================
C   Compute A(1) = (IP'th row of DG)*B
C
    3 CONTINUE
      A(1) = 0.0D0
      DO I=1,N
         A(1) = A(1) + DG(M*(I-1)+IP)*B(I)
      END DO
      RETURN
C=============================================================
C   Factorize (full)
C
    4 CONTINUE
      CALL LINSL1(1, M, N, LDG, A, B, DG, W(2*(N+M)+1), IW,
     /            W, W(N+M+1), IP)
      RETURN
C=============================================================
C   Factorize (diag update)
C
```

```
    5 CONTINUE
      CALL LINSL1(2, M, N, LDG, A, B, DG, W(2*(N+M)+1), IW,
     /            W, W(N+M+1),IP)
      RETURN
C============================================================
C   Solve B=C\B
C
    6 CONTINUE
      DO J = 1, NRHS
         CALL LINSL1(3, M, N, LDG, A, B, DG, W(2*(N+M)+1), IW,
     /               B(NM*(J-1)+1), W(N+M+1), IP)
      END DO
C
      RETURN
      END
C
C ************************************************************
C *   Auxiliary routine LINSL1 for LINSLV
C ************************************************************
C
      SUBROUTINE LINSL1(MODE, M, N, LDG, D1, D2, DG, A, P, B, X, IP)
      IMPLICIT         NONE
      INTEGER          MODE, M, N, LDG, P, IP
      DOUBLE PRECISION D1, D2, DG, A, B, X
      DIMENSION        D1(N), D2(M), DG(LDG), A(M*N+M+N)
      DIMENSION        P(64+2*(N+M)+(N+1)*(M+1)), B(N+M), X(N+M)
      INTEGER          I, J
      INTEGER*8        PT(64)
      DATA             (PT(I),I=1,64) /
     /                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     /                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     /                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     /                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/
      SAVE             PT
      INTEGER          IPERM, NM, IIA, IJA
C
      IPERM = 65
      NM    = N + M
      IIA   = IPERM + NM
      IJA   = IIA + NM + 1
      GOTO (1,2,3), MODE
C============================================================
C   Build upper right part of the symmetric KKT matrix in CSR format...
C         (  diag(D1)   :    DG^T      )
```

```
C         (  DG         :   -diag(D2)  )
C
    1 CONTINUE
      DO I=1,N
         P(IIA+I-1)         = 1 + (I-1)*(1+M)
         P(IJA+(I-1)*(1+M)) = I
         A(1+(I-1)*(1+M))   = D1(I)
         DO J=1,M
            P(IJA+(I-1)*(1+M)+J) = N + J
            A(1+(I-1)*(1+M)+J)   = DG((I-1)*M+J)
C            A(1+(I-1)*(1+M)+J)   = DG(J,I)
         END DO
      END DO
      DO I=1,M
         P(IIA+N+I-1)       = N*(1+M) + I
         P(IJA+N*(1+M)+I-1) = N + I
         A(N*(1+M)+I)       = -D2(I)
      END DO
      P(IIA+N+M) = N*(1+M) + M + 1
C   ... and factorize it
      CALL IFILL(64, 0, P, 1)
      P(1)  = 1
      P(2)  = 0
      P(5)  = 0
      P(8)  = 20
      P(10) = 8
      P(11) = 1
      P(13) = 1
      P(18) = -1
      P(21) = 1
      CALL PARDISO(PT, 1, 1, -2, 12, NM, A, P(IIA), P(IJA),
     /             P(IPERM), 1, P, 0, B, X, IP)
      RETURN
C=============================================================
C   Update diagonal entries
C
    2 CONTINUE
      DO I=1,N
         A(1+(I-1)*(1+M)) = D1(I)
      END DO
      DO I=1,M
         A(N*(1+M)+I) = -D2(I)
      END DO
      CALL PARDISO(PT, 1, 1, -2, 22, NM, A, P(IIA), P(IJA),
```

```
     /             P(IPERM), 1, P, 0, B, X, IP)
      RETURN
C=============================================================
C   Solve factorized system
C
    3 CONTINUE
      CALL PARDISO(PT, 1, 1, -2, 33, NM, A, P(IIA), P(IJA),
     /             P(IPERM), 1, P, 0, B, X, IP)
      CALL DCOPY(NM, X, 1, B, 1)
C
      RETURN
      END
```

The subsequent output is generated:

```
-------------------------------------------------------------------
Start of the IPM/SQP Algorithm NLPIP for Large-Scale Optimization
                    Version 2.0 (Jul 2013)
-------------------------------------------------------------------

  Parameter Values:
     M      =         2
     ME     =         0
     LDG    =         6
     N      =         3
     P      =         7
     ACC    =   0.1000D-09
     ACCQP  =   0.1000D-11
     MAXFUN =        20
     MAXIT  =       100
     MNFS   =         2
     IPRINT =         2
     IOUT   =         6

  Output in the following order:
     IT    - iteration number
     F     - objective function value
     MCV   - maximum constraint violation
     NA    - number of active constraints
     I     - number of line search iterations
     ALPHA - steplength parameter
     KKT   - Karush-Kuhn-Tucker optimality criterion
```

```
 IT        F             MCV        ALPHA         |D|        KKT
-------------------------------------------------------------------
  1 -0.10000000D+04   0.00D+00   0.00D+00   0.00D+00   0.42D+02
  2 -0.17494811D+04   0.00D+00   0.10D+01   0.32D+02   0.31D+03
  3 -0.32661433D+04   0.00D+00   0.10D+01   0.12D+02   0.26D+02
  4 -0.33618792D+04   0.00D+00   0.10D+01   0.12D+02   0.76D+01
  5 -0.34540311D+04   0.00D+00   0.10D+01   0.54D+01   0.30D+00
  6 -0.34559844D+04   0.00D+00   0.10D+01   0.64D+00   0.40D-02
  7 -0.34560000D+04   0.00D+00   0.10D+01   0.61D-01   0.14D-04
  8 -0.34560000D+04   0.00D+00   0.10D+01   0.85D-03   0.10D-06
  9 -0.34560000D+04   0.00D+00   0.10D+01   0.10D-05   0.10D-08
 10 -0.34560000D+04   0.00D+00   0.10D+01   0.11D-10   0.50D-11
-------------------------------------------------------------------

 The following solution was found:
      X(1), X(2), ...           =  0.24000000D+02  0.12000000D+02 ...
      Objective function        = -0.34560000D+04
      Max constraint violation =  0.00000000D+00
```

# 5 Numerical Test Results

We solve all test problems with the same set of tolerances and parameters, which are internally stored as default values and which are accessed by predefining -100 in IPARAM(20). The maximum number of iterations is 500. Moreover, the number of recursive LM-Quasi-Newton updates is $p = 7$. The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 11.0, 64 bit, under Windows 7 and Intel(R) Core(TM) i7-2720 CPU, 2.2 GHz, with 8 GB RAM.

## 5.1 Elliptic Optimal Control Problems with Control and State Constraints

Maurer and Mittelmann [13, 14] 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 $N$ in x- and y-direction, where the following abbreviations are used in Table 1:

| *prob* | $n$ | $m$ | $n_{func}$ | $n_{grad}$ | $f(x^\star)$ | $r(x^\star)$ | $c_{KKT}(x^\star)$ | *time* |
|---|---|---|---|---|---|---|---|---|
| EX 1 | 10,197 | 9,801 | 13 | 13 | 0.19652520 | 0.38E-14 | 0.21E-08 | 32.9 |
| EX 2 | 10,197 | 9,801 | 19 | 19 | 0.09669517 | 0.38E-14 | 0.81E-08 | 39.1 |
| EX 3 | 10,197 | 9,801 | 12 | 12 | 0.32100999 | 0.33E-14 | 0.11E-08 | 25.9 |
| EX 4 | 10,197 | 9,801 | 12 | 12 | 0.24917886 | 0.38E-14 | 0.25E-08 | 23.5 |
| EX 5 | 10,593 | 10,197 | 15 | 15 | 0.55224625 | 0.73E-10 | 0.11E-08 | 36.1 |
| EX 6 | 10,593 | 10,197 | 21 | 21 | 0.01507905 | 0.11E-08 | 0.11E-08 | 42.9 |
| EX 7 | 10,593 | 10,197 | 18 | 16 | 0.26389870 | 0.26E-11 | 0.11E-08 | 36.6 |
| EX 8 | 10,593 | 10,197 | 17 | 15 | 0.16166432 | 0.23E-10 | 0.17E-08 | 32.8 |
| EX 9 | 19,602 | 9,801 | 15 | 15 | 0.06216416 | 0.87E-13 | 0.12E-08 | 44.1 |
| EX 10 | 19,602 | 9,801 | 20 | 20 | 0.05645702 | 0.23E-15 | 0.10E-08 | 188.0 |
| EX 11 | 19,602 | 9,801 | 15 | 15 | 0.11026722 | 0.84E-13 | 0.11E-08 | 33.4 |
| EX 12 | 19,998 | 10,197 | 20 | 20 | 0.07806690 | 0.27E-10 | 0.50E-08 | 48.7 |
| EX 13 | 19,998 | 10,197 | 25 | 25 | 0.05267343 | 0.82E-13 | 0.24E-08 | 101.6 |

Table 1: Test Results for Semilinear Elliptic Control Problems

| | |
|---|---|
| *prob* | test problem identifier, |
| $n$ | number of optimization variables passed to NLPIP, |
| $m$ | 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 $x^\star$ |
| $r(x^\star)$ | maximum constraint violation at termination point $x^\star$ |
| $c_{KKT}(x^\star)$ | KKT convergence criterion at $x^\star$ |
| *time* | total CPU time in seconds |

One function evaluation consists of the computation of one objective function value and $m$ constraint function values. Derivatives are available in analytical form.

We apply the IPM version of NLPIP, and request the default parameter settings of all options. The number of limited-memory quasi-Newton steps is $p = 7$. The subsequent table contains results obtained for IPARAM(7)=1, i.e., the number of internal iterations to solve the QP is set to one and the algorithm behaves like a standard IPM method. Grid size is $N = 100$ in both directions, maximum number of iterations is 200 and termination accuracy is set to $10^{-8}$.

## 5.2 Small and Dense HS-Problems

We evaluate the performance of NLPIP on the set of 306 small-scale, but often highly nonlinear test problems of Hock and Schittkowski [12, 20], and compare the results to the solver NLPQLP, a dense implementation of an SQP-method, see Schittkowski [18, 19].

NLPIP is called with default parameters, forward differences for gradient approximations,

| *code* | $n_{succ}$ | $n_{func}$ | $n_{grad}$ | *time* |
|---|---|---|---|---|
| NLPIP ($p = 7$) | 300 | 48 | 23 | 1.2 |
| NLPIP ($p = 70$) | 299 | 28 | 17 | 3.0 |
| NLPQLP | 306 | 35 | 20 | 0.3 |

Table 2: Test Results for 306 Hock-Schittkowski Problems

and termination accuracy $10^{-5}$. In Table 2, we present results for $p = 7$ and $p = 70$.

NLPIP needs about the same number of iterations and function evaluations when compared to NLPQLP. Since the quadratic programming problem is iteratively solved by NLPIP, average computation times are higher, especially in case of a large number of limited-memory updates.

### 5.3 The CUTEr Collection

A large number of test problems for nonlinear programming has been collected and programmed a so-called Standard Input Format (SIF), from which, e.g., Fortran code is generated, see Gould, Orban, and Toint [10]. 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 selected 34 test problems with 5,000 or more variables. The problems are identified by their internal serial number and a identifying name. They only possess equality constraints with four exceptions. Numerical results are listed in Table 3 for default parameter settings and $p = 7$. Termination accuracy is $10^{-6}$.

## References


[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen (1999): *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA

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

[3] Byrd R.H., Gilbert J.C., Nocedal J. (2000): *A trust region method based on interior point techniques for nonlinear programming,* Mathematical Programming Vol. 89, 149–185

[4] Chen L., Goldfarb D. (2009): *An interior-point piecewise linear penalty method for nonlinear programming*, Mathematatuical Programming, Vol. 10, 1-50.

[5] Curtis F.E., Nocedal J. (2008): *Flexible penalty functions for nonlinear constrained optimization*, Journal of Numerical Analysis, Vol. 28, 335-351

| *no* | *prob* | $n$ | $m$ | $m_e$ | $n_{func}$ | $n_{grad}$ | $f(x^\star)$ | $r(x^\star)$ | *time* |
|---|---|---|---|---|---|---|---|---|---|
| 53 | GILBERT | 5,000 | 1 | 1 | 53 | 37 | 0.2459470E+04 | 0.33E-06 | 17.5 |
| 222 | BRAINPC0 | 6,907 | 6,900 | 6,900 | 68 | 38 | 0.1499973E-02 | 0.93E-06 | 86.2 |
| 223 | BRAINPC1 | 6,907 | 6,900 | 6,900 | 42 | 31 | 0.1450644E-08 | 0.31E-07 | 70.2 |
| 224 | BRAINPC2 | 13,807 | 13,800 | 13,800 | 15 | 12 | 0.7058297E-03 | 0.56E-06 | 36.2 |
| 225 | BRAINPC3 | 6,907 | 6,900 | 6,900 | 17 | 14 | 0.2792666E-03 | 0.14E-06 | 19.5 |
| 226 | BRAINPC4 | 6,907 | 6,900 | 6,900 | 577 | 177 | 0.1292671E-02 | 0.33E-06 | 503.5 |
| 227 | BRAINPC5 | 6,907 | 6,900 | 6,900 | 16 | 12 | 0.2333637E-02 | 0.25E-07 | 17.4 |
| 228 | BRAINPC6 | 6,907 | 6,900 | 6,900 | 18 | 15 | 0.1864842E-03 | 0.45E-06 | 20.2 |
| 229 | BRAINPC7 | 6,907 | 6,900 | 6,900 | 277 | 96 | 0.3862754E-04 | 0.78E-06 | 229.6 |
| 230 | BRAINPC8 | 6,907 | 6,900 | 6,900 | 327 | 117 | 0.1784133E-03 | 0.84E-06 | 367.6 |
| 231 | BRAINPC9 | 6,907 | 6,900 | 6,900 | 96 | 40 | 0.3516188E+00 | 0.86E-06 | 113.2 |
| 232 | CAR2 | 5,999 | 4,996 | 3,996 | 151 | 77 | 0.2666182E+01 | 0.10E-05 | 194.4 |
| 235 | CLNLBEAM | 60,003 | 40,000 | 40,000 | 9 | 9 | 0.3500000E+03 | 0.16E-08 | 61.5 |
| 236 | CORKSCRW | 45,006 | 35,000 | 30,000 | 28 | 25 | 0.9810425E+02 | 0.18E-12 | 518.0 |
| 237 | COSHFUN | 6,001 | 2,000 | 0 | 1,578 | 500 | -0.7732396E+00 | 0.00E+00 | 409.4 |
| 240 | DRUGDIS | 6,004 | 4,000 | 4,000 | 59 | 42 | 0.4278234E+01 | 0.25E-09 | 64.2 |
| 241 | DTOC1NA | 5,998 | 3,996 | 3,996 | 12 | 12 | 0.4138914E+01 | 0.34E-06 | 2.8 |
| 242 | DTOC1NB | 5,998 | 3,996 | 3,996 | 15 | 14 | 0.7138852E+01 | 0.61E-07 | 3.6 |
| 243 | DTOC1NC | 5,998 | 3,996 | 3,996 | 17 | 14 | 0.3519935E+02 | 0.27E-06 | 3.9 |
| 244 | DTOC1ND | 5,998 | 3,996 | 3,996 | 36 | 23 | 0.4760303E+02 | 0.59E-06 | 6.8 |
| 245 | DTOC2 | 5,998 | 3,996 | 3,996 | 76 | 47 | 0.5074931E+00 | 0.59E-06 | 14.4 |
| 247 | DTOC5 | 9,999 | 4,999 | 4,999 | 95 | 95 | 0.1531073E+01 | 0.97E-06 | 42.0 |
| 248 | DTOC6 | 10,001 | 5,000 | 5,000 | 21 | 20 | 0.1348504E+06 | 0.26E-06 | 10.2 |
| 255 | JUNKTURN | 10,010 | 7,000 | 7,000 | 595 | 295 | 0.1716158E-02 | 0.67E-06 | 200.6 |
| 267 | OPTMASS | 60,010 | 50,005 | 40,004 | 85 | 27 | -0.2409747E-01 | 0.11E-09 | 350.3 |
| 269 | ORTHRDM2 | 8,003 | 4,000 | 4,000 | 17 | 15 | 0.3110153E+03 | 0.25E-08 | 13.1 |
| 270 | ORTHRDS2 | 5,003 | 2,500 | 2,500 | 181 | 104 | 0.3288840E+04 | 0.77E-06 | 51.1 |
| 271 | ORTHREGA | 8,197 | 4,096 | 4,096 | 87 | 57 | 0.2660367E+05 | 0.12E-08 | 58.8 |
| 273 | ORTHREGC | 5,005 | 2,500 | 2,500 | 191 | 84 | 0.9481310E+02 | 0.11E-07 | 44.7 |
| 274 | ORTHREGD | 5,003 | 2,500 | 2,500 | 142 | 94 | 0.3289923E+04 | 0.85E-06 | 52.4 |
| 275 | ORTHREGE | 7,506 | 5,000 | 5,000 | 99 | 59 | 0.1278650E+04 | 0.13E-08 | 68.4 |
| 277 | ORTHRGDM | 10,003 | 5,000 | 5,000 | 397 | 206 | 0.1110009E+05 | 0.56E-06 | 260.3 |
| 278 | ORTHRGDS | 5,003 | 2,500 | 2,500 | 2,046 | 467 | 0.8946430E+04 | 0.96E-06 | 245.1 |
| 281 | READING5 | 5,001 | 5,000 | 5,000 | 60 | 31 | -0.2543357E-14 | 0.11E-12 | 72.6 |

Table 3: Test Results for cuter-Problems, $n \geq 5000$

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

[7] Fletcher R. (1987): *Practical Methods of Optimization*, John Wiley, Chichester

[8] Gill P.E., Murray W., Wright M. (1982): *Practical Optimization*, Academic Press, London

[9] Gould N.I.M, Toint Ph.L. (1999): *SQP methods for large-scale nonlinear programming,*, Proceedings of the 19th IFIP TC7 Conference on System Modelling and Optimization: Methods, Theory and Applications, pp. 149–178.

[10] Gould N.I.M., Orban D., Toint Ph.L. (2005): *General CUTEr documentation*, CERFACS Technical Report TR/PA/02/13, 2005

[11] Griva I., Shanno D.F., Vanderbei R.J., Benson H.Y. (2008): *Global convergence of a primal-dual interior-point method for nonlinear programming*, Algorithmic Operations Research, Vol. 3, 12–19

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

[13] H. Maurer, H.D. Mittelmann (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

[14] H. Maurer, H.D. Mittelmann (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

[15] Nocedal J., Wächter A., Waltz R.A. (2006): *Adaptive barrier strategies for nonlinear interior methods*, Technical Report RC 23563, IBM T.J. Watson Research Center

[16] 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, Berlin

[17] Schenk O., Gärtner K. (2006): *On fast factorizing pivoting methods for symmetric indefinite systems*, Electronic Transactions on Numerical Analysis, Vol. 23, 158-179

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

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

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

[21] K. Schittkowski (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 (2009)

[22] Sun W.Y., Yuan Y. (2006) *Optimization Theory and Methods: Nonlinear Programming*, Springer, New York

[23] Vanderbei R.J. (1999): *LOQO: An interior point code for quadratic programming*, Optimization Methods and Software, Vol. 11, 451-484

[24] Waltz R.A., Morales J.L., Novedal J., Orban D. (2006): *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Mathematical Programming, Vol. 107, 391-408