

NLPIP: A Fortran implementation of an SQP Interior Point algorithm for solving large scale nonlinear optimization problems

-User's Guide-

Björn Sachsenberg

July 29, 2010

Abstract

The Fortran subroutine NLPIP is designed to solve smooth large scale nonlinear optimization problems. The underlying algorithm is based on an SQP method where the quadratic subproblem is solved by a primal-dual interior point method. A feature of the algorithm is, that the quadratic subproblem is not necessarily solved exactly. To be able to solve large problems it can either use a limited memory BFGS update to approximate the Hessian of the Lagrangian or let the user specify the (possibly sparse) Hessian. The Jacobian of the constraints as well as the Hessian of the Lagrangian may be in any format. The user only has to provide necessary operations concerning the Jacobians and Hessians. Numerical results are included for some elliptic control problems by Maurer and Mittelmann with over 5 million variables and 2.5 million constraints.

1 Introduction

We consider the general optimization problem to minimize an objective function under nonlinear equality and inequality constraints.

$$\begin{aligned} \min \quad & f(x) \\ x \in \mathbb{R}^n : \quad & g_j(x) = 0, \quad j = 1, \dots, m_e, \\ & g_j(x) \leq 0, \quad j = m_e + 1, \dots, m, \\ & x_l \leq x \leq x_u, \end{aligned}$$

where x is an n -dimensional parameter vector. It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \dots, m$, are twice continuously differentiable on the whole \mathbb{R}^n . To simplify the notation, we omit the upper and lower bounds and equality constraints to get a problem of the form

$$\begin{aligned} \min \quad & f(x) \\ x \in \mathbb{R}^n : \quad & g(x) \leq 0. \end{aligned}$$

The basic idea of the algorithm is to mix sequential quadratic programming (SQP) and interior point (IP) methods for nonlinear programming. The outer loop is just an ordinary SQP loop, i.e. it constructs a sequence of quadratic programming subproblems by approximating the Lagrangian function

$$L(x, y) := f(x) + \sum_{i=1}^m y_i g_i(x)$$

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

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \frac{1}{2} d^T H(x_k, y_k) d + \nabla f(x_k)^T d \\ \text{subject to} \quad & g(x_k) + \nabla g(x_k) d \leq 0 \end{aligned} \tag{1}$$

is handed over to an interior point solver. Here (x_k, y_k) denotes the current iterate in primal-dual space and $H(x_k, y_k)$ is the Hessian of the Lagrangian or an approximation of it at the current iterate. The result from the IP solver is used as search direction. A step length along the search direction is determined to obtain the new iterate.

In Section 2 we give a brief overview of the algorithm. A detailed description can be found in [8]. In Section 3 we show how to use the Fortran subroutine. An example is given in Section 4. Finally we present some numerical results in Section 5.

2 The Algorithm

The idea of the proposed algorithm is to use an SQP method where the quadratic subproblem is solved using an Interior Point Method. So the algorithm basically consists of two nested loops (see Algorithm 1). Therefore we have two iteration indices k and l . By x^k, s^k, λ^k we denote the outer iterates of the primal, slack and dual variables, respectively. Correspondingly $d_x^{k,l}, d_s^{k,l}$ and $d_\lambda^{k,l}$ are the inner iterates. Here, d stands for direction. If we had a pure SQP method, the inner loop would compute the solution of the quadratic subproblem while the outer loop promotes convergence for the NLP via a line search along the direction obtained by the inner loop. In our algorithm however it is not necessary that the quadratic subproblem is solved exactly. In fact it is formulated in a way that leaves many degrees of freedom.

To avoid singularity when the gradients of the active constraints get linear dependent at some iterate, we perturb the linear system of the IPM (2) by two diagonal, positive semidefinite regularization matrices $B_{k,l}, C_{k,l} \in \mathbb{R}^{m \times m}$. So for the inner loop we solve the regularized KKT system

$$\begin{pmatrix} H^k & \nabla g(x^k)^T & 0 \\ \nabla g(x^k) & -B_{k,l} & I \\ 0 & S_{k,l} & \Lambda_{k,l} \end{pmatrix} \begin{pmatrix} \Delta d_x^{k,l} \\ \Delta d_\lambda^{k,l} \\ \Delta d_s^{k,l} \end{pmatrix} = - \begin{pmatrix} H^k d_x^{k,l} + \nabla f(x^k) + \nabla g(x^k)^T \lambda^{k,l} \\ g(x^k) + \nabla g(x^k) d_x^{k,l} + s^{k,l} - C_{k,l} \lambda^{k,l} \\ S_{k,l} \Lambda_{k,l} e - \mu_{k,l} e \end{pmatrix} \quad (2)$$

and set

$$\begin{aligned} d_x^{k,l+1} &= d_x^{k,l} + \alpha_p^{k,l} \Delta d_x^{k,l}, \\ d_\lambda^{k,l+1} &= d_\lambda^{k,l} + \alpha_d^{k,l} \Delta d_\lambda^{k,l}, \\ d_s^{k,l+1} &= d_s^{k,l} + \alpha_p^{k,l} \Delta d_s^{k,l}. \end{aligned} \quad (3)$$

where $\alpha_p, \alpha_d \in (0, 1]$ are primal and dual step lengths, such that $\lambda^{k,l+1} := \lambda^k + d_\lambda^{k,l+1} > 0$, $s^{k,l+1} := s^k + d_s^{k,l+1} > 0$ and $\Phi_q(z^k, d^{k,l+1}, \mu_{k,l}, r_{k,l+1}) \leq \Phi_q(z^k, 0, \mu_{k,l}, r_{k,l+1})$ with Φ_q being a merit function for the quadratic subproblem.

Now the difference to an ordinary SQP method is that the QP is not necessarily solved completely. In fact just a few inner iterations are performed within the interior point method.

2.1 Solving the linear system

With $\Delta d_s = \Lambda^{-1}(\mu e - S \Delta d_\lambda) - S e$ we can simplify (2) to the so-called reduced KKT system [3]

$$\begin{pmatrix} H & \nabla g(x)^T \\ \nabla g(x) & -S \Lambda^{-1} - B \end{pmatrix} \begin{pmatrix} \Delta d_x \\ \Delta d_\lambda \end{pmatrix} = - \begin{pmatrix} H d_x + \nabla f(x) + \nabla g(x)^T \lambda \\ g(x) + \nabla g(x) d_x + \sigma \mu \Lambda^{-1} e - C \lambda \end{pmatrix} \quad (4)$$

So we need to solve linear systems of the form

$$Cz = b$$

with

$$C = \begin{pmatrix} D_1 & G^T \\ G & -D_2 \end{pmatrix} \quad (5)$$

where D_1 and D_2 are positive definite diagonal matrices.

Algorithm 1 SQP-IP

1. Choose starting values $z^0 = (x^0, \lambda^0, s^0)$ and parameter $\nu \in (0, \frac{1}{2})$

2. For $k := 0, 1, 2, \dots$

(a) Check stopping criteria

(b) Choose $d_x^{k,0}, d_s^{k,0}, d_\lambda^{k,0}$

(c) For $l := 0, 1, 2, \dots$

i. Determine $\mu_{k,l}, B_{k,l}, C_{k,l}$

ii. Solve (2) and apply (3)

iii. If termination criteria for QP is satisfied, break for-loop.

(d) Find steplengths $\alpha_p, \alpha_d \in (0, 1]$ such that

$$\Phi_{\mu_k, r_k}(x^k + \alpha_p d_x^{k,l}, s^k + \alpha_p d_s^{k,l}, \lambda^k + \alpha_d d_\lambda^{k,l}) \leq \Phi_{\mu_k, r_k}(x^k, s^k, \lambda^k) + \nu \alpha_p \nabla \Phi_{\mu_k, r_k}(x^k, s^k, \lambda^k)^T (d_x^{k,l}, d_s^{k,l}, d_\lambda^{k,l})$$

(e) Set $x^{k+1} := x^k + \alpha_p d_x^{k,l}, s^{k+1} := s^k + \alpha_p d_s^{k,l}, \lambda^{k+1} := \lambda^k + \alpha_d d_\lambda^{k,l}$

2.2 Limited Memory

To approximate the Hessian of the Lagrangian we choose a limited memory BFGS update [4, 13]. The standard BFGS method stores the whole matrix which is updated in every iteration by the rule

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

with

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

Usually, we start with the unit matrix for H_0 and stabilize the formula by requiring that $a_k^T b_k \geq 0.2 b_k^T H_k b_k$, see Powell [7], Stoer [12], or Schittkowski [10].

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

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

for $i = 0, \dots, p-1$ with $0 < p \ll n$. These pairs of vectors are used to implicitly construct the approximation of the Hessian matrix at x_{k+1} . Note that it does not need to be stored explicitly and therefore we need only $O(pn)$ instead of $O(n^2)$ double precision numbers for the BFGS update. Now we omit the iteration index k for simplicity. 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. Both M and N can be obtained directly from the p stored pairs of vectors and ξ without any additional information. Now we can write the inverse of the left hand matrix in (4) as

$$\begin{aligned} & \left[\begin{pmatrix} \xi I & \nabla g(x)^T \\ \nabla g(x) & -S\Lambda^{-1} \end{pmatrix} + \begin{pmatrix} N \\ 0 \end{pmatrix} \begin{pmatrix} MN^T & 0 \end{pmatrix} \right]^{-1} \\ & =: [C + UV^T]^{-1} \\ & = C^{-1} - C^{-1} U \underbrace{(I + V^T C^{-1} U)^{-1}}_{\in \mathbb{R}^{2p \times 2p}} V^T C^{-1} \end{aligned} \tag{6}$$

So instead of solving (4) directly, we solve the system $Cz = b$ several times with different right hand sides. The matrix $I + V^T C^{-1} U$ is only of size $2p \times 2p$ and can be factored at negligible cost.

3 Program Documentation

3.1 NLPIP

NLPIP is implemented in form of a FORTRAN77 subroutine, which computes one iteration of the algorithm. Model functions and gradients are called by reverse communication. The user has to provide functions and gradients in the same program which executes NLPIP, according to the following rules:

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. 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.

Usage:

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

Definition of the parameters:

M	Total number of constraints
ME	Number of equality constraints
MMAX	Will be passed to LINSLV
N	Number of variables
X(N)	Initially, 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	Value of the objective function at X
G(M)	Values of the constraints at X. In the driving program the dimension has to be at least M
DF(N)	Gradient of the objective function
DG	Jacobian of the constraint function. This variable can be of any format since it is passed to LINSLV only.
Y(M+2*N)	On return it contains the multipliers with respect to the current iterate stored in X. The first M entries contain the multipliers of the M constraints given in G, the subsequent N entries contain the multipliers of the lower bounds and the last N entries contain the multipliers of the upper bounds.

SL(M+2*N) On return it 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 given in G.

XL(N) Lower bounds for X

XU(N) Upper bounds for X

ACTIVE(M+2*N) Logical array which on return indicates constraints which are 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 stored BFGS update pairs. Typical values are in the range [3,...,20]. You will need $\mathcal{O}(P^2)$ memory (see LWORK).

ACC The user has to specify the desired final accuracy (e.g. 1.0D-7). The termination accuracy should not be much smaller than the accuracy by which gradients are computed.

ACCQP Accuracy for the QP. This should be smaller than ACC and greater than the machine precision.

MAXFUN Maximum number of linesearch iterations

MAXIT Maximum number of outer iterations

IPRINT Output level
0 - no output
1 - only final solution and errors
2 - also outer iterates (one line for each iteration)
3 - also final solution of each subproblem and the line search
4 - also inner iterates

IOUT Output unit number (6=stdout)

IPARAM(20) Integer Parameters

Index	Value	Meaning
(1)	0	use default parameters (the rest of IPARAM will be overwritten with default values)
	1	provide all parameters (this is experimental and some parameters are subject to change in future)
	2	use default parameters for standard SQP method
	3	use default parameters for standard IP method.
(2)		merit function
(3)		rule for μ update
(4)		$10^{IPARAM(4)}$ will be treated as infinity (default = 20)
(11)	1	enable predictor corrector steps
	2	use equal step lengths
	4	always use center steps
(12)		minimum μ rule
(15)	0	use limited memory BFGS as Hessian (default)
	1	provide Hessian (see LINSLV, Section 3.2)

IFAIL	The parameter shows the reason for terminating a solution process. Initially, IFAIL must be set to zero. On return, IFAIL could contain the following values: -2 - Compute new gradient values. -1 - Compute new function values. 0 - Optimality conditions satisfied. 1 - not enough space in real work array 2 - not enough space in integer work array 4 - line search failed 5 - unboundedness of the objective or one of the constraints 6 - local infeasibility or MFCQ failure 17 - NaN detected 98 - maximum number of iterations reached 100 - no decent direction in merit function anything else - something bad happened
STEPP	Maximum step length, if IFAIL=-1. Use 1D0 if unsure.
WORK(LWORK)	Real working array
LWORK	Length of the real working array WORK. LWORK must be at least $45*N+37*M+6*P*N+2*P*M+4*P*P+1000$.
IWORK(LIWORK)	Integer working array
LIWORK	Length of the integer working array IWORK. LIWORK must be at least $N+M+150$.

Some of the termination reasons depend on the accuracy used for approximating gradients. If we assume that all functions and gradients are computed within machine precision and that the implementation is correct, there remain only 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 Hessian approximation 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 neighborhood 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, because of wrong or inaccurate gradients, the quadratic programming subproblem 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.

NLPIP returns the best iterate obtained. In case of successful termination (IFAIL=0), this is always the last iterate, in case of non-successful return (IFAIL>0) an eventually better previous iterate. The success is measured by objective function value and constraint violation.

3.2 LINSLV

3.2.1 Overview

The subroutine LINSLV has to be provided by the user. It will be called by NLPPI to do some calculations concerning the Jacobian and (if IPARAM(15)=1, see Section 3.1) the Hessian matrix. Let $u, u' \in \mathbb{R}^n$, $v \in \mathbb{R}^m$, $w \in \mathbb{R}$, $s \in \{-1, 1\}$ and $z, b \in \mathbb{R}^{n+m}$. Then these operations are

- Transpose of the Jacobian times vector

$$u \leftarrow s \cdot \nabla g(x)^T \cdot v + u \quad (7)$$

- Jacobian times vector

$$v \leftarrow s \cdot \nabla g(x) \cdot u + v \quad (8)$$

- Single row of the Jacobian times vector

$$w \leftarrow \nabla g_i(x) \cdot u \quad (9)$$

- Solve the linear system

$$\begin{pmatrix} H(x, y) + D_1 & \nabla g(x)^T \\ \nabla g(x) & -S\Lambda^{-1} - D_2 \end{pmatrix} \cdot z = b \quad (10)$$

for z where $H(x, y)$ is the Hessian of the Lagrangian or zero and D_1 and D_2 are positive semidefinite diagonal matrices.

- Hessian times vector (if user provided Hessian is used)

$$u' \leftarrow H(x, y) \cdot u$$

3.2.2 Operations for LINSLV in detail

The subroutine LINSLV must match the following interface:

```

SUBROUTINE LINSLV(MODE, M, NHRS, N, IP, DG, A, B)
  INTEGER          MODE, M, NRHS, N, IP
  DOUBLE PRECISION A, B
  DIMENSION       A(M+N), B(N+M,*)

```

The definition of DG is user specific und must match the definition of DG as given to NLPPI. It should contain the Jacobian $\nabla g(x)$.

The integer parameter MODE specifies the operation to execute. The input parameters M , N and DG are the same for all operations and will have the same value (and for DG also the same format) as given to NLPPI. Mathematically DG should be a matrix of dimension $M \times N$. So the first row of DG is the gradient of the first constraint. Note that all input parameters must NOT be modified except those that are also output parameters, of course.

MODE=1: Compute (7) with $s := IP$, $u := A$ and $v := B$. That is compute $IP \cdot DG^T \cdot B + A$ and store it in A .

Input:

- IP is an integer. It will be one or minus one.
- B is a double precision vector of dimension M
- A is a double precision vector of dimension N

Output:

- A is a double precision vector of dimension N

MODE=2: Compute (8) with $s := IP$, $v := A$ and $u := B$. That is compute $IP \cdot DG \cdot B + A$ and store it in A .

Input:

- IP is an integer. It will be one or minus one.
- B is a double precision vector of dimension N
- A is a double precision vector of dimension M

Output:

- A is a double precision vector of dimension M

MODE=3: Compute (9) with $i := IP$, $w := A$ and $u := B$. That is compute the product of the IP 'th row of DG times B and store the result in the first entry of A .

Input:

- IP is an integer indexing the row of DG
- B is a double precision vector of dimension N

Output:

- $A(1)$ is a double precision

MODE=4: Factorize the reduced KKT system (10)

$$C := \begin{pmatrix} H(x, y) + D_1 & \nabla g(x)^T \\ \nabla g(x) & -S\Lambda^{-1} - D_2 \end{pmatrix}$$

where $D_1 := \text{diag}(A)$ and $S\Lambda^{-1} + D_2 := \text{diag}(B)$. If $\text{IPARAM}(15)=1$, then H is the Hessian of the Lagrangian, else $H = 0$. The result should be stored internally.

Input:

- A is a double precision vector of dimension N
- B is a double precision vector of dimension M

Output:

- IP is an integer containing the error code. $IP = 0$ means success.

When factorizing the reduced KKT system, be careful that some of the entries of D_2 tend to zero, while others tend to infinity. Also some of the entries of D_1 may tend to infinity. The entries of both D_1 and D_2 are all nonnegative. But be sure to add the sign to the lower right part of C .

MODE=5: Same as for **MODE=4** but indicating that only the diagonal entries have changed. If you do not use a solver that can take advantage of the fact that H and $\nabla g(x)$ haven't changed with respect to the last factorization, just do the same as for **MODE=4**.

MODE=6: Solve the linear system (10)

$$Cz = b$$

where $b := B$ and store the result z in B . Use C from the previous call with $\text{MODE}=4$ or 5 .

Input/Output:

- B is a double precision vector of dimension $M + N$

MODE=7 Same as for $\text{MODE}=6$ but solve the linear system with multiple right hand sides.

Input:

- $NRHS$ number of right hand sides
- B is a double precision matrix of dimension $(M + N) \times NRHS$

Output:

- B is a double precision matrix of dimension $(M + N) \times NRHS$

MODE=8 Compute $H \cdot B$ and store it in A , where H is the user provided Hessian. This mode has to be implemented if $\text{IPARAM}(15)=1$, only.

Input:

- B is a double precision vector of dimension N

Output:

- A is a double precision vector of dimension N

3.2.3 Summary

The following operations have to be implemented by the user:

- MODE Specifies the operation. At least the following have to be provided:
- 1 - compute $A = IP \cdot DG^T \cdot B + A$
 - 2 - compute $A = IP \cdot DG \cdot B + A$
 - 3 - compute $A(1) = DG(IP) \cdot B$
 - 4 - factorize reduced KKT system
 - 5 - update diagonal entries of reduced KKT system
 - 6 - solve $CB = B$, where C is the reduced KKT matrix
 - 7 - same as 6, but B has $NRHS$ columns
 - 8 - compute $A = H \cdot B$ (for $\text{IPARAM}(15)=1$)

4 Example

To give an example how to organize the code, we consider the following problem:

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

The Fortran source code for executing NLPIP is listed below. Gradients are given in analytical form. We use a dense Jacobian and solve the KKT system with PARDISO[9]. Other examples that also demonstrate the use of sparse matrices are shipped together with the NLPIP software.

```
C   Sample for NLPIP
C   Using a dense Jacobian
C
C   Author (C):   B. Sachsenberg,
C                 Department of Computer Science,
C                 University of Bayreuth,
C                 D-95440 Bayreuth,
C                 Germany
C
PROGRAM TSTDNS
IMPLICIT NONE
INTEGER NMAX, MMAX, LMAX, MNN2X, LWA, LKWA, P
PARAMETER (NMAX=4, MMAX=2, P=3)
PARAMETER (MNN2X = MMAX+NMAX+NMAX+2,
/          LWA=45*NMAX+37*MMAX+6*P*NMAX+2*P*MMAX+4*P*P+1000,
/          LKWA=NMAX+MMAX+150)
INTEGER KWA(LKWA), N, ME, M, MAXIT, MAXFUN,
/       IOUT, IFAIL, I, J, NFUNC, IPARAM(20)
DOUBLE PRECISION X(NMAX), F, G(MMAX), DF(NMAX),
/       DG(MMAX, NMAX), Y(MNN2X), SL(MNN2X),
/       XL(NMAX), XU(NMAX), WA(LWA), ACC, ACCQP
LOGICAL ACTIVE(MNN2X)
C
C   Set some constants and initial values
C
IOUT = 6
ACC = 1.0D-11
ACCQP = 1.0D-13
MAXIT = 100
MAXFUN = 10
IPARAM(1) = 0
N = 3
M = 2
ME = 0
IFAIL = 0
NFUNC = 0
DO I=1, N
X(I) = 10.0D0
XL(I) = 0.0D0
XU(I) = 100.0D0
ENDDO
```

```

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=====
      NFUNC = NFUNC + 1
      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) = -1.0D0
      DG(1,2) = -2.0D0
      DG(1,3) = -2.0D0
      DG(2,1) = 1.0D0
      DG(2,2) = 2.0D0
      DG(2,3) = 2.0D0
C
C=====
4 CALL NLPIP(M,ME,MMAX,N,X,F,G,DF,DG,Y,SL,XL,XU,ACTIVE,
/          P,ACC,ACCQP,MAXFUN,MAXIT,2,IOUT,IPARAM,IFAIL,1D0,
/          WA,LWA,KWA,LKWA)
      IF (IFAIL.EQ.-1) GOTO 1
      IF (IFAIL.EQ.-2) GOTO 2
C
      WRITE(IOUT,1000) NFUNC
1000 FORMAT('  ***  Number of function calls: ',I3)
C
      STOP
      END

C *****
C *   LINSLV interface routine for NLPIP
C *****
      SUBROUTINE LINSLV(MODE, M, NRHS, N, IP, DG, A, B)
      IMPLICIT NONE
      INTEGER      MODE, M, NRHS, N, IP
      DOUBLE PRECISION DG, A, B

      INTEGER      NMAX,MMAX, LWA, LIWA
      PARAMETER (NMAX = 4, MMAX = 2,
/              LWA = 4*(NMAX+MMAX),
/              LIWA = 64 + 2*(NMAX+MMAX) + (NMAX+1)*(MMAX+1) )

      DIMENSION DG(MMAX,N), A((N+M)**2), B(M+N,*)

```

```

C=====
C      local parameters
      DOUBLE PRECISION WA
      INTEGER          IWA

      DIMENSION        WA(LWA), IWA(LIWA)
      SAVE              WA, IWA

      INTEGER          J
      DOUBLE PRECISION DDOT
      EXTERNAL         DDOT
C=====

      GOTO (1,2,3,4,5,6,7), MODE
C=====
C      Compute A = IP*DG'*B + A
1      CALL DGEMV('T', M, N, DBLE(IP), DG, MMAX, B, 1, 1D0, A, 1)
      RETURN
C=====
C      Compute A = IP*DG*B + A
2      CALL DGEMV('N', M, N, DBLE(IP), DG, MMAX, B, 1, 1D0, A, 1)
      RETURN
C=====
C      Compute A(1) = (IP'th row of DG)*B + A
3      CONTINUE
      A(1) = DDOT(N, DG(IP,1), MMAX, B, 1)
      RETURN
C=====
C      Factorize (full)
4      CONTINUE
      CALL LINS1(1, M, MMAX, N, A, B, DG, WA(2*(N+M)+1), IWA,
/          WA, WA(N+M+1), IP)
      RETURN
C=====
C      Factorize (diag update)
5      CONTINUE
      CALL LINS1(2, M, MMAX, N, A, B, DG, WA(2*(N+M)+1), IWA,
/          WA, WA(N+M+1), IP)
      RETURN
C=====
C      solve B=C\B
6      CONTINUE
      CALL LINS1(3, M, MMAX, N, A, B, DG, WA(2*(N+M)+1), IWA,
/          B, WA(N+M+1), IP)
      RETURN
C=====
C      solve B=C\B (multi)
7      CONTINUE
      DO 71 J = 1, NRHS
          CALL LINS1(3, M, MMAX, N, A, B, DG, WA(2*(N+M)+1), IWA,
/              B(1,J), WA(N+M+1), IP)
71     CONTINUE

```



```

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, N+M, A, P(IIA), P(IJA),
/
P(IPERM), 1, P, 0, B, X, IP)

RETURN

```

```

C=====
C      update diagonal entries
2      CONTINUE
      DO 21 I = 1,N
        A(1+(I-1)*(1+M)) = D1(I)
21     CONTINUE
      DO 22 I = 1,M
        A(N*(1+M)+I) = -D2(I)
22     CONTINUE

      CALL PARDISO (PT, 1, 1, -2, 22, N+M, A, P(IIA), P(IJA),
/
P(IPERM), 1, P, 0, B, X, IP)

RETURN

```

```

C=====
C      solve
3      CONTINUE

      CALL PARDISO (PT, 1, 1, -2, 33, N+M, A, P(IIA), P(IJA),
/
P(IPERM), 1, P, 0, B, X, IP)

      CALL DCOPY(N+M, X, 1, B, 1)

END

```

The output of the code above looks like

```

1 -0.100000D+04  0.00D+00  0.28604418D+02  0.00D+00  0.00D+00  0.00D+00
   4 iterations, KKT= 117.118414368897
2 -0.332733D+04  0.00D+00  0.29150282D+02  0.10D+01  0.20D+02  0.00D+00
   5 iterations, KKT= 7.057684528108553E-004
3 -0.339116D+04  0.00D+00  0.24235078D+01  0.10D+01  0.18D+01  0.00D+00
   4 iterations, KKT= 6.377532231633540E-013
4 -0.345456D+04  0.00D+00  0.26318341D-01  0.10D+01  0.47D+01 -0.93D+00
   7 iterations, KKT= 1.195635038454954E-009
5 -0.345599D+04  0.00D+00  0.19598223D-03  0.10D+01  0.65D+00  0.10D+01
   6 iterations, KKT= 1.117724952147405E-014
6 -0.345600D+04  0.00D+00  0.26529912D-07  0.10D+01  0.52D-01 -0.10D+01
   7 iterations, KKT= 4.364672587784021E-013
7 -0.345600D+04  0.00D+00  0.31853785D-13  0.10D+01  0.61D-03 -0.10D+01
Solution found 23.9999994113019 , Objective= -3456.000000000000
Constraints 0.000000000000000E+000
*** Number of function calls: 7

```

Odd lines show information about the outer SQP loop while even lines show the output of the QP solver. For the SQP output the first column is the iteration number, the second column shows the current value of the objective function. Next comes the violation of the constraints and then the violation of the KKT conditions. The fifth column shows the step size α accepted by the line search. Here one means a full step. The last two columns show the length, i.e. the euclidean norm of the step and the cosine of the angle between the last step and the step before.

The output of the QP solver shows the number of inner iterations and the violation of the KKT conditions on return. So as you can see, in the first two steps the QP is not solved to the desired accuracy. While this may sound to you like a problem, the opposite is the case. Let's compare it with the output of an ordinary SQP algorithm, shown below.

```

1  -0.100000D+04  0.00D+00  0.28604418D+02  0.00D+00  0.00D+00  0.00D+00
    11 iterations, KKT=  2.149682087073320E-014
2  -0.161453D+04  0.00D+00  0.37886275D+02  0.44D+00  0.48D+02  0.00D+00
    10 iterations, KKT=  3.183400714976076E-016
3  -0.332958D+04  0.00D+00  0.14969005D+02  0.10D+01  0.49D+01  0.00D+00
    9 iterations, KKT=  4.617077472594428E-017
4  -0.341522D+04  0.00D+00  0.19810083D+00  0.10D+01  0.26D+01  -0.43D+00
    8 iterations, KKT=  1.520444086529632E-013
5  -0.345516D+04  0.00D+00  0.15264787D-01  0.10D+01  0.37D+01  0.56D+00
    6 iterations, KKT=  8.599416866031812E-014
6  -0.345600D+04  0.00D+00  0.68930161D-04  0.10D+01  0.49D+00  0.10D+01
    7 iterations, KKT=  3.275305496412512E-013
7  -0.345600D+04  0.00D+00  0.54633252D-08  0.10D+01  0.31D-01  -0.10D+01
    7 iterations, KKT=  4.453168858396293E-013
8  -0.345600D+04  0.00D+00  0.58556036D-14  0.10D+01  0.28D-03  -0.10D+01
Solution found      23.9999998416988      , Objective=  -3456.000000000000
Constraints  0.0000000000000000E+000
*** Number of function calls:  9

```

It takes one more iteration and two more function calls to achieve the desired accuracy. One more iteration means one more evaluation of the gradients and function values of the objective and the constraints. The second additional function call comes from the fact that for the first step the line search does not accept step size one. Also we need more than twice the number of inner iterations. Since each inner iteration consists mainly of factorizing the reduced KKT system this can be significant for large scale problems.

5 Numerical Results

5.1 Elliptic optimal control problems by [5, 6]

Parameters for NLPIP:

- Monotone update of μ , starting with $\mu = 0.1$.
- One iteration for each QP (standard interior point method for NLP)
- Flexible penalty function [2] with regularization by [1]

In Table 4 on page 17 we show the results for $N + 1 = 20$ (see [5, 6]) of NLPIP and compare them with the results of NLPQLP [11] and SCPIP30 [14]. The results for $N + 1 = 100$ to 1600 are shown in Tables 5 to 8.

6 Conclusion

We presented an SQP interior point solver for large scale nonlinear optimization. It is the mixture of SQP and interior point methods that makes it different to other available solvers. Although it does not yet use a real heuristic for determining the best accuracy for solving the quadratic problems it performs quite well.

References

- [1] L. Chen and D. Goldfarb. Interior-point l_2 -penalty methods for nonlinear programming with strong global convergence properties. *Mathematical Programming*, 108:1–36, 2006.
- [2] F.E. Curtis and J. Nocedal. Flexible penalty functions for nonlinear constrained optimization. *IMA Journal of Numerical Analysis*, 28:749–769, 2008.
- [3] N.I.M. Gould, D. Orban, and Ph. L. Toint. Numerical methods for large-scale nonlinear optimization. Technical Report RAL-TR-2004-032, CCLRC, 2004.
- [4] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.
- [5] H. Maurer and H.D. Mittelmann. Optimization techniques for solving elliptic control problems with control and state constraints: Part 1. boundary control. *Computational Optimization and Applications*, 16:29–55, 2000.
- [6] H. Maurer and H.D. Mittelmann. Optimization techniques for solving elliptic control problems with control and state constraints. part 2: Distributed control. *Computational Optimization and Applications*, 18:141–160, 2001.
- [7] M. J. D. Powell. *A fast algorithm for nonlinearly constraint optimization calculations*, volume 630 of *Lecture Notes in Mathematics*. Springer, 1978.
- [8] B. Sachsenberg. An sqp interior point algorithm for solving large scale nonlinear optimization problems, 2008.
- [9] O. Schenk and K. Gärtner. On fast factorization pivoting methods for symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.
- [10] Klaus Schittkowski. The nonlinear programming method of Wilson, Han and Powell. Part 2: An efficient implementation with linear least squares subproblems. *Numerische Mathematik*, 38:115–127, 1981.
- [11] Klaus Schittkowski. Nlpqlp: A fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search - user’s guide, version 2.24 -. Technical report, University of Bayreuth, 2007.
- [12] J. Stoer. *Foundations of recursive quadratic programming methods for solving nonlinear programs*, volume 15 of *Nato ASI Series*. Springer, 1985.
- [13] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107:391–408, 2006.
- [14] Christian Zillober. Software manual for SCPIP 3.0. Technical report, University of Bayreuth, 2004.

Prob	n	m	Fun	Grad	Objective	Constraints	Time	Code	Error
EX 1	437	361	21	21	0.16449534D+00	0.29282132D-14	0.5609D+01	NLPIP	0
EX 1	437	361	26	25	0.16449540D+00	0.29282132D-14	0.7922D+01	NLPQLP	0
EX 1	437	361	17	17	0.16449534D+00	0.46213033D-14	0.1562D+00	SCPIP30	0
EX 2	437	361	30	29	0.68610327D-01	0.28449465D-14	0.1003D+02	NLPIP	0
EX 2	437	361	33	32	0.68612217D-01	0.33723024D-14	0.9672D+01	NLPQLP	0
EX 2	437	361	40	40	0.68610327D-01	0.50653925D-14	0.4219D+00	SCPIP30	0
EX 3	437	361	21	21	0.26666575D+00	0.29282132D-14	0.6438D+01	NLPIP	0
EX 3	437	361	38	36	0.26666575D+00	0.28449465D-14	0.1175D+02	NLPQLP	0
EX 3	437	361	26	26	0.26666575D+00	0.98695781D-09	0.2812D+00	SCPIP30	0
EX 4	437	361	20	20	0.19843615D+00	0.28449465D-14	0.6000D+01	NLPIP	0
EX 4	437	361	31	29	0.19843614D+00	0.24841240D-14	0.9188D+01	NLPQLP	0
EX 4	437	361	25	25	0.19843614D+00	0.46213033D-14	0.3281D+00	SCPIP30	0
EX 5	513	437	23	23	0.53484164D+00	0.28501194D-11	0.5953D+01	NLPIP	0
EX 5	513	437	72	49	0.62168210D+01	0.17763568D-14	0.5614D+02	NLPQLP	0
EX 5	513	437	25	25	0.53484164D+00	0.35101921D-10	0.2188D+00	SCPIP30	0
EX 6	513	437	25	25	0.11266829D-01	0.24183711D-11	0.7000D+01	NLPIP	0
EX 6	513	437	40	28	0.10238476D+02	0.26645353D-14	0.3602D+02	NLPQLP	0
EX 6	513	437	501	501	NaN	NaN	0.1516D+01	SCPIP30	1
EX 7	513	437	22	22	0.23870230D+00	0.41146947D-12	0.6281D+01	NLPIP	0
EX 7	513	437	43	42	0.23870230D+00	0.28310687D-14	0.3991D+02	NLPQLP	0
EX 7	513	437	33	33	0.23870230D+00	0.41674650D-11	0.3281D+00	SCPIP30	0
EX 8	513	437	20	20	0.13348283D+00	0.41876225D-14	0.5469D+01	NLPIP	0
EX 8	513	437	35	34	0.13348282D+00	0.42015003D-14	0.3262D+02	NLPQLP	0
EX 8	513	437	25	25	0.13348297D+00	0.74285751D-11	0.2969D+00	SCPIP30	0
EX 9	722	361	24	24	0.45903106D-01	0.50294290D-12	0.7219D+01	NLPIP	0
EX 9	722	361	71	71	0.45913897D-01	0.74100378D-12	0.7848D+02	NLPQLP	0
EX 9	722	361	13	13	0.45903100D-01	0.14005521D-11	0.1719D+00	SCPIP30	0
EX 10	722	361	23	23	0.40391009D-01	0.97019831D-14	0.7625D+01	NLPIP	0
EX 10	722	361	295	295	0.40430705D-01	0.23703153D-13	0.3620D+03	NLPQLP	0
EX 10	722	361	6	6	0.40390974D-01	0.72576493D-15	0.7812D-01	SCPIP30	0
EX 11	722	361	24	24	0.11009563D+00	0.60877301D-12	0.7016D+01	NLPIP	0
EX 11	722	361	365	365	0.11009751D+00	0.20269997D-10	0.4553D+03	NLPQLP	0
EX 11	722	361	17	17	0.11009561D+00	0.75791933D-12	0.1875D+00	SCPIP30	0
EX 12	798	437	29	28	0.75833428D-01	0.15052889D-12	0.9547D+01	NLPIP	0
EX 12	798	437	401	400	0.75850187D-01	0.95100269D-11	0.8303D+03	NLPQLP	0
EX 12	798	437	24	24	0.75833414D-01	0.11576668D-11	0.3281D+00	SCPIP30	0
EX 13	798	437	31	31	0.51365432D-01	0.50922287D-12	0.1153D+02	NLPIP	0
EX 13	798	437	501	500	0.53642251D-01	0.29138332D-09	0.1087D+04	NLPQLP	1
EX 13	798	437	41	41	0.51365397D-01	0.11419346D-11	0.7344D+00	SCPIP30	0

Table 4: Examples with $N + 1 = 20$

Prob	n	m	Fun	Grad	Objective	Constraints	Time	Code	Error
EX 1	10197	9801	20	20	0.19652520D+00	0.37730236D-14	0.1100D+03	NLPIP	0
EX 2	10197	9801	34	32	0.96695067D-01	0.37730236D-14	0.2582D+03	NLPIP	0
EX 3	10197	9801	24	24	0.32100968D+00	0.33324038D-14	0.1636D+03	NLPIP	0
EX 4	10197	9801	24	24	0.24917851D+00	0.37730236D-14	0.1708D+03	NLPIP	0
EX 5	10593	10197	26	26	0.55224596D+00	0.33214294D-09	0.1223D+03	NLPIP	0
EX 6	10593	10197	34	34	0.15078703D-01	0.10683104D-11	0.2032D+03	NLPIP	0
EX 7	10593	10197	22	22	0.26389846D+00	0.11965411D-11	0.1073D+03	NLPIP	0
EX 8	10593	10197	23	23	0.16166397D+00	0.13631891D-13	0.1347D+03	NLPIP	0
EX 9	19602	9801	22	22	0.62161757D-01	0.31716920D-12	0.1498D+03	NLPIP	0
EX 10	19602	9801	24	24	0.56448395D-01	0.23207009D-15	0.1937D+03	NLPIP	0
EX 11	19602	9801	23	23	0.11026341D+00	0.65267480D-13	0.1670D+03	NLPIP	0
EX 12	19998	10197	22	22	0.78064158D-01	0.59996818D-12	0.1500D+03	NLPIP	0
EX 13	19998	10197	37	37	0.52664771D-01	0.74898976D-13	0.5457D+03	NLPIP	0

Table 5: Examples with $N + 1 = 100$

Prob	n	m	Fun	Grad	Objective	Constraints	Time	Code	Error
EX 1	1442397	1437601	170	167	0.20433700D+00	0.14001916D-12	0.1830D+05	NLPIP	0
EX 2	1442397	1437601	25	25	0.10359844D+00	0.45129809D-10	0.2263D+04	NLPIP	0
EX 3	1442397	1437601	64	64	0.33409908D+00	0.41278592D-14	0.7150D+04	NLPIP	0
EX 4	1442397	1437601	40	40	0.26151218D+00	0.61933992D-11	0.5275D+04	NLPIP	0
EX 5	1447193	1442397	78	78	0.55608702D+00	0.65699490D-09	0.6354D+04	NLPIP	0
EX 6	1447193	1442397	79	79	0.16046697D-01	0.77635539D-09	0.6645D+04	NLPIP	0
EX 7	1447193	1442397	40	40	0.26982982D+00	0.92789822D-09	0.3075D+04	NLPIP	0
EX 8	1447193	1442397	75	75	0.16933958D+00	0.56737109D-09	0.5946D+04	NLPIP	0
EX 9	2875202	1437601	30	30	0.66358139D-01	0.22426311D-11	0.5316D+04	NLPIP	0
EX 10	2875202	1437601	31	31	0.60628103D-01	0.20131818D-12	0.5737D+04	NLPIP	0
EX 11	2875202	1437601	26	26	0.11030773D+00	0.52040230D-10	0.4628D+04	NLPIP	0
EX 12	2879998	1442397	27	27	0.78754225D-01	0.67735631D-11	0.3932D+04	NLPIP	0
EX 13	2879998	1442397	80	80	0.53314886D-01	0.16106288D-13	0.2753D+05	NLPIP	0

Table 6: Examples with $N + 1 = 1200$

Prob	n	m	Fun	Grad	Objective	Constraints	Time	Code	Error
EX 1	1962797	1957201	77	72	0.20443913D+00	0.23014356D-12	0.1561D+05	NLPIP	0
EX 2	1962797	1957201	29	29	0.10368833D+00	0.22176671D-09	0.6668D+04	NLPIP	0
EX 3	1962797	1957201	162	162	0.33427205D+00	0.15433154D-11	0.3347D+05	NLPIP	0
EX 4	1962797	1957201	46	46	0.26167567D+00	0.12301166D-10	0.1323D+05	NLPIP	0
EX 5	1968393	1962797	45	45	0.55624351D+00	0.24347013D-09	0.8454D+04	NLPIP	0
EX 6	1968393	1962797	111	93	0.16078471D-01	0.17338300D-09	0.1798D+05	NLPIP	0
EX 7	1968393	1962797	33	33	0.26994035D+00	0.34604231D-10	0.5977D+04	NLPIP	0
EX 8	1968393	1962797	49	49	0.16944687D+00	0.24494548D-09	0.8622D+04	NLPIP	0
EX 9	3914402	1957201	32	32	0.66436920D-01	0.18239817D-11	0.1596D+05	NLPIP	0
EX 10	3914402	1957201	34	34	0.60820990D-01	0.46097958D-13	0.1816D+05	NLPIP	0
EX 11	3914402	1957201	28	28	0.11031971D+00	0.20185128D-10	0.1242D+05	NLPIP	0
EX 12	3919998	1962797	30	30	0.78749823D-01	0.38579447D-11	0.1366D+05	NLPIP	0
EX 13	3919998	1962797	45	45	0.53942550D-01	0.17888781D-14	0.3243D+05	NLPIP	0

Table 7: Examples with $N + 1 = 1400$

Prob	n	m	Fun	Grad	Objective	Constraints	Time	Code	Error
EX 1	2563197	2556801	122	117	0.20451582D+00	0.67475625D-12	0.5059D+05	NLPIP	0
EX 2	2563197	2556801	29	29	0.10375304D+00	0.76996028D-09	0.1261D+05	NLPIP	0
EX 3	2563197	2556801	198	198	0.33439919D+00	0.10398793D-13	0.5826D+05	NLPIP	0
EX 4	2563197	2556801	47	47	0.26179728D+00	0.18435735D-10	0.2283D+05	NLPIP	0
EX 5	2569593	2563197	46	46	0.55633044D+00	0.16222268D-09	0.1508D+05	NLPIP	0
EX 6	2569593	2563197	79	76	0.16099869D-01	0.17766508D-09	0.2357D+05	NLPIP	0
EX 7	2569593	2563197	37	37	0.26999987D+00	0.18075206D-10	0.1103D+05	NLPIP	0
EX 8	2569593	2563197	53	53	0.16952646D+00	0.72746990D-10	0.1813D+05	NLPIP	0
EX 9	5113602	2556801	35	35	0.66463145D-01	0.27343599D-12	0.2954D+05	NLPIP	0
EX 10	5113602	2556801	36	36	0.60906059D-01	0.15587914D-11	0.3100D+05	NLPIP	0
EX 11	5113602	2556801	29	29	0.11031695D+00	0.74447046D-11	0.1994D+05	NLPIP	0
EX 12	5119998	2563197	30	30	0.78759468D-01	0.26701419D-11	0.2020D+05	NLPIP	0
EX 13	5119998	2563197	80	80	0.53391525D-01	0.26432830D-14	0.1032D+06	NLPIP	0

Table 8: Examples with $N + 1 = 1600$