Institute of Computer Science
Academy of Sciences of the Czech Republic

Subroutines for large-scale nonlinear programming

Ladislav Lukšan, Ctirad Matonoha, Jan Vlček

Technical report No. V1000

June 30, 2007

Pod Vodárenskou věží 2, 182 07 Prague 8 phone: +420 2 688 42 44, fax: +420 2 858 57 89,
e-mail:e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Subroutines for large-scale nonlinear programming

Ladislav Lukšan, Ctirad Matonoha, Jan Vlček [1]

Technical report No. V1000

June 30, 2007

Abstract:

We present two basic FORTRAN subroutines for large-scale nonlinear programming. Subroutines `PIND` and `PNUL`, intended for sparse equality constrained nonlinear programming problems, are based on the indefinitely preconditioned conjugate gradient method applied to the linear KKT system or to the reduced system obtained by a null-space approach. Besides the description of methods and codes, we propose computational experiments which demonstrate the efficiency of the proposed algorithms.

Keywords:
Large-scale optimization, large-scale nonlinear programming, sparse problems, partially separable problems, discrete Newton methods, KKT systems, indefinite preconditioners.

[1]This work was supported by the Grant Agency of the Czech Academy of Sciences, project No. IAA1030405, the Grant Agency of the Czech Republic, project No. 201/06/P397, and the institutional research plan No. AV0Z10300504

1 Introduction

We propose two basic subroutines which implement selected large-scale nonlinear programming algorithms. The double-precision FORTRAN 77 subroutines PIND, PNUL are designed to find a close approximation to a local minimum of a general twice continuously differentiable function $F : R^n \to R$ under equality constraints

$$c_i(x) = 0, \qquad 1 \le i \le n_c.$$

Here $x \in R^n$ is a vector of n variables and $c_i : R^n \to R$, $1 \le i \le n_c \le n$, are twice continuously differentiable functions. Subroutines PIND, PNUL are based on the inexact discrete Newton method applied to nonlinear KKT equations. Subroutine PIND uses the indefinitely preconditioned conjugate gradient method for solving the indefinite linear KKT system [2], [3]. Subroutine PNUL uses null-space transformations and the standard conjugate gradient method applied to a reduced system [1], [3].

To simplify the user's work, additional easy-to-use subroutines are added. These subroutines call general subroutines PIND, PNUL:

PINDU,PNULU – optimization with general equality constraints.

Each subroutine contains a description of formal parameters and extensive comments. Moreover, text files PIND.TXT, PNUL.TXT containing a detailed description of all important subroutines (including indications of required storage) are added. Finally, test programs TINDU, TNULU containing sets of test problems are included. These test programs serve as examples for using the subroutines, verify their correctness and demonstrate their efficiency.

2 Inexact discrete Newton methods for equality constrained nonlinear programming problems

Consider a general twice continuously differentiable function $F : R^n \to R$ and a twice continuously differentiable mapping $c : R^n \to R^{n_c}$ and assume that the Hessian matrix of F and the Jacobian matrix of c are both sparse. In this case, discrete versions of the Newton method can be efficiently used for seeking a local minimum of F on the manifold defined by equality constraints $c(x) = 0$. The sparsity pattern of the Hessian matrix (only the upper part) is stored in the standard compressed row format using arrays IH and JH. For example, if the Hessian matrix has the pattern

$$G = \begin{bmatrix} * & * & * & 0 & * \\ * & * & 0 & * & 0 \\ * & 0 & * & 0 & * \\ 0 & * & 0 & * & 0 \\ * & 0 & * & 0 & * \end{bmatrix}$$

(asterisks denote nonzero elements), then arrays IH and JH contain elements

$$\mathtt{IH} = \begin{bmatrix} 1 & 5 & 7 & 9 & 10 & 11 \end{bmatrix}, \qquad \mathtt{JH} = \begin{bmatrix} 1 & 2 & 3 & 5 & 2 & 4 & 3 & 5 & 4 & 5 \end{bmatrix},$$

i.e., IH contains pointers of the diagonal elements in the upper part of the Hessian matrix and JH contains column indices of the nonzero elements stored. Note that IH has $n+1$ elements and the last element is equal to $m+1$, where m is the number of elements stored.

The sparsity pattern of the Jacobian matrix is stored in the standard compressed row format using arrays ICG and JCG. For example, if the Jacobian matrix has the pattern

$$J = \begin{bmatrix} * & * & 0 & * \\ * & * & * & 0 \\ * & 0 & 0 & * \\ 0 & * & * & 0 \\ * & 0 & * & 0 \end{bmatrix}$$

(asterisks denote nonzero elements) then arrays ICG and JCG contain elements

$$\texttt{ICG} = \begin{bmatrix} 1 & 4 & 7 & 9 & 11 & 13 \end{bmatrix}, \qquad \texttt{JCG} = \begin{bmatrix} 1 & 2 & 4 & 1 & 2 & 3 & 1 & 4 & 2 & 3 & 1 & 3 \end{bmatrix},$$

i.e., ICG contains pointers of the first elements in rows of the Jacobian matrix and JCG contains column indices of the nonzero elements. Note that ICG has n_c+1 elements and the last element is equal to m_c+1, where m_c is the number of nonzero elements.

Applying the Newton method to the system

$$\begin{aligned} \nabla F(x) + A(x)u &= 0, \\ c(x) &= 0 \end{aligned}$$

of $n+n_c$ nonlinear equations for unknown vectors $x \in R^n$ and $u \in R^{n_c}$ (first-order necessary conditions), where $A(x)$ is the Jacobian matrix of $c(x)$, we obtain the iterative process

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k d_k^x, \\ u_{k+1} &= u_k + \alpha_k d_k^u, \end{aligned}$$

where d_k^x, d_k^u are direction vectors obtained by solving the linear KKT system

$$\begin{bmatrix} G(x_k,u_k) & A(x_k) \\ A(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^u \end{bmatrix} = - \begin{bmatrix} g(x_k,u_k) \\ c(x_k) \end{bmatrix}, \tag{1}$$

and $\alpha_k > 0$ is a scalar step-size. Here

$$g(x,u) = \nabla F(x) + \sum_{i=1}^{m} u_i \nabla c_i(x), \qquad G(x,u) = \nabla^2 F(x) + \sum_{i=1}^{m} u_i \nabla^2 c_i(x)$$

is the gradient and the Hessian matrix of the Lagrangian function $L(x,u) = F(x) + u^T c(x)$, respectively.

Various penalty functions can be used for obtaining the step-size α_k. We use the augmented Lagrangian function

$$P_k(\alpha) = F(x_k + \alpha d_k^x) + (u_k + d_k^u)^T c(x_k + \alpha d_k^x) + \frac{\sigma}{2} \| c(x_k + \alpha d_k^x) \|^2 \tag{2}$$

in our subroutines, where $\sigma \ge 0$ is a penalty parameter (parameter RPF1 in subroutines PIND and PNUL). It can be shown (see [2]) that if system (1) is solved in such a way that $\|G_k d_k^x + A_k d_k^u + g_k\| \le \overline{\omega}_k \|g_k\|$ and $\|A_k^T d_k^x + c_k\| \le \overline{\omega}_k \|c_k\|$ with $0 < \overline{\omega}_k < 1$ and if $\sigma > -(d_k^x)^T G_k d_k^x / ((1-\overline{\omega}_k)\|c_k\|)$, then $P_k'(0)$ (the first order derivative of $P_k(\alpha)$ at $\alpha = 0$) is negative so that $P_k(\alpha)$ decreases in the direction d_k^x. By our experience, it is not advantageous to recompute σ in every iteration. A more efficient way is to use a constant value $\sigma \ge 0$ and replace G_k by a diagonal positive definite matrix D_k (restart) if $P_k'(0) \ge 0$. Using this D_k for the construction of an indefinite preconditioner (see Section 2.1) we obtain the exact solution of (1) in the first iteration of an inner Krylov-subspace method and, moreover, $(d_k^x)^T D_k d_k^x > 0$. Thus $P_k'(0) < 0$ holds for any value $\sigma \ge 0$. This procedure allows us to choose sufficiently small values of σ, which decreases the Maratos-like effects (step-size reduction) considerably. Assuming $P_k'(0) < 0$, the step-size α_k is chosen in such a way that it is the first member of the sequence α_k^j, $j \in N$, where $\alpha_k^1 = 1$ and $\underline{\beta}\alpha_k^j \le \alpha_k^{j+1} \le \overline{\beta}\alpha_k^j$ with $0 < \underline{\beta} \le \overline{\beta} < 1$, satisfying

$$P(\alpha_k) - P(0) \le \varepsilon_1 \alpha_k P_k'(0)$$

with the line search parameter $0 < \varepsilon_1 < 1/2$ (parameter `TOLS` in subroutines `PIND` and `PNUL`). We use the values $\underline{\beta} = 0.1$ and $\overline{\beta} = 0.9$ in our subroutines. The value α_k^{j+1} can be chosen either by a bisection (`MES` $= 1$) or by a two-point quadratic interpolation (`MES` $= 2$) or by a three-point quadratic interpolation (`MES` $= 3$) or by a three-point cubic interpolation (`MES` $= 4$) (`MES` is a parameter of subroutines `PIND` and `PNUL`).

To simplify the description of individual methods for computing direction vectors, we omit the outer index k and denote the inner index by i. Thus the linear KKT system (1) can be written in the form

$$Kd = \begin{bmatrix} B & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} d^x \\ d^u \end{bmatrix} = \begin{bmatrix} b^x \\ b^u \end{bmatrix} = b, \tag{3}$$

where B is an approximation of $G(x,u)$ computed by using differences of gradients of the Lagrangian function. In this case, the method based on graph coloring problem described in [4] is used.

2.1 Methods utilizing indefinite preconditioners

In subroutine `PIND`, directions d^x and d^u are computed directly from linear KKT system (3) by the preconditioned conjugate gradient method with the indefinite preconditioner

$$C = \begin{bmatrix} D & A \\ A^T & 0 \end{bmatrix} \tag{4}$$

investigated in [2]. Here D is a positive definite diagonal matrix derived from the diagonal of B. Multiplication by C^{-1} can be expressed in the form

$$\begin{aligned} C^{-1}r &= \begin{bmatrix} D^{-1} - D^{-1}A(A^TD^{-1}A)^{-1}A^TD^{-1} & D^{-1}A(A^TD^{-1}A)^{-1} \\ (A^TD^{-1}A)^{-1}A^TD^{-1} & -(A^TD^{-1}A)^{-1} \end{bmatrix} \begin{bmatrix} r^x \\ r^u \end{bmatrix} \\ &= \begin{bmatrix} D^{-1}(r^x - At^u) \\ t^u \end{bmatrix}, \qquad t^u = (A^TD^{-1}A)^{-1}(A^TD^{-1}r^x - r^u). \end{aligned} \tag{5}$$

More precisely, we set $d_1 = 0$, $r_1 = b$,

$$t_1^u = (A^TD^{-1}A)^{-1}(A^TD^{-1}r_1^x - r_1^u), \qquad t_1^x = D^{-1}(r_1^x - At_1^u),$$

$p_1 = t_1$, and for $i = 1, 2, 3, \dots$ we proceed in the following way. If $\|r_i^x\| \le \overline{\omega}\|b^x\|$ and $\|r_i^u\| \le \overline{\omega}\|b^u\|$, where $\overline{\omega}$ is a precision used, we set $d = d_i$ and terminate the computation. Otherwise compute

$$\begin{aligned}
q_i &= K p_i, \qquad \alpha_i = r_i^T t_i / p_i^T q_i, \\
d_{i+1} &= d_i + \alpha_i p_i, \qquad r_{i+1} = r_i - \alpha_i q_i, \\
t_{i+1}^u &= (A^T D^{-1} A)^{-1} (A^T D^{-1} r_{i+1}^x - r_{i+1}^u), \\
t_{i+1}^x &= D^{-1} (r_{i+1}^x - A t_{i+1}^u), \\
\beta_i &= r_{i+1}^T t_{i+1} / r_i^T t_i, \qquad p_{i+1} = t_{i+1} + \beta_i p_i.
\end{aligned}$$

In these relations, we use vectors

$$r = \begin{bmatrix} r^x \\ r^u \end{bmatrix}, \quad t = \begin{bmatrix} t^x \\ t^u \end{bmatrix}.$$

In the above algorithm, a multiplication by the matrix $(A^T D^{-1} A)^{-1}$ is used. This matrix is not computed explicitly, but the sparse Choleski decomposition is used instead. Unfortunately, matrix $A^T D^{-1} A$ can be dense if A has dense rows. To eliminate this situation, we assume that $A^T = [A_s^T, A_d^T]$ and $D = \mathrm{diag}(D_s, D_d)$, where $A_s^T D_s^{-1} A_s$ is sparse and A_d consists of dense rows. Then

$$\begin{aligned}
(A^T D^{-1} A)^{-1} &= (A_s^T D_s^{-1} A_s + A_d^T D_d^{-1} A_d)^{-1} \\
&= (A_s^T D_s^{-1} A_s)^{-1} - (A_s^T D_s^{-1} A_s)^{-1} A_d^T M_d^{-1} A_d (A_s^T D_s^{-1} A_s)^{-1},
\end{aligned}$$

where

$$M_d = D_d + A_d (A_s^T D_s^{-1} A_s)^{-1} A_d^T$$

is a (low-dimensional) dense matrix. Again the sparse Choleski decomposition is used instead of $(A_s^T D_s^{-1} A_s)^{-1}$. To realize this elimination effectively, we need to define the maximum number of dense rows (parameter `MD` in subroutine `PIND`) and the maximum number of elements in sparse rows (parameter `MDE` in subroutine `PIND`).

2.2 Methods based on null-space transformation

Consider the unique representation $d^x = Z d^z + D^{-1} A d^a$, where

$$d^z = (Z^T D Z)^{-1} Z^T D d^x, \qquad d^a = (A^T D^{-1} A)^{-1} A^T d^x$$

and where Z is a matrix whose columns form an orthogonal basis in the subspace of vectors $v \in R^n$ satisfying the equation $A^T v = 0$ (thus $Z^T A = 0$). Using the second equation in (3), we get $d^a = (A^T D^{-1} A)^{-1} b^u$ and the first one implies $B Z d^z = b^x - B D^{-1} A d^a - A d^u$, which after premultiplying by Z^T gives

$$Z^T B Z d^z = b^z, \tag{6}$$

where $b^z = Z^T (b^x - B D^{-1} A d^a)$. Thus d^z can be obtained by the conjugate gradient method with the preconditioner $Z^T D Z$ applied to equation (6). Unfortunately, the matrix Z is not usually known (its computation is time-consuming and difficult for large sparse A

because of fill-in). For this reason, we use a modification proposed in [1]. In subroutine PNUL, the conjugate gradient iterations are modified in such a way that they use vectors $\tilde{d} = Zd^z$, $\tilde{t} = Zt^z$, $\tilde{p} = Zp^z$ and vectors $\tilde{r}$, $\tilde{q}$ such that $r^z = Z^T\tilde{r}$, $q^z = Z^T\tilde{q}$. After this transformation, the multiplication $t^z = (Z^TDZ)^{-1}r^z$ can be replaced by the formula

$$\tilde{t} = Z(Z^TDZ)^{-1}Z^T\tilde{r} = (D^{-1} - D^{-1}A(A^TD^{-1}A)^{-1}A^TD^{-1})\tilde{r}$$

so that the matrix Z need not be used explicitly. Since $\tilde{d}$ does not influence formulas in conjugate gradient iterations directly, we can use $d^x = \tilde{d} + D^{-1}Ad^a$ instead of $\tilde{d}$. Since d^u cannot be obtained from conjugate gradient iterations, it has to be estimated in another way. The most natural choice is the weighted least-square minimization of the total residual $\tilde{r} - Ad^u$. This choice leads to the formula $d^u = (A^TD^{-1}A)^{-1}A^TD^{-1}\tilde{r}$.

The above considerations form a basis for an efficient preconditioned conjugate gradient algorithm. More precisely, we set $d_1^x = D^{-1}A(A^TD^{-1}A)^{-1}b^u$, $\tilde{r}_1 = b^x - Bd_1^x$, $d_1^u = (A^TD^{-1}A)^{-1}A^TD^{-1}\tilde{r}_1$, $\tilde{t}_1 = D^{-1}(\tilde{r}_1 - Ad_1^u)$, $\gamma = \tilde{r}_1^T\tilde{t}_1$, $\tilde{p}_1 = \tilde{t}_1$ and for $i = 1, 2, 3, \ldots$ we proceed in the following way. If $\tilde{r}_i^T\tilde{t}_i \le \overline{\omega}\gamma$, where $\overline{\omega}$ is a precision used, we set $d^x = d_i^x$, $d^u = d_i^u$ and terminate the computation. Otherwise compute

$$\begin{aligned}
\tilde{q}_i &= B\tilde{p}_i, \qquad \alpha_i = \tilde{r}_i^T\tilde{t}_i/\tilde{p}_i^T\tilde{q}_i,\\
d_{i+1}^x &= d_i^x + \alpha_i\tilde{p}_i, \qquad \tilde{r}_{i+1} = \tilde{r}_i - \alpha_i\tilde{q}_i,\\
d_{i+1}^u &= (A^TD^{-1}A)^{-1}A^TD^{-1}\tilde{r}_{i+1},\\
\tilde{t}_{i+1} &= D^{-1}(\tilde{r}_{i+1} - Ad_{i+1}^u),\\
\beta_i &= \tilde{r}_{i+1}^T\tilde{t}_{i+1}/\tilde{r}_i^T\tilde{t}_i, \qquad \tilde{p}_{i+1} = \tilde{t}_{i+1} + \beta_i\tilde{p}_i.
\end{aligned}$$

In these computations, a multiplication by the matrix $(A^TD^{-1}A)^{-1}$ is used again. Thus we proceed in the same way as in Section 2.1 if A has dense rows (subroutine PNUL also uses parameters MD and MDE) .

3 Description of subroutines

In this section we describe easy-to-use subroutines PINDU, PNULU which can be called from the user's program. In the description of formal parameters we introduce a type of the argument denoted by two letters. The first letter is either I for integer arguments or R for double-precision real arguments. The second letter specifies whether the argument must have a value defined on the entry to the subroutine (I), whether it is a value which will be returned (O), or both (U), or whether it is an auxiliary value (A). Besides the formal parameters, we use a COMMON /STAT/ block containing statistical information. This block, used in each subroutine, has the following form:

```
COMMON /STAT/ NRES,NDEC,NIN,NIT,NFV,NFG,NFH
```

Its elements have the following meanings:

Element	Type	Significance
NRES	IO	Number of restarts.
NDEC	IO	Number of matrix decompositions.
NIN	IO	Number of inner iterations (for solving linear systems).

`NIT`	`IO`	Number of iterations.
`NFV`	`IO`	Number of function evaluations.
`NFG`	`IO`	Number of gradient evaluations.
`NFH`	`IO`	Number of Hessian evaluations.

Easy-to-use subroutines are called by the following statements:

```
  CALL PINDU(NF,NC,X,IH,JH,CF,ICG,JCG,IPAR,RPAR,F,GMAX,CMAX,IPRNT,ITERM)
  CALL PNULU(NF,NC,X,IH,JH,CF,ICG,JCG,IPAR,RPAR,F,GMAX,CMAX,IPRNT,ITERM)
```

Their arguments have the following meanings:

Argument	Type	Significance
`NF`	`II`	Number of variables of the objective.
`NC`	`II`	Number of constraints.
`X(NF)`	`RU`	On input, vector with the initial estimate to the solution. On output, the approximation to the minimum.
`IH(NF+1)`	`IA`	Pointers of the diagonal elements in the upper part of the Hessian matrix.
`JH(NH)`	`IA`	Column indices of the nonzero elements and additional working space for the Choleski decomposition of the Hessian matrix.
`CF(NC+1)`	`RA`	Vector containing values of constraint functions (significant only if `NC` > 0).
`ICG(NC+1)`	`IA`	Pointers of the first elements in rows of the constraint Jacobian matrix.
`JCG(MC)`	`IA`	Column indices of nonzero elements of the constraint Jacobian matrix.
`IPAR(7)`	`IA`	Integer parameters (see Table 1).
`RPAR(5)`	`RA`	Real parameters (see Table 1).
`F`	`RO`	Value of the objective function at the solution `X`.
`GMAX`	`RO`	Maximum absolute value of a particular derivative of the Lagrangian function.
`CMAX`	`RO`	Maximum constraint violation.
`IPRNT`	`II`	Print specification: `IPRNT` $= 0$: print is suppressed, `IPRNT` $= 1$: basic print of final results, `IPRNT` $= -1$: extended print of final results, `IPRNT` $= 2$: basic print of intermediate and final results, `IPRNT` $= -2$: extended print of intermediate and final results.
`ITERM`	`IU`	Variable that indicates the cause of termination: `ITERM` $= 1$: if $\|\mathtt{X}-\mathtt{X}_{\mathtt{old}}\|$ was less than or equal to `TOLX` in two subsequent iterations, `ITERM` $= 3$: if `F` is less than or equal to `TOLB`, `ITERM` $= 4$: if `GMAX` is less than or equal to `TOLG`, `ITERM` $= 11$: if `NIT` exceeded `MIT`, `ITERM` $= 12$: if `NFV` exceeded `MFV`, `ITERM` $= 13$: if `NFG` exceeded `MFG`,

`ITERM` < 0:	if the method failed.

Parameter	PIND	PNUL	Parameter	PIND	PNUL
IPAR(1)	MIT	MIT	RPAR(1)	XMAX	XMAX
IPAR(2)	MFV	MFV	RPAR(2)	TOLX	TOLX
IPAR(3)	MFG	MFG	RPAR(3)	TOLC	TOLC
IPAR(4)	MOS5	MOS5	RPAR(4)	TOLG	TOLG
IPAR(5)	MD	MD	RPAR(5)	RPF1	RPF1
IPAR(6)	MDE	MDE			
IPAR(7)	IFIL	IFIL			

Table 1: Integer and real parameters

The integer and real parameters listed in Table 1 have the following meanings:

Argument	Type	Significance
`MIT`	II	Maximum number of iterations; the choice `MIT` $=$ `0` causes that the default value `MIT` $= 1000$ will be taken.
`MFV`	II	Maximum number of function evaluations; the choice `MFV` $= 0$ causes that the default value `MFV` $= 1000$ will be taken.
`MFG`	II	Maximum number of gradient evaluations; the choice `MFG` $= 0$ causes that the default value `MFG` $= 10000$ will be taken.
`MOS5`	II	Choice of preconditioning strategy: `MOS5` < 0: preconditioning by the constraint preconditioner with incomplete Gill-Murray decomposition, `MOS5` $= 1$: preconditioning by the constraint preconditoner with complete Gill-Murray decomposition, `MOS5` $= 2$: preconditioning is not used. The choice `MOS5` $= 0$ causes that the default value `MOS5` $= 1$ will be taken.
`MD`	II	Maximum number of the dense rows; the choice `MD` $= 0$ causes that the default value `MD` $= 10$ will be taken.
`MDE`	II	Maximum number of nonzero elements in sparse rows; the choice `MDE` $= 0$ causes that the default value `MDE` $= 50$ will be taken.
`XMAX`	RI	Maximum stepsize; the choice `XMAX` $= 0$ causes that the default value `XMAX` $= 10^3$ will be taken.
`TOLX`	RI	Tolerance for the change of the coordinate vector `X`; the choice `TOLX` $= 0$ causes that the default value `TOLX` $= 10^{-12}$ will be taken.
`TOLC`	RI	Tolerance for the constraint violation; the choice `TOLC` $= 0$ causes that the default value `TOLC` $= 10^{-6}$ will be taken.
`TOLG`	RI	Tolerance for the Lagrangian function gradient; the choice `TOLG` $= 0$ causes that the default value `TOLG` $= 10^{-6}$ will be taken.
`RPF1`	RI	Value of the penalty parameter in the merit function; the choice `RPF` $= 0$ causes that the default value `RPF` $= 10^{-4}$ will be taken.

The subroutines `PINDU,PNULU` require the user supplied subroutines `OBJ,DOBJ` that define the objective function and its gradient and subroutines `CON,DCON` that define constraint functions and their gradients. These subroutines have the form

```
      SUBROUTINE  OBJ(NF,X,F)
      SUBROUTINE DOBJ(NF,X,G)
      SUBROUTINE  CON(NF,KC,X,FC)
      SUBROUTINE DCON(NF,KC,X,GC)
```

The arguments of the user supplied subroutines have the following meanings:

Argument	Type	Significance
`NF`	`II`	Number of variables of the objective function.
`KC`	`II`	Index of the constraint function.
`X(NF)`	`RI`	An estimate to the solution.
`F`	`RO`	Value of the objective function at the point `X`.
`FC`	`RO`	Value of the `KC`-th constraint function at the point `X`.
`G(NF)`	`RO`	Gradient of the objective function at the point `X`.
`GC(NF)`	`RO`	Gradient of the `KC`-th constraint function at the point `X`.

4 Verification of subroutines

In this section we report the results obtained by using test programs `TINDU`, `TNULU` which serve for demonstration, verification and testing of subroutines `PINDU`, `PNULU`. These results are listed in the following tables (rows corresponding to individual test problems contain the number of iterations `NIT`, the number of function evaluations `NFV`, the number of gradient evaluations `NFG`, the value of the termination criterion constraint `C`, the value of the termination criterion `G` and the cause of termination `ITERM`). All computations reported were performed on a Pentium PC computer, under the Windows XP system using the Digital Visual Fortran (Version 6) compiler, in double-precision arithmetic.

Problem	NIT	NFV	NFG	C	G	ITERM
1	9	10	60	0.177636E-14	0.202238E-11	4
2	12	13	182	0.425254E-10	0.621097E-07	4
3	10	11	66	0.172595E-11	0.842788E-08	4
4	16	19	102	0.141707E-10	0.405805E-09	4
5	18	19	190	0.642577E-08	0.311534E-06	4
6	18	19	266	0.125855E-11	0.711327E-08	4
7	9	11	70	0.118305E-11	0.221512E-11	4
8	38	51	273	0.444089E-15	0.111759E-07	4
9	28	70	203	0.134587E-11	0.125862E-08	4
10	12	15	78	0.140633E-07	0.486837E-07	4
11	9	10	60	0.261324E-11	0.299467E-11	4
12	7	8	56	0.143574E-11	0.288525E-10	4
13	15	17	128	0.177636E-14	0.641856E-06	4
14	12	13	91	0.229161E-07	0.274480E-07	4
15	19	35	120	0.550671E-13	0.127818E-11	4
16	8	9	45	0.679852E-09	0.608118E-07	4
17	9	10	50	0.222045E-15	0.177636E-14	4
18	10	13	55	0.368464E-08	0.809397E-06	4
Σ	259	353	2095	TIME = 00.84		

Table 2: Results obtained by program TINDU

Problem	NIT	NFV	NFG	C	G	ITERM
1	6	7	42	0.536016E-12	0.763704E-08	4
2	11	12	168	0.351663E-13	0.478977E-10	4
3	11	12	72	0.315126E-11	0.434427E-09	4
4	20	21	126	0.285079E-10	0.555964E-08	4
5	15	16	160	0.626330E-08	0.199604E-06	4
6	16	19	238	0.524025E-13	0.320156E-09	4
7	11	13	84	0.652065E-10	0.358497E-09	4
8	40	54	287	0.236033E-12	0.569894E-06	4
9	20	37	147	0.323282E-06	0.393792E-06	4
10	13	20	84	0.120160E-06	0.580382E-06	4
11	7	8	48	0.838722E-09	0.263766E-09	4
12	6	7	49	0.303063E-07	0.569950E-06	4
13	15	25	128	0.278610E-07	0.977278E-06	4
14	13	14	98	0.529941E-10	0.346411E-10	4
15	19	22	120	0.124682E-07	0.429103E-06	4
16	7	8	40	0.110654E-10	0.142114E-10	4
17	8	9	45	0.121537E-12	0.166850E-11	4
18	11	17	60	0.753158E-07	0.768419E-06	4
Σ	249	321	1996	TIME = 00.79		

Table 3: Results obtained by program TNULU

References

[1] Gould N.I.M, Hribar M.E., Nocedal J.: On the solution of equality constrained quadratic programming problems arising in optimization. Technical Report RAL-TR-1998-069, Rutherford Appleton Laboratory, 1998.

[2] Lukšan L., Vlček J.: Indefinitely Preconditioned Inexact Newton Method for Large Sparse Equality Constrained Nonlinear Programming Problems. Numerical Linear Algebra with Applications 5 (1998) 219-247.

[3] Lukšan L., Vlček J.: Numerical experience with iterative methods for equality constrained nonlinear programming problems. Optimization Methods and Software 16 (2001) 257-287.

[4] Tůma M.: A note on direct methods for approximations of sparse Hessian matrices. Aplikace Matematiky 33 (1988) 171-176.