LSA: Algorithms for Large-Scale Optimization

Lukšan, Ladislav
2004

Institute of Computer Science
Academy of Sciences of the Czech Republic

LSA: Algorithms for large-scale optimization

Ladislav Lukšan, Ctirad Matonoha, Jan Vlček

Technical report No. 896

October 30, 2004

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

LSA: Algorithms for large-scale optimization

Ladislav Lukšan, Ctirad Matonoha, Jan Vlček [1]

Technical report No. 896

October 30, 2004

Abstract:

Abstract. We present eleven basic FORTRAN subroutines for large-scale optimization with simple bounds and large-scale systems of nonlinear equations. Subroutines `PLIS` and `PLIP`, intended for dense general optimization problems, are based on limited-memory variable metric methods. Subroutines `PNED` and `PNEC`, intended for sparse general optimization problems, are based on modifications of the discrete Newton method. Subroutine `PSEP`, intended for partially separable optimization problems is based on partitioned variable metric updates. Subroutines `PGAM` and `PGAN`, intended for sparse nonlinear least squares problems, are based on modifications and corrections of the Gauss-Newton method. Subroutines `PEQN` and `PEQL`, intended for sparse systems of nonlinear equations, are based on the discrete Newton method and the inverse column-update quasi-Newton method. 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 least squares, large-scale systems of nonlinear equations, large-scale nonlinear programming, sparse problems, partially separable problems, limited-memory methods, discrete Newton methods, quasi-Newton methods, KKT systems, indefinite preconditioners.

[1]This work was supported by the Grant Agency of the Czech Academy of Sciences, project code IAA1030405, and by the Ministry of Education of the Czech Republic, project code MSM 242200002.
Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Praha 8 and Technical University of Liberec, Hálkova 6, 461 17 Liberec

1 Introduction

We propose eleven basic subroutines which implement selected large-scale optimization algorithms. The double-precision FORTRAN 77 subroutines `PLIS`, `PLIP`, `PNED`, `PNEC` are designed to find a close approximation to a local minimum of a general twice continuously differentiable function $F : R^n \to R$. Subroutines `PLIS`, `PLIP`, based on limited-memory variable metric updates, are intended for general problems with unknown or dense Hessian matrices. Subroutine `PLIS` uses Strang recurrences [12], [23]. Subroutine `PLIP` uses shifted limited-memory variable metric updates [28], [29]. Subroutines `PNED`, `PNEC`, based on the inexact discrete Newton method [6], [4], are intended for problems with sparse Hessian matrices. Subroutine `PNED` uses Moré-Sorensen direct-elimination trust-region strategy [22]. Subroutine `PNEC` uses Steihaug-Toint [25], [26] or shifted Steihaug-Toint [15] iterative trust-region strategy.

The double-precision FORTRAN 77 subroutine `PSEP` is designed to find a close approximation to a local minimum of a special partially separable objective function

$$F(x) = \sum_{i=1}^{n_a} f_i(x).$$

Here $x \in R^n$ is a vector of n variables and $f_i : R^n \to R$, $1 \le i \le n_a$, are twice continuously differentiable functions. Subroutine `PSEP` is based on partitioned variable metric updates [11], [16].

The double-precision FORTRAN 77 subroutines `PGAM`, `PGAN` are designed to find a close approximation to a local minimum of a special least-square function

$$F(x) = \frac{1}{2} \sum_{i=1}^{n_a} f_i^2(x).$$

Here $x \in R^n$ is a vector of n variables and $f_i : R^n \to R$, $1 \le i \le n_a$, are twice continuously differentiable functions. Subroutines `PGAM`, `PGAN` are based on hybrid methods that combine the Gauss-Newton method with the Newton or the variable metric corrections [14], [16].

The double-precision FORTRAN 77 subroutines `PEQN`, `PEQL` are designed to find a solution to a system of nonlinear equations

$$f_i(x) = 0, \qquad 1 \le i \le n.$$

Here $x \in R^n$ is a vector of n variables and $f_i : R^n \to R$, $1 \le i \le n$, are continuously differentiable functions. Subroutine `PEQN` is based on the inexact discrete Newton method [2], [5], [18]. Subroutine `PEQL` is based on the inverse column-update quasi-Newton method [21], [18].

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 [19], [20]. Subroutine PNUL uses null-space transformations and the standard conjugate gradient method applied to reduced system [9], [20].

Subroutines PLIS, PLIP, PNED, PNEC, PSEP, PGAM, PGAN allow us to work with simple bounds. Simple bounds are assumed in the form

$$\begin{aligned} x_i - \text{unbounded} \quad &, \quad I_i^x = 0, \\ x_i^l \le x_i \quad &, \quad I_i^x = 1, \\ x_i \le x_i^u \quad &, \quad I_i^x = 2, \\ x_i^l \le x_i \le x_i^u \quad &, \quad I_i^x = 3, \\ x_i = x_i^l = x_i^u \quad &, \quad I_i^x = 5, \end{aligned}$$

where $1 \le i \le n$ (I^x corresponds to array IX in Section 8).

To simplify the user's work, additional easy-to-use subroutines are added. These subroutines call general subroutines PLIS, PLIP, PNED, PNEC, PSEP, PGAM, PGAN, PEQN, PEQL, PIND, PNUL:

PLISU,PLIPU,PNEDU,PNECU,PSEPU,PGAMU,PGANU – unconstrained optimization.
PLISS,PLIPS,PNEDS,PNECS,PSEPS,PGAMS,PGANS – optimization with simple bounds.
PEQNU,PEQLU – solution of systems of nonlinear equations.
PINDU,PNULU – optimization with general equality constrains.

Each subroutine contains a description of formal parameters and extensive comments. Moreover, text files PLIS.TXT, PLIP.TXT, PNED.TXT, PNEC.TXT, PSEP.TXT, PGAM.TXT, PGAN.TXT, PEQN.TXT, PEQL.TXT, PIND.TXT, PNUL.TXT are added, which contain a detailed description of all important subroutines (including indications of required storage). Finally, test programs TLISU, TLISS, TLIPU, TLIPS, TNEDU, TNEDS, TNECU, TNECS, TSEPU, TSEPS, TGAMU, TGAMS, TGANU, TGANS, TEQNU, TEQLU, TINDU, TNULU are included, which contain sets of test problems. These test programs serve as examples for using the subroutines, verify their correctness and demonstrate their efficiency.

For a better orientation, Table 1 contains the numbers of sections, where direction determination, step-size selection and sparsity pattern are described for a given subroutine. The active set strategy for constraint handling, which is the same for all subroutines that use box constraints, is described in Section 2.

2 Limited-memory variable metric methods for general problems

Consider a general continuously differentiable function $F : R^n \to R$, where n is large, and assume that the Hessian matrix of F is not known. In this case, limited-memory variable metric methods can be efficiently used for seeking a local minimum of F. These methods are realized in the line-search framework so that they generate a sequence of points $x_k \in \mathcal{R}^n$, $k \in \mathcal{N}$, by the simple process

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1}$$

where $d_k = -H_k g_k$ is a direction vector, H_k is a positive definite approximation of the inverse Hessian matrix and $0 < \alpha_k \le \overline{\alpha}_k$ is a scalar step-size chosen in such a way that

$$F_{k+1} - F_k \le \varepsilon_1 \, \alpha_k \, d_k^T g_k, \quad d_k^T g_{k+1} \ge \varepsilon_2 \, d_k^T g_k \tag{2}$$

subroutine	direction	step-size	pattern
PLIS	2.1	2	-
PLIP	2.2	2	-
PNED	3.1	3	3
PNEC	3.3	3	3
PSEP	4	2	4
PGAN	3.1	3	4
PGAM	3.2	3	4
PEQN	6.1	6	4
PEQL	6.2	6	4
PIND	7.1	7	3
PNUL	7.2	7	3

Table 1: Description of subroutines

(the weak Wolfe conditions), where $F_k = F(x_k)$, $g_k = \nabla F(x_k)$, $0 < \varepsilon_1 < 1/2$ is a tolerance for the function value decrease (parameter TOLS in our subroutines) and $\varepsilon_1 < \varepsilon_2 < 1$ is a tolerance for the directional derivative increase (parameter TOLP in our subroutines). The maximum step-size $\overline{\alpha}_k$ is given by formula $\overline{\alpha}_k = \overline{\Delta}/\|d_k\|$, where $\overline{\Delta}$ is an upper bound for difference $x_{k+1} - x_k$ (parameter XMAX in our subroutines). Step-size α_k is chosen iteratively 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). We start with the initial estimate $\alpha_k = 1$ if INITS = 0 or the initial estimate is derived by using the lower bound for F (parameter TOLB in our subroutines) if INITS = 1. Matrices H_k, $k \in N$, are computed recursively either by using a limited (small) number of variable metric updates applied to the scaled unit matrix or by updating low dimension matrices. In the first iteration or after restart, we set $H_k = I$ (the unit matrix) and $\underline{k} = k$. Restart is performed if $-d_k^T g_k \leq \underline{\varepsilon}\|d_k\|\|g_k\|$, where $\underline{\varepsilon}$ is a restart tolerance (parameter TOLD in our subroutines).

If box constraints are considered, then a simple active set strategy is used. To simplify the notation, we omit iteration index k in the following description. Every iteration is started by detection of active constraints. Thus we set

$$\begin{aligned}
x_i = x_i^l, \quad I_i^x = -1 \quad &\text{if} \quad I_i^x = 1, \quad x_i \leq x_i^l + \varepsilon_c \max(x_i^l, 1), \\
x_i = x_i^u, \quad I_i^x = -2 \quad &\text{if} \quad I_i^x = 2, \quad x_i \geq x_i^u - \varepsilon_c \max(x_i^u, 1), \\
x_i = x_i^l, \quad I_i^x = -3 \quad &\text{if} \quad I_i^x = 3, \quad x_i \leq x_i^l + \varepsilon_c \max(x_i^l, 1), \\
x_i = x_i^u, \quad I_i^x = -4 \quad &\text{if} \quad I_i^x = 3, \quad x_i \geq x_i^u - \varepsilon_c \max(x_i^u, 1), \\
x_i = x_i^l = x_i^u, \quad I_i^x = -5 \quad &\text{if} \quad I_i^x = 5
\end{aligned}$$

for $1 \leq i \leq n$, where ε_c is a required precision (we use value $\varepsilon_c = 10^{-8}$ in our subroutines). After computing gradient $g = g(x)$, we determine projected gradient g^p and chopped gradient g^c in such a way that

$$
\begin{array}{rrll}
g_i^p = 0, & g_i^c = \max(0, g_i) & \text{for} & I_i^x = -1 \quad \text{or} \quad I_i^x = -3, \\
g_i^p = 0, & g_i^c = \min(0, g_i) & \text{for} & I_i^x = -2 \quad \text{or} \quad I_i^x = -4, \\
g_i^p = 0, & g_i^c = 0 & \text{for} & I_i^x = -5, \\
g_i^p = g_i, & g_i^c = g_i & \text{for} & I_i^x = 0, \\
g_i^p = g_i, & g_i^c = 0 & \text{for} & I_i^x > 0.
\end{array}
$$

If $\max_{1 \le i \le n} |g_i^c| > \max_{1 \le i \le n} |g_i^p|$ and $\overline{\alpha} > 0$, we delete redundant active constraints in such a way that

$$
\begin{array}{llll}
I_i^x = 1 & \text{if} & I_i^x = -1 \quad \text{and} & g_i > 0, \\
I_i^x = 2 & \text{if} & I_i^x = -2 \quad \text{and} & g_i < 0, \\
I_i^x = 3 & \text{if} & I_i^x = -3 \quad \text{and} & g_i > 0, \\
I_i^x = 3 & \text{if} & I_i^x = -4 \quad \text{and} & g_i < 0.
\end{array}
$$

Note that the iterative process is always restarted after a constraint deletion to assure the validity of condition $d^T g < 0$. Direction vector d has to be determined in such a way that $d_i = 0$ if $I_i^x < 0$. If d is computed by a matrix multiplication or by solving a system of linear equations, the matrix used can be changed by deleting rows and columns corresponding to active constraints. This is realized by changing signs of elements of arrays `JH` or `JAG` in our subroutines. Before step-size selection, we have to determine the maximum step-size $\overline{\alpha}$ to assure the feasibility. This is computed by the formula $\overline{\alpha} = \min(\overline{\alpha}_1, \overline{\alpha}_2, \overline{\alpha}_3, \overline{\alpha}_4, \overline{\Delta}/\|d\|)$, where

$$
\overline{\alpha}_1 = \min_{I_i^x = 1, d_i < 0} \frac{x_i^l - x_i}{d_i}, \qquad \overline{\alpha}_2 = \min_{I_i^x = 2, d_i > 0} \frac{x_i^u - x_i}{d_i},
$$

$$
\overline{\alpha}_3 = \min_{I_i^x = 3, d_i < 0} \frac{x_i^l - x_i}{d_i}, \qquad \overline{\alpha}_4 = \min_{I_i^x = 3, d_i > 0} \frac{x_i^u - x_i}{d_i}
$$

(if a corresponding set is empty we use the value ∞). This simple active set strategy is also used in a trust region framework. Thus the above process is also utilized in subroutines described in Sections 3-5 if box constraints are considered.

2.1 Limited-memory BFGS method

Subroutine `PLIS` is an implementation of the limited-memory BFGS method proposed in [12], [23]. This method works with matrices $H_k = H_k^k$, where $H_{k-m}^k = \gamma_k I$ (we use $\gamma_k = b_{k-1}/a_{k-1}$ in our implementation) and

$$
H_{j+1}^k = V_j^T H_j^k V_j + \frac{1}{b_j} s_j s_j^T, \qquad V_j = I - \frac{1}{b_j} y_j s_j^T \tag{3}
$$

for $k - m \le j \le k - 1$. Here $s_j = x_{j+1} - x_j$, $y_j = g_{j+1} - g_j$, $a_j = y_j^T H_j y_j$, $b_j = y_j^T s_j$. Thus

$$
H_{j+1}^k = \frac{b_{k-1}}{a_{k-1}} \left(\prod_{i=k-m}^{j} V_i \right)^T \left(\prod_{i=k-m}^{j} V_i \right) + \sum_{l=k-m}^{j} \frac{1}{b_l} \left(\prod_{i=l+1}^{j} V_i \right)^T s_l s_l^T \left(\prod_{i=l+1}^{j} V_i \right) g_k. \tag{4}
$$

Matrix $H_k = H_k^k$ need not be constructed explicitly since we need only vector $d_k = -H_k^k g_k$, which can be computed by using two recurrences (the Strang formula). First, vectors

$$u_j = -\left(\prod_{i=j}^{k-1} V_i\right) g_k,$$

$k-1 \geq j \geq k-m$, are computed by using the backward recurrence

$$\begin{aligned} \sigma_j &= s_j^T u_{j+1}/b_j, \\ u_j &= u_{j+1} - \sigma_j y_j, \end{aligned}$$

where $u_k = -g_k$. Then vectors

$$v_{j+1} = \frac{b_{k-1}}{a_{k-1}}\left(\prod_{i=k-m}^{j} V_i\right)^T u_{k-m} + \sum_{l=k-m}^{j} \frac{1}{b_l}\left(\prod_{i=l+1}^{j} V_i\right)^T s_l s_l^T u_{l+1},$$

$k-m \leq j \leq k-1$, are computed by using the forward recurrence

$$v_{j+1} = v_j + (\sigma_j - y_j^T v_j/b_j) s_j,$$

where $v_{k-m} = (b_{k-1}/a_{k-1}) u_{k-m}$. Finally we set $d_k = v_k$. Note that $2m$ vectors s_j, y_j, $k-m \leq j \leq k-1$ are used and stored. The number of consecutive variable metric updates is defined as $m = \min(\overline{m}, k-\underline{k})$ where $\overline{m} =$ `MF` (`MF` is a parameter of the subroutine `PLIS`) and $\underline{k}$ is an index of the iteration corresponding to the last restart.

2.2 Shifted limited-memory variable metric methods

Subroutine `PLIP` is an implementation of shifted limited-memory variable metric methods proposed in [28], [29]. These methods work with matrices $H_k = \zeta_k I + U_k U_k^T$, where $n \times m$ matrix U_k is updated by formula $U_{k+1} = V_k U_k$ with a low rank matrix V_k chosen in such a way that the (modified) quasi-Newton condition $U_{k+1} U_{k+1}^T y_k = \rho_k \tilde{s}_k$ with $\tilde{s}_k = s_k - \zeta_{k+1} y_k$ is satisfied (we use the same notation, namely s_k, y_k, a_k, b_k, as in Section 2.1). This condition can be replaced by equations

$$U_{k+1}^T y_k = z_k, \qquad U_{k+1} z_k = \rho_k \tilde{s}_k, \qquad \|z_k\|^2 = \rho_k y_k^T \tilde{s}_k. \tag{5}$$

where z_k is an optional vector parameter. Note that the last equality, which is a consequence of the first two equalities, is the only restriction laid on the vector z. To simplify the notation, we define vectors $u_k = U_k^T y_k$ and $v_k = U_k^T H_k^{-1} s_k = -\alpha_k U_k^T g_k$.

The choice of shift parameter ζ_{k+1} is a crucial part of shifted limited-memory variable metric methods. The value

$$\zeta_{k+1} = \mu_k \frac{b_k}{\|y_k\|^2}, \qquad \mu_k = \frac{\sqrt{1 - \|u_k\|^2/a_k}}{1 + \sqrt{1 - b_k^2/(\|s_k\|^2 \|y_k\|^2)}}. \tag{6}$$

is used in subroutine `PLIP`. The most efficient shifted limited-memory variable metric methods can be derived by a variational principle. Let T_k be a symmetric positive definite

matrix. It can be shown (see [29]) that the Frobenius norm $\|T_k^{-1/2}(U_{k+1}-U_k)\|_F^2$ is minimal on the set of all matrices satisfying quasi-Newton condition (5) if and only if

$$U_{k+1} = U_k - \frac{T_k y_k}{y_k^T T_k y_k} y_k^T U_k + \left(\rho_k \tilde{s}_k - U_k z_k + \frac{y_k^T U_k z_k}{y_k^T T_k y_k} T_k y_k\right) \frac{z_k^T}{\|z_k\|^2}. \tag{7}$$

Here $T_k y_k$ and z_k are vector parameters defining a class of shifted limited-memory variable metric methods. Using suitable values of these vectors, we obtain particular methods of this class.

Assuming that $T_k y_k$ and $\rho_k \tilde{s}_k - U_k z_k$ are linearly dependent and setting

$$z_k = \vartheta_k v_k, \qquad \vartheta_k = \pm\sqrt{\rho_k y_k^T \tilde{s}_k / \|v_k\|^2} \tag{8}$$

we obtain rank 1 variationally derived method (VAR1), where

$$U_{k+1} = U_k - \frac{\rho_k \tilde{s}_k - \vartheta_k U_k v_k}{\rho_k y_k^T \tilde{s}_k - \vartheta_k u_k^T v_k} \left(u_k - \vartheta_k v_k\right)^T, \tag{9}$$

which gives the best results for the choice $\mathrm{sgn}(\vartheta_k u_k^T v_k) = -1$. Method VAR1 is chosen if `MET` $= 1$ in the subroutine `PLIP`. Using z_k given by (8) and setting $T_k y_k = \tilde{s}_k$, which corresponds to the BFGS method in the full-memory case, we obtain rank 2 variationally derived method (VAR2), where

$$U_{k+1} = U_k - \frac{\tilde{s}_k}{y_k^T \tilde{s}_k} u_k^T + \left(\rho_k \frac{\tilde{s}_k}{\vartheta_k} - U_k v_k + \frac{u_k^T v_k}{y_k^T \tilde{s}_k} \tilde{s}_k\right) \frac{v_k^T}{\|v_k\|^2}. \tag{10}$$

Method VAR2 is chosen if `MET` $= 2$ in the subroutine `PLIP`. The efficiency of both these methods depends on the value of correction parameter ρ_k. Value of this parameter is determined by variable `MEC`. If `MEC` $= 1$, then $\rho_k = 1$. If `MEC` $= 2$, then $\rho_k = \nu_k$. If `MEC` $= 3$, then $\rho_k = \varepsilon_k$. If `MEC` $= 4$, then $\rho_k = \sqrt{\nu_k \varepsilon_k}$. If `MEC` $= 5$, then $\rho_k = \zeta_k/(\zeta_k + \zeta_{k+1})$. Here $\nu_k = \mu_k/(1-\mu_k)$, μ_k is a relative shift parameter defined by (6) and

$$\varepsilon_k = \sqrt{1 - \|u_k\|^2/a_k}$$

is the damping factor of μ_k (nominator in (6)). The number of columns of matrix U_k is defined as $m = \min(\overline{m}, k - \underline{k})$ where $\overline{m} =$ `MF` (`MF` is a parameter of the subroutine `PLIP`) and $\underline{k}$ is an index of the iteration corresponding to the last restart.

3 Inexact discrete Newton methods for sparse problems

Consider a general twice continuously differentiable function $F : R^n \to R$, where n is large, and assume that the Hessian matrix $G(x) = [G_{ij}(x)] = [\partial^2 F(x)/(\partial x_i \partial x_i)]$ is sparse. In this case, discrete versions of the Newton method can be efficiently used for seeking a local minimum of F. These methods are based on the fact that sufficiently sparse Hessian matrices can be estimated by using a small number of gradient differences [4]. We use the algorithm proposed in [27] in subroutines `PNED`, `PNEC`. 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 following 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

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

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 NF+1 elements and the last element is equal to `MH`+1, where `MH` is the number of nonzero elements stored. This convention is also used in subroutines `PIND`, `PNUL` described in Section 7.

Since the Hessian matrix can be indefinite, discrete versions of the Newton method are realized in the trust-region framework. Let B_k be a gradient-difference approximation of the Hessian matrix $G_k = G(x_k)$. Denote

$$Q_k(d) = \frac{1}{2} d^T B_k d + g_k^T d$$

the quadratic function which locally approximates difference $F(x_k + d) - F(x_k)$,

$$\omega_k(d) = (B_k d + g_k)/\|g_k\|$$

the accuracy of direction determination and

$$\rho_k(d) = \frac{F(x_k + d) - F(x_k)}{Q_k(d)}$$

the ratio of actual and predicted decrease of the objective function. Trust-region methods (used in subroutines `PNED`, `PNEC`, `PGAM`, `PGAN`) generate points $x_k \in \mathcal{R}^n$, $k \in N$, in such a way that x_1 is arbitrary and

$$x_{k+1} = x_k + \alpha_k d_k, \quad k \in N, \tag{11}$$

where $d_k \in \mathcal{R}^n$ are direction vectors and $\alpha_k \geq 0$ are step-sizes. Direction vectors $d_k \in \mathcal{R}^n$ are chosen to satisfy conditions

$$\|d_k\| \leq \Delta_k, \tag{12}$$

$$\|d_k\| < \Delta_k \quad \Rightarrow \quad \|\omega_k(d_k)\| \leq \overline{\omega}_k, \tag{13}$$

$$-Q_k(d_k) \geq \underline{\sigma} \|g_k\| \min(\|d_k\|, \|g_k\|/\|B_k\|), \tag{14}$$

where $0 \leq \overline{\omega}_k \leq \overline{\omega} < 1$ and $0 < \underline{\sigma} < 1$ (we use value $\overline{\omega} = 0.9$ in our subroutines; $\underline{\sigma}$ is a theoretical value given implicitly). Step-sizes $\alpha_k \geq 0$ are selected so that

$$\rho_k(d_k) \leq 0 \quad \Rightarrow \quad \alpha_k = 0, \tag{15}$$

$$\rho_k(d_k) > 0 \quad \Rightarrow \quad \alpha_k = 1. \tag{16}$$

Trust-region radii $0 < \Delta_k \le \overline{\Delta}$ are chosen in such a way that $0 < \Delta_1 \le \overline{\Delta}$ (Δ_1 and $\overline{\Delta}$ are given by parameters `XDEL` and `XMAX` of our subroutines) and

$$\rho_k(d_k) < \underline{\rho} \quad \Rightarrow \quad \underline{\beta}\|d_k\| \le \Delta_{k+1} \le \overline{\beta}\|d_k\|, \tag{17}$$

$$\underline{\rho} \le \rho_k(d_k) \le \overline{\rho} \quad \Rightarrow \quad \Delta_{k+1} = \Delta_k, \tag{18}$$

$$\overline{\rho} < \rho_k(d_k) \quad \Rightarrow \quad \Delta_{k+1} = \min(\gamma\Delta_{k+1}, \overline{\Delta}), \tag{19}$$

where $0 < \underline{\beta} \le \overline{\beta} < 1 < \gamma$ and $0 < \underline{\rho} < \overline{\rho} < 1$ (we use values $\underline{\beta} = 0.05$, $\overline{\beta} = 0.75$, $\gamma = 2$, $\underline{\rho} = 0.1$, $\overline{\rho} = 0.9$ in our subroutines). Note that the initial trust-region radius Δ_1 is computed by a special formula when `XDEL` $= 0$. This formula contains lower bound for F (parameter `TOLB` in our subroutines) if `INITS` $= 1$. If `INITS` $= 0$, parameter `TOLB` need not be defined.

To simplify the description of individual methods for computing a trust-region step, we omit the outer index k and denote the inner index by i.

3.1 Direct Moré-Sorensen trust region method

Subroutine `PNED` is based on the computation of the optimum locally constrained step. In this case, vector $d \in R^n$ is obtained by solving subproblem

$$\text{minimize} \quad Q(d) = \frac{1}{2}d^T B d + g^T d \quad \text{subject to} \quad \|d\| \le \Delta. \tag{20}$$

Necessary and sufficient conditions for this solution are

$$\|d\| \le \Delta, \quad (B + \lambda I)d + g = 0, \quad B + \lambda I \succeq 0, \quad \lambda \ge 0, \quad \lambda(\Delta - \|d\|) = 0 \tag{21}$$

(we use symbol $\succeq$ for ordering by positive semidefiniteness). The Moré-Sorensen method [22] is based on solving nonlinear equation $1/\|d(\lambda)\| = 1/\Delta$ with $(B + \lambda I)d(\lambda) + g = 0$ by the Newton method using the sparse Choleski decomposition of $B + \lambda I$. More precisely, we determine $\underline{\mu}_1$ as the maximal diagonal element of matrix $-B$, set

$$\underline{\lambda}_1 = \max(\underline{\mu}_1, 0), \qquad \overline{\lambda}_1 = \|g\|/\Delta + \|B\|, \qquad \lambda_1 = \underline{\lambda}_1$$

and for $i = 1, 2, 3, \ldots$ we proceed in the following way. Carry out the Gill-Murray [8] decomposition $B + \lambda_i I + E_i = R_i^T R_i$. If $E_i \ne 0$, determine vector $v_i \in R^n$ such that $\|v_i\| = 1$ and $v_i^T(B + \lambda_i I)v_i < 0$, set $\underline{\mu}_i = \lambda_i - v_i^T(B + \lambda_i I)v_i$, $\underline{\lambda}_i = \underline{\mu}_i$, $\lambda_i = \underline{\lambda}_i$ and repeat this process (i.e., carry out the new Gill-Murray decomposition $B + \lambda_i I + E_i = R_i^T R_i$). If $E_i = 0$, compute vector $d_i \in R^n$ by solving the equation $R_i^T R_i d_i + g = 0$. If $\|d_i\| > \overline{\delta}\Delta$, set $\underline{\lambda}_{i+1} = \lambda_i$ and $\overline{\lambda}_{i+1} = \overline{\lambda}_i$. If $\underline{\delta}\Delta \le \|d_i\| \le \overline{\delta}\Delta$ or $\|d_i\| < \underline{\delta}\Delta$ and $\lambda_i = 0$, set $d = d_i$ and terminate the computation. If $\|d_i\| < \underline{\delta}\Delta$ and $\lambda_i \ne 0$, set $\underline{\lambda}_{i+1} = \underline{\lambda}_i$, $\overline{\lambda}_{i+1} = \lambda_i$, determine vector $v_i \in R^n$ such that $\|v_i\| = 1$ and $v_i^T d_i \ge 0$, which is a good approximation of the eigenvector of matrix B corresponding to its minimal eigenvalue, and compute number $\alpha_i \ge 0$ such that $\|d_i + \alpha_i v_i\| = \Delta$. If

$$\alpha_i^2\|R_i v_i\|^2 \le (1 - \underline{\delta}^2)(\|R_i d_i\|^2 + \lambda_i \Delta^2),$$

set $d = d_i + \alpha_i v_i$ and terminate the computation, otherwise set $\underline{\mu}_i = \lambda_i - \|R_i v_i\|^2$. In this case or if $\|d_i\| > \overline{\delta}\Delta$, compute vector $v_i \in R^n$ by solving the equation $R_i^T v_i = d_i$ and set

$$\lambda_{i+1} := \lambda_i + \frac{\|d_i\|^2}{\|v_i\|^2}\left(\frac{\|d_i\| - \Delta}{\Delta}\right).$$

If $\lambda_{i+1} < \underline{\lambda}_{i+1}$, set $\lambda_{i+1} = \underline{\lambda}_{i+1}$. If $\lambda_{i+1} > \overline{\lambda}_{i+1}$, set $\lambda_{i+1} = \overline{\lambda}_{i+1}$. We use values $\underline{\delta} = 0.9$ and $\overline{\delta} = 1.1$ in our subroutines.

3.2 Direct dog-leg trust region method

The Moré-Sorensen method is very robust but requires 2-3 Choleski decompositions per iteration on average. Simpler methods are based on minimization of $Q(d)$ on the two-dimensional subspace containing Cauchy step $d_C = -(g^T g / g^T B g) g$ and Newton step $d_N = -B^{-1} g$. Subroutine `PGAM` described in Section 5.2 is based on the dog-leg method proposed in [24]. This method uses vectors $d = d_N$ if $\|d_N\| \leq \Delta$ or $d = (\Delta / \|d_C\|) d_C$ if $\|d_C\| \geq \Delta$. In the remaining case (i.e., if $\|d_C\| < \Delta < \|d_N\|$), d is a convex combination of d_C and d_N such that $\|d\| = \Delta$. This method requires only one Choleski decomposition per iteration.

3.3 Iterative shifted Steihaug-Toint trust region method

If B is not sufficiently sparse, then the sparse Choleski decomposition of B is expensive. In this case, methods based on preconditioned conjugate gradient (CG) iterations are more suitable. Steihaug [25] and Toint [26] proposed a method based on the fact that $Q(d_{i+1}) < Q(d_i)$ and $\|d_{i+1}\|_C > \|d_i\|_C$ (where $\|d_i\|_C^2 = d_i^T C d_i$) hold in the preconditioned CG iterations if CG coefficients are positive. We either obtain an unconstrained solution with a sufficient precision or stop on the trust-region boundary if a negative curvature is indicated or if the trust-region is left. More precisely, we set $d_1 = 0$, $g_1 = g$, $p_1 = C^{-1} g$ and for $i = 1, 2, 3, \ldots$ we proceed in the following way. If $\|g_i\| \leq \overline{\omega} \|g\|$, then set $d = d_i$ and terminate the computation, otherwise set

$$q_i = B p_i, \qquad \alpha_i = g_i^T C^{-1} g_i, / p_i^T q_i.$$

If $\alpha_i \leq 0$, determine $\alpha_i \geq 0$ in such a way that $\|d_i + \alpha_i p_i\| = \Delta$, set $d := d_i + \alpha_i p_i$ and terminate the computation, otherwise compute $d_{i+1} = d_i + \alpha_i p_i$. If $\|d_{i+1}\| \geq \Delta$, determine $\alpha_i \geq 0$ in such a way that $\|d_i + \alpha_i p_i\| = \Delta$, set $d := d_i + \alpha_i p_i$ and terminate the computation, otherwise compute

$$\begin{aligned} g_{i+1} &= g_i + \alpha_i q_i, \qquad \beta_i = g_{i+1}^T C^{-1} g_{i+1} / g_i^T C^{-1} g_i \\ p_{i+1} &= -C^{-1} g_{i+1} + \beta_i p_i \end{aligned}$$

Matrix C serves as a preconditioner. The choice $C = I$ is used if `MOS2` $= 0$ or an incomplete Choleski decomposition of matrix B is used if `MOS2` $\neq 0$ (`MOS2` is a parameter of the subroutine `PNEC`). If `MOS2` > 0, a preliminary solution obtained by the incomplete Choleski decomposition can be accepted. In this case, we first compute $p_1 = -C^{-1} g$. If $\|B p_1 + g\| \leq \overline{\omega} \|g\|$, we set $d = p_1$ and terminate the computation, otherwise we continue by CG iterations as above.

Preconditioned CG method gives the monotone increasing sequence $\|d_i\|_C$, $i \in N$, but we use sequence $\|d_i\|$, $i \in N$, to satisfy constraint $\|d\| \leq \Delta$. Thus the solution on the trust-region boundary obtained by the preconditioned CG method can be farther from the optimal locally constrained step than that obtained without preconditioning. This insufficiency is usually compensated by the rapid convergence of the preconditioned CG method.

Subroutine `PNEC` is based on the Steihaug-Toint method described above if `MOS3` $= 0$ or on a slightly more complicated shifted Steihaug-Toint method proposed in [13], [15] if `MOS3` > 0 (`MOS3` is a parameter of the subroutine `PNEC`). The shifted Steihaug-Toint method consists of the three steps:

1. Let $m =$ `MOS3` (the default value is `MOS3` $= 5$). Determine tridiagonal matrix T of order m by using m steps of the (unpreconditioned) Lanczos method (described, e.g., in [10], [13]) applied to matrix B with the initial vector g.

2. Solve subproblem

$$\text{minimize} \quad \frac{1}{2}\tilde{d}^T T \tilde{d} + \|g\| e_1^T \tilde{d} \quad \text{subject to} \quad \|\tilde{d}\| \leq \Delta \tag{22}$$

   by using the method of Moré and Sorensen described in Section 3.1 to obtain Lagrange multiplier $\tilde{\lambda}$.

3. Apply the (preconditioned) Steihaug-Toint method described above to subproblem

$$\text{minimize} \quad \frac{1}{2} d^T (B + \tilde{\lambda} I) d + g^T d \quad \text{subject to} \quad \|d\| \leq \Delta \tag{23}$$

   to obtain direction vector $d = d(\tilde{\lambda})$.

Let $\tilde{\lambda}$ be the Lagrange multiplier of small-size subproblem (22) and λ be the Lagrange multiplier obtained by the Moré-Sorensen method applied to the original trust-region subproblem (20). It can be shown (see [15]) that $0 \leq \tilde{\lambda} \leq \lambda$. This inequality assures that $\lambda = 0$ implies $\tilde{\lambda} = 0$ so $\|d\| < \Delta$ implies $\tilde{\lambda} = 0$. Thus the shifted Steihaug-Toint method reduces to the standard one in this case. At the same time, if B is positive definite and $\tilde{\lambda} > 0$, then one has $\Delta \leq \|(B + \tilde{\lambda} I)^{-1} g\| < \|B^{-1} g\|$. Thus the unconstrained minimizer of the shifted quadratic function (23) is closer to the trust-region boundary than the unconstrained minimizer of the original quadratic function (20) and we can expect that $d(\tilde{\lambda})$ is closer to the optimal locally constrained step than $d(0)$. Finally, if $\tilde{\lambda} > 0$, then matrix $B + \tilde{\lambda} I$ is better conditioned than B and we can expect that the shifted Steihaug-Toint method will converge more rapidly than the original one.

4 Variable metric methods for partially separable problems

Consider functions of the form

$$F(x) = \sum_{i=1}^{m} f_i(x), \tag{24}$$

where $f_i(x)$, $1 \leq i \leq m$ (m is usually large), are smooth particular functions depending on a small number of variables (n_i, say). In this case, the Jacobian matrix $J(x) = [J_{ij}(x)] = [\partial f_i(x)/\partial x_j]$ is sparse. The sparsity pattern of the Jacobian matrix is stored in the standard compressed row format using arrays `IAG` and `JAG`. For example, if the Jacobian matrix has the following pattern

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

(asterisks denote nonzero elements) then arrays IAG and JAG contain elements

$$\texttt{IAG} = \left[\begin{array}{cccccc} 1 & 4 & 7 & 9 & 11 & 13 \end{array}\right], \qquad \texttt{JAG} = \left[\begin{array}{cccccccccccc} 1 & 2 & 4 & 1 & 2 & 3 & 1 & 4 & 2 & 3 & 1 & 3 \end{array}\right],$$

i.e., IAG contains pointers of the first elements in rows of the Jacobian matrix and JAG contains column indices of the nonzero elements. Note that IAG has NA+1 elements and the last element is equal to MA+1, where MA is the number of nonzero elements. This convention is also used in subroutines PGAM, PGAN, PEQN, PEQL described in Sections 5-6.

Using the sparsity pattern of the Jacobian matrix, we can define packed gradients $\hat{g}_i(x) \in \mathcal{R}^{n_i}$ and packed Hessian matrices $\hat{G}_i(x) \in \mathcal{R}^{n_i \times n_i}$ of functions $f_i(x)$, $1 \leq i \leq m$, as dense but small-size vectors and matrices. Subroutine PSEP is based on partitioned variable metric updates, which consider each particular function separately. Thus approximations $\hat{B}_i$, $1 \leq i \leq m$, of the packed Hessian matrices $\hat{G}_i(x)$ are updated by using the quasi-Newton conditions $\hat{B}_i^+ \hat{s}_i = \hat{y}_i$, where $\hat{s}_i \in \mathcal{R}^{n_i}$ is a part of the vector s consisting of components corresponding to variables of f_i and $\hat{y}_i = \hat{g}_i^+ - \hat{g}_i$ (we omit outer index k and replace index $k+1$ by $+$ in this section). Therefore, a variable metric update can be used for each of the particular functions. However, there is a difference between the classic and the partitioned approach. Denoting $\hat{b}_i = \hat{y}_i^T \hat{s}_i$, $\hat{c}_i = \hat{s}_i^T \hat{B}_i \hat{s}_i$, we can observe that $\hat{b}_i \geq 0$ does not have to be guaranteed for all $1 \leq i \leq m$. This difficulty is unavoidable and an efficient algorithm has to handle this situation. Subroutine PSEP uses two strategies. If MET = 1, then the safeguarded partitioned BFGS method with updates

$$\begin{aligned} \hat{B}_i^+ &= \hat{B}_i + \frac{1}{\hat{b}_i} \hat{y}_i \hat{y}_i^T - \frac{1}{\hat{c}_i} \hat{B}_i \hat{s}_i \left(\hat{B}_i \hat{s}_i\right)^T, \quad \hat{b}_i > 0, \\ \hat{B}_i^+ &= \hat{B}_i, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \hat{b}_i \leq 0. \end{aligned} \tag{25}$$

is used. If MET = 2, then the BFGS updates are combined with the rank-one updates

$$\begin{aligned} \hat{B}_i^+ &= \hat{B}_i + \frac{1}{\hat{b}_i - \hat{c}_i} \left(\hat{y}_i - \hat{B}_i \hat{s}_i\right) \left(\hat{y}_i - \hat{B}_i \hat{s}_i\right)^T, \quad |\hat{b}_i - \hat{c}_i| \neq 0, \\ \hat{B}_i^+ &= \hat{B}_i, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad |\hat{b}_i - \hat{c}_i| = 0. \end{aligned} \tag{26}$$

We use a strategy, which is based on the observation that (25) usually fails in the case when too many particular functions have indefinite Hessian matrices. We start with the partitioned BFGS update (25). If $m_{neg} \geq \theta m$, where m_{neg} is a number of particular functions with a negative curvature and θ is a threshold value, then (26) is used for all particular functions in all subsequent iterations (we use value $\theta = 1/2$ in the subroutine PSEP).

A disadvantage of partitioned variable metric methods is the fact that approximations of packed Hessian matrices need to be stored. Therefore, the number of stored elements can be much greater than the number of nonzero elements in the Hessian pattern. Moreover, a partitioned structure cannot be used for sparse elimination directly. Thus the standard sparse Hessian representation is constructed in subroutine PSEP before solving linear systems. Variable metric methods for partially separable problems are implemented in the line-search framework described in Section 2.

5 Hybrid methods for nonlinear least-squares

Consider functions of the form

$$F(x) = \frac{1}{2}\sum_{i=1}^{m} f_i^2(x) = \frac{1}{2} f^T(x) f(x)$$

(sum of squares), where $f_i(x)$, $1 \le i \le m$ (m is usually large), are smooth functions depending on a small number of variables (n_i, say). In this case, the Jacobian matrix $J(x) = [J_{ij}(x)] = [\partial f_i(x)/\partial x_j]$ is sparse. The sparsity pattern of the Jacobian matrix is stored in the standard compressed row format using arrays `IAG` and `JAG` in the way described in Section 4.

Using the Jacobian matrix, we can express gradient $g(x)$ and Hessian matrix $G(x)$ in the form

$$\begin{aligned} g(x) &= \sum_{i=1}^{m} f_i(x) g_i(x) = J^T(x) f(x), \\ G(x) &= \sum_{i=1}^{m} \left(g_i(x) g_i^T(x) + f_i(x) G_i(x) \right) = J^T(x) J(x) + C(x) \end{aligned}$$

($G_i(x)$ are Hessian matrices of $f_i(x)$, $1 \le i \le m$). The well-known Gauss-Newton method uses matrix $J^T(x)J(x)$ instead of the Hessian matrix $G(x) = J^T(x)J(x) + C(x)$ (i.e., it omits the second order information contained in $C(x)$). We assume that matrix $J^T(x)J(x)$ is sparse (then also $C(x)$ is sparse). Matrix $J^T(x)J(x)$ is frequently ill-conditioned (even singular) so that the Gauss-Newton method requires a trust-region realization.

If the minimum value $F(x^*)$ is large (large residual problem), the Gauss-Newton method can be inefficient. Therefore, modifications based on the estimation of the second-order term have been developed. These modifications are based on the fact (proved in [1]) that $(F_k - F_{k+1})/F_k \to 1$ if $F_k \to 0$ Q-superlinearly and $(F_k - F_{k+1})/F_k \to 0$ if $F_k \to F^* > 0$. Thus we can use the following philosophy. Direction vector d_k is obtained by the trust-region strategy described in Section 3. If $x_{k+1} \ne x_k$ (i.e., if (16) holds), we compute $F_{k+1} = F(x_{k+1})$, $J_{k+1} = J(x_{k+1})$ and set

$$\begin{aligned} B_{k+1} &= J_{k+1}^T J_{k+1}, \qquad\quad F_k - F_{k+1} > \underline{\vartheta} F_k, \\ B_{k+1} &= J_{k+1}^T J_{k+1} + C_{k+1}, \quad F_k - F_{k+1} \le \underline{\vartheta} F_k, \end{aligned}$$

where C_{k+1} is an approximation of the second order term and $\underline{\vartheta}$ is a suitable value (parameter `ETA` in subroutines `PGAM` and `PGAN`).

For medium-size problems, matrix C_{k+1} is usually obtained by dense variable metric updates [1], which are unsuitable in the large-scale case. Fortunately, simple corrections utilizing sparsity considerably increase the efficiency of the Gauss-Newton method. We have implemented two hybrid methods proposed in [14].

5.1 Gauss-Newton method with the Newton corrections

Subroutine `PGAN` is based on the Newton corrections. In the first iteration (or after a restart) we use matrix $B_k = J_k^T J_k$. In the subsequent iterations we set

$$
\begin{aligned}
B_{k+1} &= J_{k+1}^T J_{k+1}, & F_k - F_{k+1} > \underline{\vartheta} F_k, \\
B_{k+1} &= J_{k+1}^T J_{k+1} + \sum_{i=1}^{m} f_i(x_{k+1}) G_i(x_{k+1}), & F_k - F_{k+1} \leq \underline{\vartheta} F_k,
\end{aligned}
$$

where $G_i(x_{k+1})$, $1 \leq i \leq m$, are approximations of Hessian matrices determined by using gradient differences at the point x_{k+1}. Subroutine `PGAN` uses the Moré-Sorensen trust-region strategy described in Section 3.1.

5.2 Gauss-Newton method with the Marwil corrections

Subroutine `PGAM` is based on the Marwil variable metric corrections. In the first iteration (or a after restart) we use matrix $B_k = J_k^T J_k$. In the subsequent iterations we set

$$
\begin{aligned}
B_{k+1} &= J_{k+1}^T J_{k+1}, & F_k - F_{k+1} > \underline{\vartheta} F_k, \\
B_{k+1} &= \mathcal{P}_S \mathcal{P}_{QG}(J_{k+1}^T J_{k+1}), & F_k - F_{k+1} \leq \underline{\vartheta} F_k,
\end{aligned}
$$

where $\mathcal{P}_S$ realizes an orthogonal projection into the subspace of symmetric matrices of order n and $\mathcal{P}_{QG}$ realizes an orthogonal projection into the intersection of the subspace of matrices having the same sparsity pattern as $J^T J$ and the linear manifold of matrices satisfying quasi-Newton condition $W s_k = y_k$ with $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$. Thus

$$
\mathcal{P}_S W = (W + W^T)/2,
$$

$$
\begin{aligned}
(\mathcal{P}_G W)_{ij} &= W_{ij}, & (J^T J)_{ij} \neq 0, \\
(\mathcal{P}_G W)_{ij} &= 0, & (J^T J)_{ij} = 0,
\end{aligned}
$$

for a given square matrix W and

$$
\mathcal{P}_{QG}(J_{k+1}^T J_{k+1}) = \mathcal{P}_G(J_{k+1}^T J_{k+1} + u_k s_k^T),
$$

where $u_k \in R^n$ is a solution to the linear system $D_k u_k = y_k - J_{k+1}^T J_{k+1} s_k$ with diagonal matrix D_k such that

$$
(D_k)_{ii} = \sum_{(J^T J)_{ij} \neq 0} (e_j^T s_k)^2
$$

(e_j is the j-th column of the unit matrix). Subroutine `PGAM` uses the dog-leg trust-region strategy described in Section 3.2.

6 Methods for sparse systems of nonlinear equations

Consider the system of nonlinear equations

$$
f(x) = 0,
$$

where $f : R^n \to R^n$ is a continuously differentiable mapping and assume that the Jacobian matrix $J(x) = [J_{ij}(x)] = [\partial f_i(x)/\partial x_j]$ is sparse. The sparsity pattern of the Jacobian

matrix is stored in the standard compressed-row format using arrays IAG and JAG in the way described in Section 4. Let A be an approximation of the Jacobian matrix $J = J(x)$ and let $F = F(x) = (1/2)\|f(x)\|^2$. Methods considered in this section are realized in the line-search framework. They generate a sequence of points $x_i \in R^n$, $i \in N$, such that

$$x_{k+1} = x_k + \alpha_k d_k, \quad k \in N, \tag{27}$$

where $d_k \in R^n$ is the direction vector determined as an approximate solution of the linear system $A_k d + f_k = 0$ such that

$$\|A_k d_k + f_k\| \le \overline{\omega}_k \|f_k\| \tag{28}$$

with the precision $0 \le \overline{\omega}_k \le \overline{\omega} < 1$ and α_k is the step-size 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

$$F_{k+1} - F_k \le -\varepsilon_1 \alpha_k f_k^T A_k d_k,$$

with the line search parameter $0 < \varepsilon_1 < 1/2$ (parameter TOLS in the subroutines PEQN and PEQL). We use 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 the subroutines PEQN and PEQL).

To obtain a superlinear rate of convergence, the condition $\overline{\omega}_k \to 0$ has to be satisfied. Therefore, we choose $\overline{\omega}_k = \min(\max(\|f_k\|^\nu, \gamma(\|f_k\|/\|f_{k-1}\|)^\alpha), 1/k, \overline{\omega})$, with the values $\nu = 1/2$, $\gamma = 1$, $\alpha = (1+\sqrt{5})/2$ and $\overline{\omega} = 1/2$.

If $A_k \ne J_k$, then a safeguard based on restarts is used. It consists in setting $A_{k+1} = J_{k+1}$ if $j > \underline{j}$ or $A_k = J_k$ (with repeating the k-th iteration) if $j > \overline{j}$, where $0 < \underline{j} < \overline{j}$. We use the values $\underline{j} = 1$ and $\overline{j} = 5$. The restart of the form $A_k = J_k$ is also used whenever $-d_k^T J_k^T f_k \le \underline{\varepsilon}\|d_k\|\|J_k^T f_k\|$, where $0 < \underline{\varepsilon} < 1$ is a restart tolerance (parameter TOLD in the subroutines PEQN and PEQL).

The direction vector d_k (an approximate solution of the linear system $A_k d + f_k = 0$) is determined by using the preconditioned smoothed CGS method. To simplify the description of this method, we omit the outer index k and denote the inner index by i. Let $h = A^T f$. We set $s_1 = 0$, $\overline{s}_1 = 0$, $r_1 = f$, $\overline{r}_1 = f$, $p_1 = f$, $u_1 = f$ and for $i = 1, 2, 3, \ldots$ we proceed in the following way. If $\|r_i\| \le \overline{\omega}\|f\|$, then set $d = s_i$ and terminate the process. Otherwise compute

$$\begin{aligned}
v_i &= AC^{-1}p_i, \quad \alpha_i = h^T\overline{r}_i / h^T v_i, \\
q_i &= u_i - \alpha_i v_i, \\
\overline{s}_{i+1} &= \overline{s}_i + \alpha_i C^{-1}(u_i + q_i), \\
\overline{r}_{i+1} &= \overline{r}_i + \alpha_i AC^{-1}(u_i + q_i), \quad \beta_i = h^T\overline{r}_{i+1} / h^T\overline{r}_i, \\
u_{i+1} &= \overline{r}_{i+1} + \beta_i q_i, \\
p_{i+1} &= u_{i+1} + \beta_i(q_i + \beta_i p_i), \\
[\lambda_i, \mu_i]^T &= \arg \min_{[\lambda,\mu]^T \in \mathcal{R}^2} \|\overline{r}_{i+1} + \lambda(r_i - \overline{r}_{i+1}) + \mu v_i\|, \\
s_{i+1} &= \overline{s}_{i+1} + \lambda_i(s_i - \overline{s}_{i+1}) + \mu_i C^{-1} p_i, \\
r_{i+1} &= \overline{r}_{i+1} + \lambda_i(r_i - \overline{r}_{i+1}) + \mu_i v_i.
\end{aligned}$$

Matrix C serves as a preconditioner. The choice $C = I$ is used if `MOS2` $= 0$ or C is defined as an incomplete LU decomposition of matrix A if `MOS2` $\neq 0$ (`MOS2` is a parameter of the subroutines `PEQN` and `PEQL`). If `MOS2` > 0, a preliminary solution obtained by the incomplete Choleski decomposition can be accepted. In this case, we first compute vectors $d_1 = -C^{-1}f$, $r_1 = Ad_1 + f$. If $\|r_1\| \leq \overline{\omega}\|f\|$, then we set $d = d_1$ and terminate the process, otherwise we continue by CGS iterations as above.

More details concerning globally convergent line-search methods for systems of nonlinear equations can be found in [18].

6.1 Inexact discrete Newton method

Subroutine `PEQN` is an implementation of the inexact discrete Newton method. This simple method is based on elementwise differentiation. We always set $A_k = J(x_k)$, where

$$J_{ij}(x) = \frac{f_i(x + \delta_j e_j) - f_i(x)}{\delta_j}$$

for all pairs (i, j) corresponding to structurally nonzero elements of $J(x)$. Thus we need m scalar function evaluations (i.e. m/n equivalent vector function evaluations), where m is the number of structurally nonzero elements of $J(x)$.

6.2 Inverse column-update quasi-Newton method

Subroutine `PEQL` is an implementation of the inverse column update method, which is introduced in [21]. This method uses an approximation $S_k = A_k^{-1}$ of the inverse Jacobian matrix J_k^{-1} in (28). Therefore, we simply set $d_k := -S_k f_k$ instead of using the preconditioned smoothed CGS method if the restart is not used (if $A_k \neq J_k$). Denote by $s_k = x_{k+1} - x_k$, $s_{k-1} = x_k - x_{k-1}$, ..., $s_{k-m} = x_{k-m+1} - x_{k-m}$ and $y_k = f_{k+1} - f_k$, $y_{k-1} = f_k - f_{k-1}$, ..., $y_{k-m} = f_{k-m+1} - f_{k-m}$ the last m differences of points and function vectors, respectively, where the lower index $k - m$ corresponds to the iteration with the restart. Let $e_{k-1} = \arg\max_{e_i} |e_i^T y_{k-1}|$, ..., $e_{k-m} = \arg\max_{e_i} |e_i^T y_{k-m}|$ (arg max is taken over all columns e_i, $1 \leq i \leq n$, of the unit matrix). Then the vector $S_k f_k$ can be computed by the formula

$$S_k f_k = S_{k-m} f_k + \frac{e_{k-1}^T f_k}{e_{k-1}^T y_{k-1}} v_{k-1} + \ldots + \frac{e_{k-m}^T f_k}{e_{k-m}^T y_{k-m}} v_{k-m},$$

where $v_{k-1} = d_{k-1} - S_{k-1} y_{k-1}$, ..., $v_{k-m} = d_{k-m} - S_{k-m} y_{k-m}$ are vectors computed recursively by the formula

$$S_k y_k = S_{k-m} y_k + \frac{e_{k-1}^T y_k}{e_{k-1}^T y_{k-1}} v_{k-1} + \ldots + \frac{e_{k-m}^T y_k}{e_{k-m}^T y_{k-m}} v_{k-m}.$$

In both of these formulae we use the matrix $S_{k-m} = (L_{k-m} U_{k-m})^{-1}$, where $L_{k-m} U_{k-m}$ is the incomplete LU decomposition of the Jacobian matrix $J(x_{k-m})$. Note that the vectors e_{k-1}, ..., e_{k-m} do not need to be stored. We only use indices of their unique nonzero elements. The limited memory column update method needs to be restarted periodically after $\overline{m}$ iterations (parameter `MF` in the subroutine `PEQL`), since at most $\overline{m}$ vectors can be stored.

7 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 $c : R^n \to R^m$ 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` in the way described in Section 3. The sparsity pattern of the Jacobian matrix is stored using arrays `ICG` and `JCG`, where `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 `NC`+1 elements and the last element is equal to `MC`+1, where `MC` is the number of nonzero elements.

Applying the Newton method to the system

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

of $n + m$ nonlinear equations for unknown vectors $x \in R^n$ and $u \in R^m$ (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

$$\begin{bmatrix} G(x_k, u_k) & A(x_k) \\ A(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} d_k^x \\ d_k^u \end{bmatrix} = - \begin{bmatrix} g(x_k, u_k) \\ c(x_k) \end{bmatrix}, \tag{29}$$

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 step-size α_k. We use augmented Lagrangian function

$$P_k(\alpha) = F(x_k + \alpha d_k^x) + (u_k + d_k^u)^T c(x_k + \alpha d_k^x) + \frac{\sigma}{2} \|c(x_k + \alpha d_k^x)\|^2 \tag{30}$$

in our subroutines, where $\sigma \geq 0$ is a penalty parameter (parameter `RPF1` in subroutines `PIND` and `PNUL`). It can be shown (see [19]) that if system (30) is solved in such a way that $\|G_k d_k^x + A_k d_k^u + g_k\| \leq \overline{\omega}_k \|g_k\|$ and $\|A_k^T d_k^x + c_k\| \leq \overline{\omega}_k \|c_k\|$ with $0 < \overline{\omega}_k < 1$ and if $\sigma > -(d_k^x)^T G_k d_k^x / \left((1 - \overline{\omega}_k)\|c_k\|\right)$, then $P_k'(0)$ (first order derivative of $P_k(\alpha)$ at $\alpha = 0$) is negative so that $P_k(\alpha)$ decreases in 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 $\sigma \geq 0$ and replace G_k by a diagonal positive definite matrix D_k (restart) if $P'_k(0) \geq 0$. Using this D_k for the construction of an indefinite preconditioner (see Section 7.1), we obtain the exact solution of (30) 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 $\sigma \geq 0$. 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 $\underline{\beta}\alpha_k^j \leq \alpha_k^{j+1} \leq \overline{\beta}\alpha_k^j$ with $0 < \underline{\beta} \leq \overline{\beta} < 1$, satisfying

$$P(\alpha_k) - P(0) \leq \varepsilon_1 \alpha_k P'_k(0)$$

with the line search parameter $0 < \varepsilon_1 < 1/2$ (parameter `TOLS` in the subroutines `PIND` and `PNUL`). We use 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 the 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 (29) can be written in the form

$$Kd = \begin{bmatrix} B & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} d^x \\ d^u \end{bmatrix} = \begin{bmatrix} b^x \\ b^u \end{bmatrix} = b, \tag{31}$$

where B is an approximation of $G(x,u)$ computed by using differences of gradients of the Lagrangian function in the same way as in subroutines `PNED` and `PNEC` described in Section 3 (the algorithm proposed in [27] is again used).

7.1 Methods utilizing indefinite preconditioners

In subroutine `PIND`, directions d^x and d^u are computed directly from linear KKT system (31) by the preconditioned conjugate gradient method. The indefinite preconditioner

$$C = \begin{bmatrix} D & A \\ A^T & 0 \end{bmatrix} \tag{32}$$

investigated in [19] is used, where D is a positive definite diagonal matrix derived from the diagonal of B. Thus multiplication by C^{-1} can be expressed in the form

$$\begin{aligned} C^{-1}r &= \begin{bmatrix} D^{-1} - D^{-1}A(A^TD^{-1}A)^{-1}A^TD^{-1} & D^{-1}A(A^TD^{-1}A)^{-1} \\ (A^TD^{-1}A)^{-1}A^TD^{-1} & -(A^TD^{-1}A)^{-1} \end{bmatrix} \begin{bmatrix} r^x \\ r^u \end{bmatrix} \\ &= \begin{bmatrix} D^{-1}(r^x - At^u) \\ t^u \end{bmatrix}, \qquad t^u = (A^TD^{-1}A)^{-1}(A^TD^{-1}r^x - r^u). \end{aligned} \tag{33}$$

More precisely, we set $d_1 = 0$, $r_1 = b$, $t_1^u = (A^TD^{-1}A)^{-1}(A^TD^{-1}r_1^x - r_1^u)$, $p_1 = t_1$, $t_1^x = D^{-1}(r_1^x - At_1^u)$ and for $i = 1, 2, 3, \ldots$ we proceed in the following way. If $\|r_i^x\| \leq \overline{\omega}\|b^x\|$ and $\|r_i^u\| \leq \overline{\omega}\|b^u\|$, then set $d = d_i$ and terminate the computation. Otherwise compute

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

(we use vectors

$$
r = \begin{bmatrix} r^x \\ r^u \end{bmatrix} \quad t = \begin{bmatrix} t^x \\ t^u \end{bmatrix} \quad p = \begin{bmatrix} p^x \\ p^u \end{bmatrix} \quad q = \begin{bmatrix} q^x \\ q^u \end{bmatrix}
$$

in this description).

In the above algorithm, multiplication by 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 $A^T = [A_s^T, A_d^T]$ 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`).

7.2 Methods based on null-space transformation

Consider the unique representation $d^x = Zd^z + D^{-1} A d^a$, 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 equation $A^T v = 0$ (thus $Z^T A = 0$). Using the second equation in (31), we get $d^a = (A^T D^{-1} A)^{-1} b^u$ and the first one implies $BZd^z = b^x - BD^{-1} A d^a - Ad^u$, which after premultiplying by Z^T gives

$$
Z^T B Z d^z = b^z, \tag{34}
$$

where $b^z = Z^T (b^x - BD^{-1} A d^a)$. Thus d^z can be obtained by the conjugate gradient method with preconditioner $Z^T D Z$ applied to equation (34). Unfortunately, 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 [9]. In subroutine

PNUL, 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, 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 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 $\tilde{r}_i^T\tilde{t}_i \leq \overline{\omega}\gamma$, then 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, multiplication by matrix $(A^TD^{-1}A)^{-1}$ is again used. Thus we proceed in the same way as in Section 7.1 if A has dense rows (subroutine PNUL also uses parameters MD and MDE) .

8 Description of subroutines

In this section we describe easy-to-use subroutines PLISU, PLISS, PLIPU, PLIPS, PNEDU, PNEDS, PNECU, PNECS, PSEPU, PSEPS, PGAMU, PGAMS, PGANU, PGANS, PEQNU, PEQLU, 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). Beside 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,NIIT,NIT,NFV,NFG,NFH
```

Its elements have the following meanings:

Element	Type	Significance
NRES	IO	Number of restarts.
NDEC	IO	Number of matrix decompositions.
NIIT	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 PLISU(NF,X,RA,NRA,IPAR,RPAR,F,GMAX,MF,ITERM)
CALL PLISS(NF,NB,X,IX,XL,XU,RA,NRA,IPAR,RPAR,F,GMAX,MF,ITERM)
CALL PLIPU(NF,X,RA,NRA,IPAR,RPAR,F,GMAX,MF,ITERM)
CALL PLIPS(NF,NB,X,IX,XL,XU,RA,NRA,IPAR,RPAR,F,GMAX,MF,ITERM)
CALL PNEDU(NF,X,IH,JH,NH,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PNEDS(NF,NB,X,IX,XL,XU,IH,JH,NH,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PNECU(NF,X,IH,JH,NH,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PNECS(NF,NB,X,IX,XL,XU,IH,JH,NH,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PSEPU(NF,NA,X,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PSEPS(NF,NA,NB,X,IX,XL,XU,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PGAMU(NF,NA,X,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PGAMS(NF,NA,NB,X,IX,XL,XU,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PGANU(NF,NA,X,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PGANS(NF,NA,NB,X,IX,XL,XU,AF,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,ITERM)
CALL PEQNU(NF,X,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,AF,F,GMAX,ITERM)
CALL PEQLU(NF,X,IAG,JAG,IA,NIA,RA,NRA,IPAR,RPAR,AF,F,GMAX,MF,ITERM)
CALL PINDU(NF,NC,X,IH,JH,NH,CF,ICG,JCG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,CMAX,ITERM)
CALL PNULU(NF,NC,X,IH,JH,NH,CF,ICG,JCG,IA,NIA,RA,NRA,IPAR,RPAR,F,GMAX,CMAX,ITERM)
```

Their arguments have the following meanings:

Argument	Type	Significance
NF	II	Number of variables of the objective.
NA	II	Number of particular functions.
NB	II	Specification whether the simple bounds are suppressed (NB = 0) or accepted (NB > 0).
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.
IX(NF)	II	Vector containing the simple bound types (significant only if NB > 0): IX(I) = 0: the variable X(I) is unbounded, IX(I) = 1: the lower bound X(I) ≥ XL(I), IX(I) = 2: the upper bound X(I) ≤ XU(I),

		IX(I) = 3: the two-side bound XL(I) ≤ X(I) ≤ XU(I), IX(I) = 5: the variable X(I) is fixed (given by its initial estimate).
XL(NF)	RI	Vector with lower bounds for variables (significant only if NB > 0).
XU(NF)	RI	Vector with upper bounds for variables (significant only if NB > 0).
F	RO	Value of the objective function at the solution X.
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.
NH	II	Dimension of array JH(NH). Since this array is also used for storing the Choleski decomposition, NH has to be greater than MH=IH(NF+1)-1 (the number of nonzero elements in the sparse Hessian matrix). We recommend the value NH=3*MH
AF(NA)	RO	Vector which contains values of particular functions.
IAG(NA+1)	IA	Pointers of the first elements in rows of the Jacobian matrix.
JAG(MA)	IA	Column indices of nonzero elements of the Jacobian matrix.
CF(NC+1)	RA	Vector which contains 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.
IA(NIA)	IA	Working array of the dimension NIA (see Table 2).
RA(NRA)	RA	Working array of the dimension NRA (see Table 2).
IPAR(8)	IA	Integer parameters (see Table 3).
RPAR(9)	RA	Real parameters (see Table 3).
GMAX	RO	Maximum absolute value of a particular derivative of the Lagrangian function.
CMAX	RO	Maximum constraint violation.
MF	II	The number of limited-memory variable metric updates in each iteration.
ITERM	IU	Variable that indicates the cause of termination: ITERM = 1: if $\lvert X - X_{old}\rvert$ was less than or equal to TOLX in two subsequent iterations, ITERM = 2: if $\lvert F - F_{old}\rvert$ was less than or equal to TOLF 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.

Dimension NH of array JH has to be sufficiently large to include indices of new nonzero elements of the Choleski factor. This value should be greater than $3 * MH$, where $MH = IH(NF + 1) - 1$ is the number of nonzero elements in the original pattern. Dimensions NIA and NRA of working arrays IA and RA should be greater than values introduced in Table 2. When NH, NIA, NRA are

not sufficiently large, then a message containing the required values is printed and computation is terminated. This standard action can be suppressed by the input choice ITERM < 0 (this is advantageous in case the recommended values are too large). Sometimes, when the number of new nonzero elements is too large, the recommended values can be insufficient. In this case, which is detected by the output value ITERM (ITERM > 30), values NH and NRA should be increased. Table 2 contains additional values which are not input parameters. Thus MA = IAG(NA + 1) − 1 and MC = ICG(NC + 1) − 1 are numbers of nonzero elements in the Jacobian patterns (dimensions of arrays JAG and JCG). Moreover, denoting KA(I) = IAG(I + 1) − IAG(I) and KC(I) = ICG(I + 1) − ICG(I) the numbers of nonzero elements in the corresponding rows of Jacobian matrices, we define

$$\begin{aligned} \texttt{KH} &= \sum_{\texttt{I}=1}^{\texttt{NA}} \texttt{KA(I)} * (\texttt{KA(I)} - 1)/2, \\ \texttt{KA} &= \texttt{MAX}(\texttt{KA(I)} * (\texttt{KA(I)} - 1)/2, \quad 1 \le \texttt{I} \le \texttt{NA}), \\ \texttt{KC} &= \texttt{MAX}(\texttt{KC(I)} * (\texttt{KC(I)} - 1)/2, \quad 1 \le \texttt{I} \le \texttt{NC}). \end{aligned}$$

Value MD is described in Table 3.

subroutine	NIA	NRA
PLIS	-	2*(MF+1)*NF+2*MF
PLIP	-	(MF+5)*NF+2*MF
PNED	8*NF+6	5*NF+1+3*NH
PNEC	4*NF+3	7*NF+1+2*NH
PSEP	8*NF+NA+6+2*KH	5*NF+2*MA+3*KH
PGAN	8*NF+6+2*KH	6*NF+2*KH+KA
PGAM	8*NF+6+2*KH	6*NF+2*KH
PEQN	5*NF	11*NF+2*MA
PEQL	6*NF	11*NF+2*MA+(NF-1)*MF
PIND	4*NF+6*(NC+1)+2*KC	7*NF+(MD+8)*NC+MC+MD*(MD+3)/2+2*(MH+KC+1)
PNUL	4*NF+6*(NC+1)+2*KC	7*NF+(MD+5)*NC+MC+MD*(MD+3)/2+2*(MH+KC+1)

Table 2: Dimensions of working arrays

The integer and real parameters listed in Table 3 have the following meanings:

Argument	Type	Significance
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.
MIT	II	Maximum number of iterations; the choice MIT = 0 causes that the default value MIT = 9000 in PLIS, PLIP, PSEP or MIT = 5000 in PNED, PNEC, PGAM, PGAN or MIT = 1000 in the other subroutines will be taken.
MFV	II	Maximum number of function evaluations; the choice MFV = 0 causes that the default value MIT = 9000 in PLIS, PLIP, PSEP or MIT = 5000 in PNED, PNEC, PGAM, PGAN or MIT = 1000 in the other subroutines will be taken.

Parameter	PLIS	PLIP	PNED	PNEC	PSEP	PGAM PGAN	PEQN PEQL	PIND PNUL
IPAR(1)	IPRNT	IPRNT	IPRNT	IPRNT	IPRNT	IPRNT	IPRNT	IPRNT
IPAR(2)	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT
IPAR(3)	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV
IPAR(4)	MFG	MFG	MFG	MFG	MFG	MFG	MES	MFG
IPAR(5)	IEST	IEST	IEST	IEST	IEST	IEST	MOS1	MES
IPAR(6)	MES	MES	-	MOS2	MES	-	MOS2	MOS
IPAR(7)	-	MEC	-	MOS3	MET	-	MOS3	MD
IPAR(8)	-	MET	-	-	-	-	-	MDE
RPAR(1)	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX
RPAR(2)	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX
RPAR(3)	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLC
RPAR(4)	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLG
RPAR(5)	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLD
RPAR(6)	TOLD	TOLD	XDEL	XDEL	TOLD	XDEL	TOLD	TOLS
RPAR(7)	TOLS	TOLS	FMIN	FMIN	TOLS	FMIN	TOLS	RPF1
RPAR(8)	TOLP	TOLP	-	-	TOLP	ETA	ETA2	-
RPAR(9)	FMIN	FMIN	-	-	FMIN	-	-	-

Table 3: Integer and real parameters

MFG II Maximum number of gradient evaluations; the choice MFG = 0 causes that the default value MFG = 10000 will be taken.

IEST II Estimation of the minimum function value for the line search:

IEST = 0: estimation is not used,
IEST = 1: lower bound FMIN is used as an estimation for the minimum function value.

MES II Variable that specifies the interpolation method selection in a line search:

MES = 1: bisection,
MES = 2: two-point quadratic interpolation,
MES = 3: three-point quadratic interpolation,
MES = 4: three-point cubic interpolation.

The choice MES = 0 causes that the default value MES = 4 in PLIS, PLIP, PSEP or MES = 1 in PEQN, PEQL, PIND, PNUL will be taken.

MEC II Variable that determines the correction parameter:

MEC = 1: unit value,
MEC = 2: balancing value,
MEC = 3: square root,
MEC = 4: geometric mean,
MEC = 5: ratio of shift parameters.

The choice MEC = 0 causes that the default value MEC = 4 will be taken.

MET II In PLIP: Variable that specifies the limited-memory method:

MET = 1: rank-one method,
MET = 2: rank-two method.

The choice MET = 0 causes that the default value MET = 2 will be taken.

In PSEP: Variable that specifies partitioned variable metric update:

MET = 1: the BFGS update,
MET = 2: combination of the BFGS and the rank-one updates.

The choice MET = 0 causes that the default value MET = 2 will be taken.

MOS II Choice of preconditioning strategy:

MOS = 1: preconditioning by the constraint preconditoner with complete Gill-Murray decomposition.
MOS = −1: preconditioning by the constraint preconditioner with incomplete Gill-Murray decomposition,

The choice MOS = 0 causes that the default value MOS = 1 will be taken.

MOS1 II Variable that specifies dual vector in the CGS method:

MOS1 = 1: gradient of the objective function is used,
MOS1 = 2: vector containing values of particular functions is used.

The choice MOS1 = 0 causes that the default value MOS1 = 1 in PEQN and MOS1 = 2 in PEQL will be taken.

MOS2 II Choice of preconditioning strategy:

MOS2 = 0: preconditioning is not used,
MOS2 = −1: in PNEC: preconditioning by the incomplete Gill-Murray decomposition,
MOS2 = −1: in PEQN, PEQL: preconditioning by the incomplete LU decomposition,
MOS2 = 1: in PNEC: preconditioning by the incomplete Gill-Murray decomposition combined with preliminary solution of the preconditioned system.
MOS2 = 1: in PEQN, PEQL: preconditioning by the incomplete LU decomposition combined with preliminary solution of the preconditioned system.

MOS3 II In PNEC: Number of Lanczos steps for determination of the Levenberg-Marquardt parameter (recommended value is MOS3 = 5).

In PEQN, PEQL: Variable that specifies the smoothing strategy for the CGS method:

MOS3 = 1: smoothing is not used,
MOS3 = 2: single smoothing strategy is used,
MOS3 = 3: double smoothing strategy is used.

The choice MOS3 = 0 causes that the default value MOS3 = 3 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 = 10^5 in PEQN, PEQL or XMAX = 10^3 in the other subroutines 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^{-16} in PEQN, PEQL, PIND, PNUL or TOLX = 10^{-12} in the other subroutines will be taken.
TOLF	RI	Tolerance for the change of function values; the choice TOLF = 0 causes that the default value TOLF = 10^{-16} in PEQN, PEQL or TOLF = 10^{-15} in PGAM, PGAN or TOLF = 10^{-14} in the other subroutines 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.
TOLB	RI	Minimum acceptable function value; the choice TOLB = 0 causes that the default value TOLB = 10^{-16} in PEQN, PEQL or TOLB = FMIN + 10^{-15} in the others subroutines will be taken.
TOLG	RI	Tolerance for the Lagrangian function gradient; the choice TOLG = 0 causes that the default value 10^{-5} in PGAM, PGAN or TOLG = 10^{-6} in the other subroutines will be taken.
TOLD	RI	Tolerance for a descent direction; the choice TOLD = 0 causes that the default value TOLD = 10^{-4} in PLIS, PLIP, PSEP, or TOLD = 10^{-15} in PEQN,PEQL or TOLD = 10^{-5} in PIND, PNUL will be taken.
TOLS	RI	Tolerance parameter for a function decrease in the line search; the choice TOLS = 0 causes that the default value TOLS = 10^{-4} will be taken.
TOLP	RI	Tolerance parameter for directional derivative increase in the line search; the choice TOLP = 0 causes that the default value TOLP = 0.8 in PLIS or TOLP = 0.9 in PLIP, PSEP will be taken.
XDEL	RI	Trust region step-size; the choice XDEL = 0 causes that a suitable default value will be computed.
FMIN	RI	Lower bound for the minimum function value.
ETA	RI	Parameter for switch between the Gauss-Newton method and variable metric correction; the choice ETA = 0 causes that the default value ETA = $1.5 \cdot 10^{-4}$ will be taken.
ETA2	RI	Damping parameter for an incomplete LU preconditioner.
RPF1	RI	Value of the penalty parameter in the merit function; the choice RPF1 = 0 causes that the default value RPF1 = 10^{-4} will be taken.

The subroutines PLISU,PLISS,PLIPU,PLIPS,PNEDU,PNEDS,PNECU,PNECS require the user supplied subroutines OBJ,DOBJ that define the objective function and its gradient and have the form

```
SUBROUTINE  OBJ(NF,X,F)
SUBROUTINE DOBJ(NF,X,G)
```

The subroutines PSEPU,PSEPS,PGAMU,PGAMS,PGANU,PGANS require the user supplied subroutines FUN,DFUN that define particular functions and their gradients and have the form

```
SUBROUTINE  FUN(NF,KA,X,FA)
SUBROUTINE DFUN(NF,KA,X,GA)
```

The subroutines PEQNU,PEQLU require the user supplied subroutine FUN that defines particular functions and has the form

```
SUBROUTINE  FUN(NF,KA,X,FA)
```

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.
KA	II	Index of the particular 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.
FA	RO	Value of the KA-th particular 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.
GA(NF)	RO	Gradient of the KA-th particular function at the point X.
GC(NF)	RO	Gradient of the KC-th constraint function at the point X.

9 Verification of subroutines

In this section we report the results obtained by using test programs TLISU, TLISS, TLIPU, TLIPS, TNEDU, TNEDS, TNECU, TNECS, TSEPU, TSEPS, TGAMU, TGAMS, TGANU, TGANS, TEQNU, TEQLU, TINDU, TNULU which serve for demonstration, verification and testing of subroutines PLISU, PLISS, PLIPU, PLIPS, PNEDU, PNEDS, PNECU, PNECS, PSEPU, PSEPS, PGAMU, PGAMS, PGANU, PGANS, PEQNU, PEQLU, 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 final value of the objective function F, 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 2000 system using the Digital Visual Fortran (Version 6) compiler, in double-precision arithmetic. All subroutines were checked with a Fortran verifier and also implemented and tested on various UNIX workstations (Digital, Silicon Graphics, Hewlet Packard).

Problem	NIT	NFV	NFG	F	G	ITERM
1	4988	5554	5554	0.96378001D−14	0.8905D−06	4
2	411	440	440	0.35727672D+01	0.1559D−05	2
3	74	78	78	0.65510169D−09	0.5391D−06	4
4	103	112	112	0.26949954D+03	0.8990D−06	4
5	24	26	26	0.13063928D−11	0.6711D−06	4
6	30	31	31	0.21610223D−10	0.9461D−06	4
7	38	43	43	0.33513743D+03	0.7296D−06	4
8	29	33	33	0.76177495D+06	0.4315D−03	2
9	13	16	16	0.31643614D+03	0.3686D−06	4
10	1540	1582	1582	−0.12463000D+03	0.1240D−04	2
11	107	116	116	0.10776588D+02	0.1754D−05	2
12	366	376	376	0.98227362D+03	0.1215D−04	2
13	36	37	37	0.87510394D−14	0.2230D−07	4
14	10	12	12	0.12872917D−08	0.9163D−06	4
15	2092	2157	2157	0.19240160D+01	0.9344D−06	4
16	193	203	203	−0.42740448D+03	0.8227D−05	2
17	1007	1032	1032	−0.37992109D−01	0.8761D−06	4
18	1449	1474	1474	−0.24574119D−01	0.8616D−06	4
19	1393	1431	1431	0.59598624D+02	0.2592D−05	2
20	2129	2191	2191	−0.10001352D+01	0.9083D−06	4
21	3090	3166	3166	0.21386638D+01	0.9973D−06	4
22	1305	1346	1346	0.10000000D+01	0.9817D−06	4

Table 4: Results obtained by program TLISU

Problem	NIT	NFV	NFG	F	G	ITERM
1	5063	5738	5738	0.00000000D+00	0.0000D+00	3
2	3167	4664	4664	0.39264596D+04	0.6260D−04	2
3	113	124	124	0.45950339D−12	0.6001D−06	4
4	59	64	64	0.26952269D+03	0.8381D−06	4
5	24	26	26	0.13063928D−11	0.6711D−06	4
6	30	31	31	0.21610223D−10	0.9461D−06	4
7	33	40	40	0.33772248D+03	0.5918D−06	4
8	50	55	55	0.76192573D+06	0.2319D−03	2
9	505	508	508	0.42805692D+03	0.3340D−07	4
10	1176	1242	1242	−0.81091359D+02	0.1585D−04	2
11	17	18	18	0.96517295D+05	0.1714D−05	2
12	173	177	177	0.49942141D+04	0.5035D−05	2
13	36	37	37	0.87510394D−14	0.2230D−07	4
14	10	12	12	0.12872917D−08	0.9163D−06	4
15	2092	2157	2157	0.19240160D+01	0.9344D−06	4
16	178	184	184	−0.42739165D+03	0.1069D−04	2
17	1007	1032	1032	−0.37992109D−01	0.8761D−06	4
18	1449	1474	1474	−0.24574119D−01	0.8616D−06	4
19	1561	1595	1595	0.16549452D+04	0.1117D−04	2
20	2075	2121	2121	−0.10001352D+01	0.9157D−06	4
21	1181	1205	1205	0.24135487D+01	0.9447D−06	4
22	1562	1598	1598	0.10000000D+01	0.7856D−06	4

Table 5: Results obtained by program TLISS

Problem	NIT	NFV	NFG	F	G	ITERM
1	5383	5417	5417	0.60102266D−13	0.5992D−06	4
2	541	571	571	0.35727672D+01	0.1404D−05	2
3	125	128	128	0.33827028D−12	0.5181D−06	4
4	109	114	114	0.26949954D+03	0.6692D−06	4
5	26	27	27	0.71007240D−11	0.9508D−06	4
6	35	36	36	0.14294227D−10	0.7368D−06	4
7	36	41	41	0.33693718D+03	0.9561D−06	4
8	33	36	36	0.76177495D+06	0.1925D−02	2
9	15	18	18	0.31643614D+03	0.2636D−06	4
10	1941	1957	1957	−0.12495000D+03	0.2178D−04	2
11	140	146	146	0.10776588D+02	0.5924D−06	4
12	415	418	418	0.98227362D+03	0.1279D−04	2
13	40	41	41	0.49777715D−12	0.8522D−06	4
14	8	10	10	0.12883424D−08	0.9771D−06	4
15	1239	1294	1294	0.19240160D+01	0.9144D−06	4
16	248	253	253	−0.42740448D+03	0.5924D−05	2
17	598	604	604	−0.37992109D−01	0.9079D−06	4
18	989	998	998	−0.24574119D−01	0.9751D−06	4
19	1261	1272	1272	0.59598624D+02	0.4099D−05	2
20	2336	2360	2360	−0.10001352D+01	0.8626D−06	4
21	2489	2519	2519	0.21386638D+01	0.8705D−06	4
22	1261	1292	1292	0.10000000D+01	0.9269D−06	4

Table 6: Results obtained by program TLIPU

Problem	NIT	NFV	NFG	F	G	ITERM
1	5263	5321	5321	0.53013200D−13	0.3705D−05	2
2	2293	2447	2447	0.39304496D+04	0.2507D−04	2
3	127	132	132	0.21055015D−11	0.4368D−06	4
4	70	72	72	0.26952269D+03	0.7939D−06	4
5	26	27	27	0.71007240D−11	0.9508D−06	4
6	35	36	36	0.14294227D−10	0.7368D−06	4
7	38	44	44	0.33693718D+03	0.8210D−06	2
8	59	68	68	0.76192573D+06	0.6760D−03	2
9	508	510	510	0.42805692D+03	0.7761D−06	4
10	1299	1327	1327	−0.82540057D+02	0.1576D−04	2
11	16	17	17	0.96517295D+05	0.1463D−03	2
12	181	184	184	0.49942141D+04	0.1045D−04	2
13	40	41	41	0.49777715D−12	0.8522D−06	4
14	8	10	10	0.12883424D−08	0.9771D−06	4
15	1239	1294	1294	0.19240160D+01	0.9144D−06	4
16	227	228	228	−0.42739165D+03	0.9524D−05	2
17	598	604	604	−0.37992109D−01	0.9079D−06	4
18	989	998	998	−0.24574119D−01	0.9751D−06	4
19	1367	1383	1383	0.16549452D+04	0.1053D−04	2
20	2284	2297	2297	−0.10001352D+01	0.8125D−06	4
21	1184	1190	1190	0.24135487D+01	0.9597D−06	4
22	1361	1381	1381	0.10000000D+01	0.9624D−06	4

Table 7: Results obtained by program TLIPS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1419	1423	5680	0.39866239D+01	0.1307D−09	4
2	39	45	200	0.23140639D−14	0.3499D−06	4
3	17	18	108	0.83978290D−09	0.9328D−06	4
4	24	25	100	0.26949954D+03	0.6658D−10	4
5	11	12	72	0.79510946D−10	0.4730D−06	4
6	13	16	196	0.12594486D−10	0.8150D−06	4
7	12	13	78	0.33693718D+03	0.3004D−06	4
8	4	5	90	0.76177495D+06	0.2155D−06	4
9	7	9	16	0.31643614D+03	0.1464D−06	4
10	78	80	711	−0.12523000D+03	0.2705D−09	4
11	67	68	408	0.10776588D+02	0.1987D−06	4
12	127	128	512	0.98227362D+03	0.4950D−09	4
13	6	7	28	0.59899867D−10	0.6926D−06	4
14	2	3	18	0.12901360D−08	0.7917D−06	4
15	9	10	40	0.19240160D+01	0.1999D−06	4
16	7	8	48	−0.42740448D+03	0.5655D−07	4
17	8	9	54	−0.37992109D−01	0.3144D−10	4
18	7	8	48	−0.24574119D−01	0.2184D−09	4
19	6	7	42	0.59598624D+02	0.9516D−08	4
20	14	15	90	−0.10001352D+01	0.1388D−08	4
21	11	12	72	0.21386638D+01	0.3314D−08	4
22	30	34	186	0.10000000D+01	0.1636D−08	4

Table 8: Results obtained by program TNEDU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1420	1424	5680	0.00000000D+00	0.0000D+00	3
2	128	130	640	0.19800505D+04	0.9110D−10	4
3	17	19	108	0.18935586D−09	0.3399D−06	4
4	10	12	44	0.26952269D+03	0.3280D−09	4
5	13	15	84	0.39190563D−12	0.5364D−06	4
6	13	14	196	0.13639663D−11	0.9010D−06	4
7	30	32	186	0.33692005D+03	0.1514D−05	2
8	37	38	684	0.76192573D+06	0.1187D−06	4
9	530	531	1062	0.42805692D+03	0.3267D−13	4
10	109	127	990	−0.80451821D+02	0.6391D−06	4
11	27	28	168	0.72291495D+05	0.3918D−08	4
12	519	520	2080	0.49942141D+04	0.2361D−06	4
13	3	4	16	0.66054208D−23	0.3631D−11	3
14	2	3	18	0.12901360D−08	0.7917D−06	4
15	9	10	40	0.19240160D+01	0.1999D−06	4
16	10	12	66	−0.42739165D+03	0.2023D−07	4
17	8	9	54	−0.37992109D−01	0.3144D−10	4
18	7	8	48	−0.24574119D−01	0.2184D−09	4
19	13	16	84	0.16549452D+04	0.1738D−08	4
20	14	15	90	−0.10001352D+01	0.1388D−08	4
21	9	10	60	0.24135487D+01	0.3885D−08	4
22	30	34	186	0.10000000D+01	0.1636D−08	4

Table 9: Results obtained by program TNEDS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1436	1439	5748	0.39866239D+01	0.1375D−08	4
2	79	89	400	0.16914409D−20	0.3819D−09	3
3	18	19	114	0.18069232D−09	0.3157D−06	4
4	24	25	100	0.26949954D+03	0.1357D−08	4
5	11	12	72	0.99092247D−10	0.5108D−06	4
6	17	21	252	0.16690487D−10	0.8977D−06	4
7	11	12	72	0.33693718D+03	0.6291D−06	4
8	5	6	108	0.76177495D+06	0.3794D−06	4
9	7	8	16	0.31643614D+03	0.3617D−08	4
10	66	72	603	−0.13351000D+03	0.1935D−06	4
11	71	72	432	0.10776588D+02	0.2373D−10	4
12	133	134	536	0.98227362D+03	0.2030D−07	4
13	7	8	32	0.40253017D−26	0.1527D−13	3
14	2	3	18	0.12902879D−08	0.8199D−06	4
15	10	11	44	0.19240160D+01	0.7151D−07	4
16	12	15	78	−0.42740448D+03	0.8938D−09	4
17	8	9	54	−0.37992109D−01	0.3913D−09	4
18	8	9	54	−0.24574119D−01	0.7054D−10	4
19	7	8	48	0.59598624D+02	0.1064D−08	4
20	10	11	66	−0.10001352D+01	0.2766D−11	4
21	11	12	72	0.21386638D+01	0.1542D−06	4
22	46	51	282	0.10000000D+01	0.3761D−08	4

Table 10: Results obtained by program TNECU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1433	1438	5736	0.00000000D+00	0.0000D+00	3
2	294	321	1470	0.39185930D+04	0.5125D−06	4
3	17	19	108	0.19126360D−09	0.3492D−06	4
4	10	12	44	0.26952269D+03	0.7327D−08	4
5	13	15	84	0.30904440D−13	0.1936D−06	4
6	13	14	196	0.32809593D−12	0.4112D−06	4
7	19	27	120	0.33741307D+03	0.3848D−06	4
8	37	38	684	0.76192573D+06	0.5363D−06	4
9	583	584	1168	0.42805692D+03	0.3174D−12	4
10	115	145	1044	−0.79522041D+02	0.5355D−07	4
11	29	30	180	0.72291495D+05	0.1091D−10	4
12	199	200	800	0.49942141D+04	0.2376D−06	4
13	4	5	20	0.65098612D−23	0.5040D−11	3
14	2	3	18	0.12902879D−08	0.8199D−06	4
15	10	11	44	0.19240160D+01	0.7151D−06	4
16	14	17	90	−0.42739165D+03	0.3095D−12	4
17	8	9	54	−0.37992109D−01	0.3913D−09	4
18	8	9	54	−0.24574119D−01	0.7054D−10	4
19	13	16	84	0.16549452D+04	0.1545D−08	4
20	10	11	66	−0.10001352D+01	0.2766D−11	4
21	9	10	60	0.24135487D+01	0.5172D−06	4
22	46	51	282	0.10000000D+01	0.3761D−08	4

Table 11: Results obtained by program TNECS

Problem	NIT	NFV	NFG	F	G	ITERM
1	2654	3627	3627	0.79478973D−16	0.2128D−06	3
2	105	179	179	0.83316140D+02	0.4982D−06	4
3	40	45	45	0.26700768D−12	0.8234D−06	4
4	47	58	58	0.27358820D+03	0.2785D−06	4
5	16	17	17	0.10602671D−11	0.7283D−06	4
6	38	40	40	0.54696139D−11	0.8818D−06	4
7	22	26	26	0.33525262D+03	0.1048D−06	4
8	26	40	40	0.76177495D+06	0.2954D−04	2
9	185	194	194	0.31643614D+03	0.1559D−05	2
10	214	245	245	−0.12255000D+03	0.2694D−04	2
11	92	115	115	0.86867306D+02	0.2975D−07	4
12	141	142	142	0.98227362D+03	0.8385D−06	4
13	36	37	37	0.79413669D−12	0.3125D−06	4
14	25	28	28	0.10428935D−08	0.9268D−06	4
15	8	14	14	0.19240160D+01	0.6038D−07	4
16	30	39	39	−0.42740448D+03	0.1512D−05	2
17	15	17	17	−0.37992109D−01	0.1413D−06	4
18	5	11	11	−0.24574119D−01	0.1165D−06	4
19	19	23	23	0.59598624D+02	0.4658D−06	4
20	42	98	98	−0.10001352D+01	0.5952D−07	4
21	37	40	40	0.21386638D+01	0.7673D−06	4
22	55	211	211	0.10000000D+01	0.6104D−07	4

Table 12: Results obtained by program TSEPU

Problem	NIT	NFV	NFG	F	G	ITERM
1	2591	3322	3322	0.00000000D+00	0.0000D+00	3
2	344	347	347	0.35121131D+02	0.1068D−06	4
3	39	43	43	0.44169182D−12	0.4247D−06	4
4	18	21	21	0.26952269D+03	0.1540D−06	4
5	13	16	16	0.41704722D+01	0.4256D−06	4
6	32	33	33	0.95952646D−11	0.8013D−06	4
7	19	21	21	0.33772248D+03	0.2470D−06	4
8	52	56	56	0.76192573D+06	0.7796D−04	2
9	1001	1003	1003	0.42805692D+03	0.1760D−06	4
10	147	180	180	−0.77737596D+02	0.1441D−04	2
11	27	28	28	0.72291495D+05	0.5862D−10	4
12	228	235	235	0.49942141D+04	0.3035D−06	4
13	36	37	37	0.79413669D−12	0.3125D−06	4
14	25	28	28	0.10428935D−13	0.9268D−06	4
15	8	14	14	0.19240160D+01	0.3871D−07	4
16	21	22	22	−0.42739165D+03	0.7595D−06	4
17	15	17	17	−0.37992109D−01	0.2986D−06	4
18	5	10	10	−0.24574119D−01	0.1933D−07	4
19	20	25	25	0.16549452D+04	0.3506D−06	4
20	78	130	130	−0.10001352D+01	0.1963D−06	4
21	27	31	31	0.24135487D+01	0.2021D−06	4
22	52	190	190	0.10000000D+01	0.4184D−06	4

Table 13: Results obtained by program TSEPS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1905	1907	1906	0.11113065D−16	0.5358D−07	3
2	196	202	197	0.10350094D−21	0.1407D−09	3
3	9	10	10	0.16141497D−07	0.8566D−05	4
4	23	28	24	0.13474977D+03	0.4995D−05	4
5	4	5	5	0.19859476D−14	0.9279D−07	4
6	5	6	6	0.49896319D−21	0.4526D−10	3
7	49	53	50	0.60734855D+05	0.5068D−05	4
8	44	54	45	0.17502221D−06	0.5161D−05	4
9	18	19	19	0.22198067D+04	0.6387D−05	4
10	41	49	42	0.19151134D+03	0.8357D−05	4
11	3764	3768	3765	0.96014265D−19	0.4404D−08	3
12	37	57	38	0.10030058D+05	0.1040D−04	1
13	19	22	20	0.13123402D+06	0.6887D−05	4
14	45	57	46	0.10851789D+03	0.3943D−05	4
15	15	18	16	0.18176315D+02	0.5262D−06	4
16	46	57	47	0.25110968D+01	0.7028D−05	4
17	10	12	11	0.28493252D−11	0.1007D−05	4
18	12	13	13	0.95556791D−19	0.5474D−08	3
19	14	15	15	0.11850699D−11	0.4660D−05	4
20	17	18	18	0.22432820D−12	0.1210D−07	4
21	34	37	35	0.64769614D+03	0.5217D−05	4
22	60	74	61	0.44869702D+04	0.6894D−05	4

Table 14: Results obtained by program TGAMU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1038	1042	1038	0.00000000D+00	0.0000D+00	3
2	247	247	247	0.19572970D+04	0.4746D−05	4
3	9	11	10	0.12549679D−07	0.6944D−05	4
4	19	23	20	0.13476134D+03	0.6597D−05	4
5	17	18	18	0.21443411D+01	0.1162D−05	4
6	6	7	7	0.79146071D−17	0.9336D−08	3
7	33	34	34	0.14581400D+06	0.0000D+00	4
8	61	76	62	0.59435758D−06	0.4930D−05	4
9	48	49	49	0.22201788D+04	0.4641D−06	4
10	33	40	34	0.19151134D+03	0.5164D−05	4
11	3867	2873	2870	0.00000000D+00	0.0000D+00	3
12	422	427	423	0.67887239D+05	0.1847D−06	4
13	21	22	22	0.14790600D+06	0.0000D+00	4
14	5	6	6	0.12669056D+03	0.0000D+00	4
16	12	16	13	0.35907414D+01	0.6644D−06	4
17	11	16	12	0.74191693D−10	0.5264D−05	4
18	0	1	1	0.00000000D+00	0.0000D+00	3
19	25	29	26	0.40187989D−12	0.2714D−05	4
20	963	964	964	0.49880012D+03	0.9445D−05	4
21	349	353	350	0.64960140D+03	0.6027D−05	4
22	63	75	64	0.44889615D+04	0.5376D−05	4

Table 15: Results obtained by program TGAMS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1377	1379	1379	0.69739198D−22	0.1298D−09	3
2	41	46	46	0.21657216D−16	0.1536D−06	3
3	10	11	13	0.21877074D−08	0.1864D−05	4
4	13	16	21	0.13474977D+03	0.2795D−06	4
5	4	5	7	0.11105836D−10	0.8867D−06	4
6	5	6	12	0.11841738D−11	0.3956D−05	4
7	10	12	23	0.60734855D+05	0.6483D−07	4
8	20	25	23	0.22988283D−07	0.5944D−05	4
9	14	15	31	0.22164587D+04	0.3465D−05	4
10	12	18	21	0.19151134D+03	0.5242D−07	4
11	2586	2592	2648	0.39784315D−12	0.8406D−05	4
12	13	17	19	0.22287907D+05	0.5451D−06	4
13	17	21	28	0.13123402D+06	0.7836D−08	4
14	5	8	18	0.10851789D+03	0.2265D−08	4
15	6	7	15	0.18176315D+02	0.2904D−06	4
16	15	21	40	0.25110968D+01	0.7243D−06	4
17	14	19	18	0.40185937D−10	0.4226D−05	4
18	42	44	45	0.15151799D−24	0.1223D−10	3
19	13	14	21	0.27214150D−11	0.7062D−05	4
20	26	27	29	0.37816152D−10	0.4067D−07	4
21	9	10	15	0.64782852D+03	0.2886D−05	4
22	26	32	45	0.44869702D+04	0.6023D−07	4

Table 16: Results obtained by program TGANU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1010	1012	1012	0.43700268D−13	0.2956D−05	4
2	259	272	506	0.19592865D+04	0.3450D−05	4
3	9	11	12	0.12549679D−07	0.6944D−05	4
4	11	15	17	0.13476134D+03	0.1613D−07	4
5	16	17	23	0.21443411D+01	0.9701D−06	4
6	6	7	13	0.79146068D−17	0.9336D−08	3
7	22	23	61	0.14581400D+06	0.0000D+00	4
8	23	30	26	0.87935249D−05	0.7357D−05	4
9	43	44	148	0.22201788D+04	0.9390D−05	4
10	12	19	21	0.19151134D+03	0.3013D−07	4
11	3977	2992	2990	0.00000000D+00	0.0000D+00	3
12	251	261	496	0.67887239D+05	0.2461D−05	4
13	19	20	36	0.14790600D+06	0.0000D+00	4
14	1	2	6	0.12669056D+03	0.0000D+00	4
16	46	50	135	0.35907414D+01	0.4692D−10	4
17	10	11	17	0.64961579D−03	0.1197D−05	4
18	0	1	3	0.00000000D+00	0.0000D+00	3
19	24	28	32	0.15533910D−11	0.5336D−05	4
20	904	905	2705	0.49880012D+03	0.9716D−05	4
21	236	273	403	0.64961789D+03	0.6713D−07	4
22	24	31	55	0.44889615D+04	0.2424D−07	4

Table 17: Results obtained by program TGANS

Problem	NIT	NFV	NFG	F	G	ITERM
1	10	41	0	0.22453071D−22	0.1682D−07	3
2	9	46	0	0.10689671D−22	0.1635D−06	3
3	3	19	0	0.33398898D−19	0.2231D−06	3
4	7	23	0	0.34819642D−17	0.1771D−02	3
5	13	65	0	0.67103619D−17	0.1419D−05	3
6	17	52	0	0.11091879D−16	0.1676D−11	3
7	15	46	0	0.12424559D−27	0.8730D−05	3
8	13	73	0	0.12574803D−25	0.1939D−04	3
9	27	190	0	0.76273114D−23	0.7442D−04	3
10	5	41	0	0.80384581D−25	0.4160D−03	3
11	20	61	0	0.35889147D−17	0.3364D−02	3
12	18	55	0	0.12927197D−16	0.7133D−13	3
13	20	41	0	0.57649443D−16	0.1215D−12	3
14	4	13	0	0.77478267D−20	0.4420D−05	3
15	5	36	0	0.18256673D−17	0.4713D−03	3
16	53	319	0	0.46216910D−17	0.1540D+00	3
17	14	48	0	0.44914028D−22	0.1055D−03	3
18	27	82	0	0.24970821D−20	0.5717D−01	3
19	2	7	0	0.30932438D−21	0.3701D−09	3
20	13	43	0	0.42827885D−20	0.2034D−07	3
21	12	37	0	0.20062314D−20	0.2554D−10	3
22	7	49	0	0.16272096D−17	0.2452D−05	3
23	29	262	0	0.39032690D−17	0.2007D−10	3
24	6	31	0	0.82252601D−23	0.8125D−09	3
25	9	46	0	0.14712666D−23	0.3954D−09	3
26	12	61	0	0.60883681D−17	0.4209D−07	3
27	10	51	0	0.27507809D−20	0.1218D−06	3
28	10	60	0	0.22953247D−16	0.2138D−05	3
29	4	53	0	0.12454916D−19	0.1307D−05	3
30	12	162	0	0.22295860D−21	0.1079D−07	3

Table 18: Results obtained by program TEQNU

Problem	NIT	NFV	NFG	C	G	ITERM
1	9	10	60	0.17763568D−14	0.2022D−11	4
2	12	13	182	0.42525442D−10	0.6211D−07	4
3	10	11	66	0.17259527D−11	0.8428D−08	4
4	16	19	102	0.14170665D−10	0.4058D−09	4
5	18	19	190	0.64257735D−08	0.3115D−06	4
6	18	19	266	0.12585488D−11	0.7113D−08	4
7	9	11	70	0.11830537D−11	0.2215D−11	4
8	38	51	273	0.44408921D−15	0.1118D−07	4
9	38	80	273	0.26595347D−08	0.8717D−08	4
10	13	14	84	0.37190317D−09	0.1666D−08	4
11	9	10	60	0.26132430D−11	0.2995D−11	4
12	7	8	56	0.14357404D−11	0.2885D−10	4
13	15	17	128	0.17763568D−14	0.6419D−06	4
14	12	13	91	0.22916137D−07	0.2745D−07	4
15	19	35	120	0.55067062D−13	0.1278D−11	4
16	8	9	45	0.67985217D−09	0.6081D−07	4
17	9	10	50	0.22204464D−15	0.1776D−14	4
18	10	13	55	0.36846441D−08	0.8094D−06	4

Table 19: Results obtained by program TINDU

Problem	NIT	NFV	NFG	F	G	ITERM
1	30	64	0	0.32607863D−18	0.1541D−03	3
2	17	57	0	0.72005783D−19	0.2616D−07	3
3	5	11	0	0.86121959D−16	0.3664D−03	3
4	11	19	0	0.11506040D−18	0.3589D−01	3
5	24	59	0	0.71834973D−16	0.5648D−06	3
6	22	31	0	0.16737688D−16	0.8986D−08	3
7	25	42	0	0.13700399D−20	0.1859D−05	3
8	21	60	0	0.49624281D−28	0.1838D−07	3
9	32	71	0	0.22087587D−21	0.8006D−05	3
10	9	24	0	0.20231594D−20	0.1630D−03	3
11	23	32	0	0.25126998D−18	0.4444D−02	3
12	23	40	0	0.86169047D−16	0.1905D−08	3
13	27	33	0	0.16083597D−16	0.3499D−07	3
14	8	13	0	0.59697408D−21	0.8116D−05	3
15	12	28	0	0.12490051D−17	0.3059D+00	3
16	22	78	0	0.98484038D−20	0.1254D−03	3
17	17	43	0	0.13023526D−20	0.1547D−04	3
18	46	61	0	0.22479321D−17	0.1164D−01	3
19	2	5	0	0.70440283D−18	0.2216D−06	3
20	18	30	0	0.15878735D−16	0.3125D−03	3
21	25	34	0	0.23392501D−16	0.1351D−05	3
22	14	45	0	0.18986181D−17	0.1288D−01	3
23	23	106	0	0.19474244D−18	0.5505D−08	3
24	20	53	0	0.73749958D−17	0.6112D−08	3
25	29	50	0	0.20879439D−17	0.4136D−08	3
26	36	67	0	0.13205478D−17	0.4810D−08	3
27	40	75	0	0.65935599D−17	0.8620D−08	3
28	27	83	0	0.46185557D−18	0.2687D−08	3
29	12	95	0	0.20696250D−16	0.7540D−08	3
30	18	145	0	0.74053256D−16	0.1680D−07	3

Table 20: Results obtained by program TEQLU

Problem	NIT	NFV	NFG	C	G	ITERM
1	6	7	42	0.53601568D−12	0.7637D−08	4
2	11	12	168	0.26914207D−09	0.5633D−07	4
3	11	12	72	0.31512570D−11	0.4344D−09	4
4	20	21	126	0.28507863D−10	0.5560D−08	4
5	15	16	160	0.62633035D−08	0.1996D−06	4
6	18	19	266	0.10901857D−09	0.3841D−06	4
7	11	13	84	0.65206507D−10	0.3585D−09	4
8	40	54	287	0.23603342D−12	0.5699D−06	4
9	31	48	224	0.22057214D−07	0.3808D−07	4
10	18	42	114	0.86457730D−09	0.2428D−08	4
11	7	8	48	0.83872231D−09	0.2638D−09	4
12	6	7	49	0.30306333D−07	0.5700D−06	4
13	15	25	128	0.27860979D−07	0.9773D−06	4
14	13	14	98	0.52994054D−10	0.3464D−10	4
15	19	22	120	0.12468239D−07	0.4291D−06	4
16	7	8	40	0.11065371D−10	0.1421D−10	4
17	8	9	45	0.12153709D−12	0.1669D−11	4
18	11	17	60	0.75315815D−07	0.7684D−06	4

Table 21: Results obtained by program TNULU

References

[1] Al-Baali M., Fletcher R.: Variational methods for nonlinear least squares. Journal of Optimization Theory and Applications 36 (1985) 405-421.

[2] Brown, P.N., and Saad, Y.: Convergencet theory of nonlinear Newton-Krylov algorithms. SIAM Journal on Optimization 4 (1994) 297-330.

[3] Byrd R.H., Nocedal J., Schnabel R.B.: Representation of quasi-Newton matrices and their use in limited memory methods. Math. Programming 63 (1994) 129-156.

[4] Coleman, T.F., Moré J.J.: Estimation of sparse Hessian matrices and graph coloring problems. Mathematical Programming 28 (1984) 243-270.

[5] Curtis, A.R., and Powell, M.J.D., and Reid, J.K.: On the estimation of sparse Jacobian matrices. IMA Journal of Aplied Mathematics 13 (1974) 117-119.

[6] Dembo, R.S, Eisenstat, S.C., and Steihaug T.: Inexact Newton Methods. SIAM J. on Numerical Analysis 19 (1982) 400-408.

[7] Dennis J.E., Mei H.H.W: An unconstrained optimization algorithm which uses function and gradient values. Report No. TR 75-246, 1975.

[8] Gill P.E., Murray W.: Newton type methods for unconstrained and linearly constrained optimization. Math. Programming 7 (1974) 311-350.

[9] 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.

[10] Gould N.I.M, Lucidi S., Roma M., Toint P.L.: Solving the trust-region subproblem using the Lanczos method. Report No. RAL-TR-97-028, 1997.

[11] Griewank A., Toint P.L.: Partitioned variable metric updates for large-scale structured optimization problems. Numer. Math. 39 (1982) 119-137.

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

[13] Lukšan L.: Combined trust region methods for nonlinear least squares. Kybernetika 32 (1996) 121-138.

[14] Lukšan L.: Hybrid methods for large sparse nonlinear least squares. J. Optimizaton Theory and Applications 89 (1996) 575-595.

[15] Lukšan L., Matonoha C., Vlček J.: A shifted Steihaug-Toint method for computing a trust-region step Report V-914, Prague, ICS AS CR, 2004.

[16] Lukšan L., Spedicato E.: Variable metric methods for unconstrained optimization and nonlinear least squares. Journal of Computational and Applied Mathematics 124 (2000) 61-93.

[17] Lukšan L., Vlček J. Sparse and partially separable test problems for unconstrained and equality constrained optimization. Report V-767, Prague, ICS AS CR, 1998.

[18] Lukšan L., Vlček J.: Computational Experience with Globally Convergent Descent Methods for Large Sparse Systems of Nonlinear Equations. Optimization Methods and Software 8 (1998) 201-223.

[19] 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.

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

[21] Martinez, J.M., and Zambaldi, M.C.: An Inverse Column-Updating Method for Solving Large-Scale Nonlinear Systems of Equations. Optimization Methods and Software 1 (1992) 129-140.

[22] Moré J.J., Sorensen D.C.: Computing a trust region step. SIAM Journal on Scientific and Statistical Computations 4 (1983) 553-572.

[23] Nocedal J.: Updating quasi-Newton matrices with limited storage. Math. Comp. 35 (1980) 773-782.

[24] Powell M.J.D: A new algorithm for unconstrained optimization. In: Nonlinear Programming (J.B.Rosen O.L.Mangasarian, K.Ritter, eds.) Academic Press, London 1970.

[25] Steihaug T.: The conjugate gradient method and trust regions in large-scale optimization. SIAM Journal on Numerical Analysis 20 (1983) 626-637.

[26] Toint P.L.: Towards an efficient sparsity exploiting Newton method for minimization. In: Sparse Matrices and Their Uses (I.S.Duff, ed.), Academic Press, London 1981, 57-88.

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

[28] Vlček J., Lukšan L.: New variable metric methods for unconstrained minimization covering the large-scale case. Report V-876, Prague, ICS AS CR, 2002.

[29] Vlček J., Lukšan L.: Additional properties of shifted variable metric methods. Report V-899, Prague, ICS AS CR, 2004.