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:

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. Subroutines PSEP and PSEN, intended for partially separable optimization problems, are based on partitioned variable metric updates (subroutine PSEN is able to solve nonsmooth problems). 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. Subroutine PMAX, intended for minimization of maximum value (minimax), is based on the primal line-search interior-point method. Subroutine PSUM, intended for minimization of sum of absolute values, is based on the primal trust-region interior-point 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.
The first author is also from Technical University of Liberec, Hálkova 6, 461 17 Liberec

1 Introduction

We propose fourteen basic subroutines which implement selected large-scale optimization algorithms. The double-precision FORTRAN 77 subroutines `PLIS`, `PLIP`, `PNET` are designed to find a close approximation to a local minimum of a general twice continuously differentiable function $F : R^n \to R$ with unknown or dense Hessian matrix. Subroutines `PLIS`, `PLIP` are based on limited-memory variable metric updates: `PLIS` uses Strang recurrences [12], [25] and `PLIP` uses shifted limited-memory variable metric updates [33]. Subroutine `PNET` is based on an inexact truncated Newton method.

The double-precision FORTRAN 77 subroutines `PNED`, `PNEC` are designed to find a close approximation to a local minimum of a general twice continuously differentiable function $F : R^n \to R$ with sparse Hessian matrix. These subroutines are based on inexact discrete Newton methods [6], [4]: `PNED` uses matrix direct methods (Dennis-Mei [7] or More-Sorensen [24]) and `PNEC` uses matrix iterative methods (Steihaug-Toint [28], [29] or shifted Steihaug-Toint [15]) for computing a trust region step.

The double-precision FORTRAN 77 subroutines `PSED`, `PSEC` are designed to find a close approximation to a local minimum of a smooth 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 $f_i : R^n \to R$, $1 \leq i \leq n_a$, are twice continuously differentiable functions. These subroutines are based on partitioned variable metric updates [11], [18]: `PSED` uses matrix direct method (Gill-Murray [8]) and `PSEC` uses matrix iterative method (Steihaug [28]) for computing a line-search step.

The double-precision FORTRAN 77 subroutine `PSEN` is designed to find a close approximation to a local minimum of a nonsmooth 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 $f_i : R^n \to R$, $1 \leq i \leq n_a$, are locally Lipschitz nondifferentiable functions. This subroutine is based on a bundle variable metric method described in [22].

The double-precision FORTRAN 77 subroutines `PGAD`, `PGAC` are designed to find a close approximation to a local minimum of the 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 $f_i : R^n \to R$, $1 \leq i \leq n_a$, are twice continuously differentiable functions. These subroutines are based on hybrid methods that combine the Gauss-Newton method with the Newton or the variable metric corrections [14], [18]: `PGAD` uses matrix direct methods (Dennis-Mei [7] or More-Sorensen [24]) and `PGAC` uses matrix iterative methods (Steihaug-Toint [28], [29] or shifted Steihaug-Toint [15]) for computing a trust region step.

The double-precision FORTRAN 77 subroutine `PMAX` is designed to find a close approximation to a local minimum of a minimax functions

$$F(x) = \max_{1 \leq i \leq n_a} f_i(x)$$

or

$$F(x) = \max_{1 \le i \le n_a} |f_i(x)|$$

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 are twice continuously differentiable functions. This subroutine is based on primal interior point methods described in [16].

The double-precision FORTRAN 77 subroutine `PSUM` is designed to find a close approximation to a local minimum of a sum of absolute values

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

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 are twice continuously differentiable functions. This subroutine is based on primal interior point methods described in [17].

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], [21]. Subroutine `PEQL` is based on the inverse column-update quasi-Newton method [23], [21].

Subroutines `PLIS`, `PLIP`, `PNET`, `PNED`, `PNEC`, `PSED`, `PSED`, `PGAD`, `PGAC` allow us to work with simple bounds. Simple bounds are assumed in the form

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

where $1 \le i \le n$ (I^x corresponds to array `IX` in Appendix A).

To simplify the user's work, additional easy-to-use subroutines are added. These subroutines, written in FORTRAN 90 (they use dynamic allocation of working arrays) call general subroutines `PLIS`, `PLIP`, `PNET`, `PNED`, `PNEC`, `PSED`, `PSEC`, `PSEN`, `PGAD`, `PGAC`, `PSUM`, `PMAX`, `PEQN`, `PEQL`:

- `PLISU,PLIPU,PNETU,PNEDU,PNECU,PSEDU,PSECU,PGADU,PGACU` – unconstrained smooth optimization.
- `PLISS,PLIPS,PNETS,PNEDS,PNECS,PSEDS,PSECS,PGADS,PGACS` – smooth optimization with simple bounds.
- `PSENU,PMAXU,PSUMU` – unconstrained nonsmooth optimization.
- `PEQNU,PEQLU` – solution of nonlinear equations.

Each subroutine contains a description of formal parameters and extensive comments. Moreover, text files PLIS.TXT, PLIP.TXT, PNET.TXT, PNED.TXT, PNEC.TXT, PSED.TXT, PSEC.TXT, PSEN.TXT, PGAD.TXT, PGAC.TXT, PMAX.TXT, PSUM.TXT, PEQN.TXT, PEQL.TXT, are added, which contain a detailed description of all important subroutines. Finally, test programs TLISU, TLISS, TLIPU, TLIPS, TNETU, TNETS, TNEDU, TNEDS, TNECU, TNECS, TSEDU, TSEDS, TSECU, TSECS, TSENU, TGADU, TGADS, TGACU, TGACS, TMAXU, TSUMU, TEQNU, TEQLU 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.

Subroutines presented in this report were generated using the UFO system [19]. We selected the most efficient methods from this system to prepare individual problem-oriented subroutines. This point of view gives a reason for our selection of optimization methods and trust-region strategies (system UFO contains many additional methods and strategies, which were also tested and compared). At the same time, this approach explains the fact, that some methods and strategies are implemented in slightly different ways than they are described in the original papers (we have carefully tuned them to obtain the best results). Subroutines PLIS, PNET, PNED, PNEC, PSED, PSEC, PGAD, PGAC, PEQN, PEQL implement classic optimization methods and have equivalents, e.g., L-BFGS [35], TNPACK [27], VE08 [30], VE10 [31], HOMPACK [34]. Numerical comparisons with these equivalents are given in Section 9. Subroutines PLIP, PSEN, PMAX, PSUM are based on quite new original methods proposed in [33], [22], [16], [17].

2 Matrix-free methods for general problems

Consider a general continuously differentiable function $F : R^n \to R$, where n is large, and assume that the structure of the Hessian matrix of F is not known. In this case, subroutines PLIS, PLIP, PNET based on matrix-free methods can be used for seeking a local minimum of F. Matrix-free 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 $0 < \alpha_k \leq \overline{\alpha}_k$ is a scalar step-size chosen in such a way that

$$F(x_{k+1}) - F_k \leq \varepsilon_1 \, \alpha_k \, d_k^T g_k, \quad d_k^T g(x_{k+1}) \geq \varepsilon_2 \, d_k^T g_k \tag{2}$$

(the weak Wolfe conditions) hold with $F_k = F(x_k)$, $g_k = \nabla F(x_k)$. Here $0 < \varepsilon_1 < 1/2$ is a tolerance for the function value decrease and $\varepsilon_1 < \varepsilon_2 < 1$ is a tolerance for the directional derivative increase (we use values $\varepsilon_1 = 0.0001$ and $\varepsilon_2 = 0.9$ 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 bisection (MES = 1), or by quadratic interpolation with two function values (MES = 2), or by quadratic interpolation with two directional derivatives (MES = 3), or by cubic interpolation (MES = 4). We start with the initial estimate $\alpha_k = 1$ if IEST = 0 or the initial estimate is derived by using the lower bound for F (parameter FMIN in our subroutines) if IEST = 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

2.1 Limited-memory BFGS method

Subroutine `PLIS` is an implementation of the limited-memory BFGS method proposed in [12], [25]. 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 $k - m \leq j \leq k - 1$. 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 [33]. 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 [33]) 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 the formula $\rho_k = \sqrt{\nu_k \varepsilon_k}$. 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.

2.3 Inexact truncated Newton method

2.4 An active set strategy for line-search methods

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 the detection of active constraints. Thus we set

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

for $1 \le i \le n$, 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

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

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

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

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 ∞).

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 [32] in subroutines `PNED`, `PNEC`. The sparsity pattern of the Hessian matrix (only the upper part) is stored in the coordinate form (if `ISPAS=1`) or in the standard compressed row format (if `ISPAS=2`) 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

$$\texttt{IH} = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 5 \end{bmatrix}, \quad \texttt{JH} = \begin{bmatrix} 1 & 2 & 3 & 5 & 2 & 4 & 3 & 5 & 4 & 5 \end{bmatrix}$$

if `ISPAS=1` or

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

if `ISPAS=2`. In the first case, nonzero elements in the upper part of the Hessian matrix can be sorted in an arbitrary order (not only by rows as in the above example) and arrays `IH` and `JH` have to be declared with lengths $n + m$, where m is the number of nonzero elements. In the second case, nonzero elements can be sorted only by rows. Components of `IH` contain addresses of the diagonal elements in this sequence and components of `JH` contain corresponding column indices (note that `IH` has $n + 1$ elements and the last element is equal to $m + 1$). Arrays `IH` and `JH` have to be declared with lengths $n + 1$ and m, respectively.

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`, `PGAD`, `PGAC`) 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

$$\begin{aligned} \|d_k\| &\leq \Delta_k, && (12)\\ \|d_k\| &< \Delta_k \quad \Rightarrow \quad \|\omega_k(d_k)\| \leq \overline{\omega}_k, && (13)\\ -Q_k(d_k) &\geq \underline{\sigma}\|g_k\| \min(\|d_k\|, \|g_k\|/\|B_k\|), && (14) \end{aligned}$$

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

$$\begin{aligned} \rho_k(d_k) \leq 0 \quad &\Rightarrow \quad \alpha_k = 0, && (15)\\ \rho_k(d_k) > 0 \quad &\Rightarrow \quad \alpha_k = 1. && (16) \end{aligned}$$

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

$$\begin{aligned} \rho_k(d_k) < \underline{\rho} \quad &\Rightarrow \quad \underline{\beta}\|d_k\| \leq \Delta_{k+1} \leq \overline{\beta}\|d_k\|, && (17)\\ \underline{\rho} \leq \rho_k(d_k) \leq \overline{\rho} \quad &\Rightarrow \quad \Delta_{k+1} = \Delta_k, && (18)\\ \overline{\rho} < \rho_k(d_k) \quad &\Rightarrow \quad \Delta_{k+1} = \min(\gamma\Delta_{k+1}, \overline{\Delta}), && (19) \end{aligned}$$

where $0 < \underline{\beta} \leq \overline{\beta} < 1 < \gamma$ 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 `FMIN` in our subroutines) if `INITS` $= 1$. If `INITS` $= 0$, parameter `FMIN` need not be defined.

Trust-region step d_k can be computed by four different strategies. Subroutines `PNED`, `PGAD` use the Moré-Sorensen (if `MOS=1`) or Dennis-Mei (if `MOS=2`) matrix decomposition methods, respectively. Subroutines `PNEC`, `PGAC` use the Steihaug-Toint (if `MOS=1`) or shifted Steihaug-Toint (if `MOS=2`) matrix iterative methods, respectively. To simplify the description of these methods, we omit the outer index k and denote the inner index by i.

3.1 Matrix decomposition 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

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

Necessary and sufficient conditions for this solution are

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

(we use symbol $\succeq$ for ordering by positive semidefiniteness). The Moré-Sorensen method [24] 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 $E_i \neq 0$, 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 $\underline{\delta}\Delta \leq \|d_i\| \leq \overline{\delta}\Delta$ 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 $\lambda_i \neq 0$, 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 $v_i^T d_i \geq 0$, which is a good approximation of the eigenvector of matrix B corresponding to its minimal eigenvalue, and compute number $\alpha_i \geq 0$ such that $\|d_i + \alpha_i v_i\| = \Delta$. If

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

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 Matrix decomposition Dennis-Mei 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 `PGAD` is based on the dog-leg method proposed in [26]. 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 Matrix iterative 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 [28] and Toint [29] 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

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

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 $\|Bp_1 + g\| \leq \overline{\omega}\|g\|$, 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`).

3.4 Matrix iterative shifted Steihaug-Toint trust region method

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 Partitioned 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

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

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 `PGAD`, `PGAC`, `PEQN`, `PEQL` described below.

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.

4.1 Partitioned variable metric methods

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

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

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

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

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`).

4.2 Partitioned Newton methods

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.

4.3 Partitioned variable metric methods for nonsmooth problems

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 $f_i(x)$, $1 \leq i \leq m$ (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 $f_i(x)$, $1 \leq i \leq m$). 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 $x_{k+1} \neq x_k$ (i.e., if (16) holds), we compute $F_{k+1} = F(x_{k+1})$, $J_{k+1} = J(x_{k+1})$ and set

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

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 Newton's 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 `PGAD` uses the Moré-Sorensen trust-region strategy described in Section 3.1.

5.2 Gauss-Newton method with sparse variable metric 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^TJ and the linear manifold of matrices satisfying quasi-Newton condition $Ws_k = y_k$ with $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$. Thus

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

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

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

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

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

5.3 Gauss-Newton method with partitioned variable metric corrections

6 Primal interior point methods for minimax optimization

7 Primal interior point methods for l_1 optimization

8 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

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

with the precision $0 \leq \overline{\omega}_k \leq \overline{\omega} < 1$ 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 $\underline{\beta}\alpha_k^j \leq \alpha_k^{j+1} \leq \overline{\beta}\alpha_k^j$ with $0 < \underline{\beta} \leq \overline{\beta} < 1$, satisfying

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

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 $A_k \neq J_k$, 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 $-d_k^T J_k^T f_k \leq \underline{\varepsilon}\|d_k\|\|J_k^T f_k\|$, 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 $\|r_i\| \leq \overline{\omega}\|f\|$, 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 [21].

8.1 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 $J(x)$.

8.2 Inverse column-update quasi-Newton method

Subroutine `PEQL` is an implementation of the inverse column update method, which is introduced in [23]. 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.

A Description of subroutines

In this section we describe easy-to-use subroutines `PLISU`, `PLISS`, `PLIPU`, `PLIPS`, `PNETU`, `PNETS`, `PNEDU`, `PNEDS`, `PNECU`, `PNECS`, `PSEDU`, `PSEDS`, `PSECU`, `PSECS`, `PSENU`, `PGADU`, `PGADS`, `PGACU`, `PGACS`, `PMAXU`, `PSUMU`, `PEQNU`, `PEQLU`, 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 form

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

whose elements have the following meanings:

Element	Type	Significance
NRES	IO	Number of restarts.
NDEC	IO	Number of matrix decompositions.
NIN	IO	Number of inner iterations (for solving linear systems).
NIT	IO	Number of iterations.
NFV	IO	Number of function evaluations.
NFG	IO	Number of gradient evaluations.
NFH	IO	Number of Hessian evaluations.

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

```
CALL PLISU(NF,X,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PLISS(NF,X,IX,XL,XU,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PLIPU(NF,X,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PLIPS(NF,X,IX,XL,XU,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PNETU(NF,X,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PNETS(NF,X,IX,XL,XU,IPAR,RPAR,F,GMAX,IPRNT,ITERM)
CALL PNEDU(NF,MH,X,IH,JH,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PNEDS(NF,MH,X,IX,XL,XU,IH,JH,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PNECU(NF,MH,X,IH,JH,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PNECS(NF,MH,X,IX,XL,XU,IH,JH,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PSEDU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PSEDS(NF,NA,MA,X,IX,XL,XU,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PSECU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PSECS(NF,NA,MA,X,IX,XL,XU,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PSENU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PGADU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PGADS(NF,NA,MA,X,IX,XL,XU,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PGACU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PGACS(NF,NA,MA,X,IX,XL,XU,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PMAXU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,IEXT,ISPAS,IPRNT,ITERM)
CALL PSUMU(NF,NA,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PEQNU(N,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
CALL PEQLU(N,MA,X,AF,IAG,JAG,IPAR,RPAR,F,GMAX,ISPAS,IPRNT,ITERM)
```

Their arguments have the following meanings:

Argument	Type	Significance

NF	II	Number of variables of the objective function.
NA	II	Number of partial functions.
N	II	Number of variables of the partially separable function.
MH	II	Number of nonzero elements in the upper part of the Hessian matrix. This parameter is used as input only if ISPAS = 1 (it defines dimensions of arrays IH and JH in this case).
MA	II	Number of nonzero elements in the Jacobian matrix. This parameter is used as input only if ISPAS = 1 (it defines dimensions of arrays IAG and JAG in this case).
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).
IH(NF+1)	IA	Pointers of the diagonal elements in the upper part of the Hessian matrix.
JH(MH)	IA	Column indices of the nonzero elements and additional working space for the Choleski decomposition of the Hessian matrix.
AF(NA)	RO	Vector which contains values of partial 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.
IPAR(8)	IA	Integer parameters (see Table 1).
RPAR(9)	RA	Real parameters (see Table 1).
F	RO	Value of the objective function at the solution X.
GMAX	RO	Maximum absolute value of a partial derivative of the Lagrangian function.
IEXT	II	The type of minimax: IEXT = 0: minimization of the maximum value, IEXT = 1: minimization of the maximum absolute value.
ISPAS	II	Sparse structure of the Hessian or Jacobian matrix: ISPAS = 1: the coordinate form is used, ISPAS = 2: the standard row compresed format is used.
IPRNT	II	Print specification: IPRNT = 0: print is suppressed, IPRNT = 1: basic print of final results, IPRNT = −1: extended print of final results, IPRNT = 2: basic print of intermediate and final results,

IPRNT $= -2$: extended print of intermediate and final results.

ITERM IO Variable that indicates the cause of termination:

- ITERM $= 1$: if $|X - X_{old}|$ was less than or equal to TOLX in two subsequent iterations,
- ITERM $= 2$: if $|F - F_{old}|$ 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 $= 6$: if termination criterion was not satisfied, but the solution is probably acceptable,
- ITERM $= 11$: if NIT exceeded MIT,
- ITERM $= 12$: if NFV exceeded MFV,
- ITERM $= 13$: if NFG exceeded MFG,
- ITERM < 0: if the method failed.

The sparsity pattern of the Hessian matrix (only the upper part) or the Jacobian matrix is stored in the coordinate form if ISPAS $= 1$ or in the standard compressed row format if ISPAS $= 2$ using arrays IH and JH or IAG and JAG. In the first case, nonzero elements can be sorted in an arbitrary order (not only by rows). Arrays IH and JH have to be declared with lengths NF + MH at least, while arrays IAG and JAG have to be declared with lengths NA + MA and MA at least, respectively. Here MH and MA are the numbers of nonzero elements. In the second case, nonzero elements can be sorted only by rows. Components of IH contain adresses of the diagonal elements in this sequence and components of JH contain corresponding column indices (note that IH has NF + 1 elements and the last element is equal to MH + 1). Arrays IH and JH have to be declared with lengths NF + 1 and MH at least, respectively. Similarly, components of IAG contain total numbers of nonzero elements in all previous rows increased by 1 and elements of JAG contain corresponding column indices (note that IAG has NA + 1 elements and the last element is equal to MA + 1). Arrays IAG and JAG have to be declared with length NA + 1 and MA at least, respectively.

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

Argument	Type	Significance
MIT	II	Maximum number of iterations; the choice MIT $= 0$ causes that the default value (see Table 2) will be taken.
MFV	II	Maximum number of function evaluations; the choice MFV $= 0$ causes that the default value (see Table 2) will be taken.
MFG	II	Maximum number of gradient evaluations; the choice MFG $= 0$ causes that the default value (see Table 2) 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.
MEC	II	Variable that determines the method of a second order correction: MEC $= 1$: correction by the Marwil sparse variable metric update, MEC $= 2$: correction by differences of gradients (discrete Newton correction), MEC $= 3$: correction by the Griewank-Toint partitioned variable metric update (symmetric rank-one). The choice MEC $= 0$ causes that the default value MEC $= 2$ will be taken.

Parameter	PLIS	PLIP	PNET	PNED	PNEC	PSED	PSEC	PSEN	PGAD	PGAC	PMAX	PSUM	PEQN	PEQL
IPAR(1)	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT	MIT
IPAR(2)	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV	MFV
IPAR(3)	-	-	MFG	MFG	MFG	MFG	MFG	-	MFG	MFG	MFG	MFG	-	-
IPAR(4)	IEST	IEST	IEST	IEST	IEST	IEST	IEST	IEST	MEC	MEC	IEST	IEST	-	-
IPAR(5)	-	MET	MOS1	MOS	MOS1	MET	MET	-	MOS	MOS1	MED	MED	MOS1	MOS1
IPAR(6)	-	-	MOS2	-	MOS2	-	MOS2	MB	-	MOS2	-	-	MOS2	MOS2
IPAR(7)	MF	MF	MF	IFIL	IFIL	IFIL	IFIL	IFIL	IFIL	IFIL	IFIL	IFIL	-	MF
RPAR(1)	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX	XMAX
RPAR(2)	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX	TOLX
RPAR(3)	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF	TOLF
RPAR(4)	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB	TOLB
RPAR(5)	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG	TOLG
RPAR(6)	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	FMIN	-	-
RPAR(7)	-	-	-	XDEL	XDEL	-	-	-	XDEL	XDEL	-	XDEL	-	-
RPAR(8)	-	-	-	-	-	-	-	ETA3	ETA	ETA	ETA4	-	ETA2	ETA2
RPAR(9)	-	-	-	-	-	-	-	ETA5	-	-	ETA5	ETA5	-	-

Table 1: Integer and real parameters

`MED` `II` Variable that specifies the method used:

`MED` = 1: partitioned variable metric method,
`MED` = 2: safeguarded discrete Newton method.

The choice `MED` = 0 causes that the default value `MED` = 1 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 `PSED,PSEC`: Variable that specifies the variable metric update:

`MET` = 1: safeguarded BFGS method,
`MET` = 2: combination of the BFGS and the symmetric rank-one method,
`MET` = 3: discrete Newton method.

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

`MOS` `II` Method for computing the trust-region step:

`MOS` = 1: double dog-leg method of Dennis and Mei,
`MOS` = 2: method of More and Sorensen for obtaining optimum locally constrained step.

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

`MOS1` `II` In `PNET`: Choice of restarts after constraint change:

`MOS1` = 1: restarts are suppressed,
`MOS1` = 2: restarts with steepest descent directions are used.

The choice `MOS1` = 0 causes that the default value `MOS1` = 1 will be taken.

In PNEC: Number of Lanczos steps for determination of the Levenberg-Marquardt parameter (recommended value is MOS1 = 5).
In PGAC: Method for computing trust-region step:

MOS1 = 1: Steihaug-Toint conjugate gradient method,
MOS1 = 2: shifted Steihaug-Toint method with five Lanczos steps,
MOS1 > 2: shifted Steihaug-Toint method with MOS1 Lanczos steps.

The choice MOS1 = 0 causes that the default value MOS1 = 2 will be taken.

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

MOS1 = 1: smoothing is not used,
MOS1 = 2: single smoothing strategy is used,
MOS1 = 3: double smoothing strategy is used.

The choice MOS1 = 0 causes that the default value MOS1 = 3 will be taken.

MOS2 II Choice of preconditioning strategy:

MOS2 = 1: preconditioning is not used,
MOS2 = 2: preconditioning by the incomplete Gill-Murray (LU in PEQN,PEQL) decomposition,
MOS2 = 3: preconditioning by the incomplete Gill-Murray (LU in PEQN,PEQL) decomposition combined with preliminary solution of the preconditioned system.

The choice MOS2 = 0 causes that the default value MOS2 = 2 (MOS2 = 1 in PNET and MOS2 = 3 in PEQN,PEQL) will be taken.

MB II Dimension of a bundle used in the line search.

MF II The number of limited-memory variable metric updates in each iteration.

IFIL II Variable that specifies a relative size of the space reserved for fill-in. The choice IFIL = 0 causes that the default value IFIL = 1 will be taken.

XMAX RI Maximum stepsize; the choice XMAX = 0 causes that the default value (see Table 2) will be taken.

TOLX RI Tolerance for the change of the coordinate vector X; the choice TOLX = 0 causes that the default value (see Table 2) will be taken.

TOLF RI Tolerance for the change of function values; the choice TOLF = 0 causes that the default value (see Table 2) will be taken.

TOLB RI Minimum acceptable function value; the choice TOLB = 0 causes that the default value (see Table 2) will be taken.

TOLG RI Tolerance for the Lagrangian function gradient; the choice TOLG = 0 causes that the default value (see Table 2) will be taken.

FMIN RI Lower bound for the minimum function value.

XDEL RI Trust region step-size; the choice XDEL = 0 causes that a suitable default value will be computed.

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.

ETA3 RI Correction parameter; the choice ETA3 = 0 causes that the default value ETA3 = 10^{-12} will be taken.

ETA4	RI	Coefficient for the barrier parameter decrease; the choice ETA4 = 0 causes that the default value ETA4 = 0.85 will be taken.
ETA5	RI	In PSEN: Parameter for subgradient locality measure; the choice ETA5 = 0 causes that the default value ETA5 = 10^{-12} will be taken. In PMAX,PSUM: Minimum permitted value of the barrier parameter; the choice ETA5 = 0 causes that the default value ETA5 = 10^{-10} in PMAX and ETA5 = 10^{-8} in PSUM will be taken.

Value	PLIS PLIP	PNET	PNED PNEC	PSED PSEC	PSEN	PGAD PGAC	PMAX	PSUM	PEQL PEQN
MIT	9000	5000	5000	9000	20000	5000	10000	10000	1000
MFV	9000	5000	5000	9000	20000	5000	10000	10000	1000
MFG	9000	30000	10000	9000	20000	10000	20000	20000	10000
XMAX	10^{16}	10^{16}	10^{16}	10^{16}	10^{16}	10^{16}	10^{16}	10^{16}	10^{16}
TOLX	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}
TOLF	10^{-14}	10^{-14}	10^{-14}	10^{-14}	10^{-12}	10^{-14}	10^{-14}	10^{-12}	10^{-16}
TOLB	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-12}	10^{-16}	10^{-16}	10^{-12}	10^{-16}
TOLG	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-8}	10^{-6}	10^{-6}	10^{-6}	10^{-6}

Table 2: Default values (TOLB = TOLB + FMIN)

The subroutines PLISU,PLISS,PLIPU,PLIPS,PNETU,PNETS,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 PSEDU,PSEDS,PSECU,PSECS,PSENU,PGADU,PGADS,PGACU,PGACS,PMAXU,PSUMU 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(N,KA,X,FA)
```

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

Argument	Type	Significance
NF,N	II	Number of variables of the objective function.
KA	II	Index of the partial function.
X(NF),X(N)	RI	An estimate to the solution.
F	RO	Value of the objective function at the point X.

FA	RO	Value of the KA-th partial 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 partial function at the point X. In PSEN: An arbitrary subgradient of the KA-th partial function at the point X.

B Verification of subroutines

In this section we report the results obtained by using test programs TLISU, TLISS, TLIPU, TLIPS, TNETU, TNETS, TNEDU, TNEDS, TNECU, TNECS, TSEDU, TSEDS, TSECU, TSECS, TSENU, TGADU, TGADS, TGACU, TGACS, TMAXU, TSUMU, TEQNU, TEQLU, which serve for demonstration, verification and testing of subroutines PLISU, PLISS, PLIPU, PLIPS, PNETU, PNETS, PNEDU, PNEDS, PNECU, PNECS, PSEDU, PSEDS, PSECU, PSECS, PSENU, PGADU, PGADS, PGACU, PGACS, PMAXU, PSUMU, PEQNU, PEQLU, 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 G, and the cause of termination ITERM). The last row of every table contains the total number of NIT, NFV, NFG, and, moreover, for programs TNECU, TNECS, TSECU, TSECS, TGACU, TGACS also the total number of conjugate gradient iterations NCG. The total computational time in seconds is included. 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	0.963780013E−14	0.891E−06	4
2	425	454	454	14.9944763	0.773E−05	2
3	74	78	78	0.655101686E−09	0.539E−06	4
4	103	112	112	269.499543	0.899E−06	4
5	24	26	26	0.130639280E−11	0.671E−06	4
6	30	31	31	0.216102227E−10	0.946E−06	4
7	38	43	43	335.137433	0.730E−06	4
8	29	33	33	761774.954	0.432E−03	2
9	13	16	16	316.436141	0.369E−06	4
10	1540	1582	1582	−124.630000	0.124E−04	2
11	114	138	138	10.7765879	0.380E−06	4
12	248	267	267	982.273617	0.123E−04	2
13	7	8	8	0.165734137E−12	0.453E−06	4
14	10	12	12	0.128729169E−08	0.916E−06	4
15	2830	2929	2929	1.92401599	0.936E−06	4
16	196	210	210	−427.404476	0.991E−05	2
17	1007	1032	1032	−0.379921091E−01	0.876E−06	4
18	1449	1474	1474	−0.245741193E−01	0.862E−06	4
19	1393	1431	1431	59.5986241	0.259E−05	2
20	2129	2191	2191	−1.00013520	0.908E−06	4
21	2120	2169	2169	2.13866377	0.927E−06	4
22	1305	1346	1346	1.00000000	0.982E−06	4
Σ	20072	21136	21136		TIME = 8.90	

Table 3: Results obtained by program TLISU

Problem	NIT	NFV	NFG	F	G	ITERM
1	5055	5595	5595	0.00000000	0.000E+00	3
2	2016	2289	2289	3926.45961	0.304E−04	2
3	95	106	106	0.217616100E-12	0.780E−06	4
4	58	64	64	269.522686	0.124E−05	2
5	24	26	26	0.130639280E-11	0.671E−06	4
6	30	31	31	0.216102227E-10	0.946E−06	4
7	31	35	35	337.722479	0.776E−06	4
8	50	58	58	761925.725	0.257E−03	2
9	504	506	506	428.056916	0.940E−07	4
10	1152	1211	1211	−82.0207503	0.176E−04	2
11	13	23	23	96517.2947	0.126E−08	4
12	79	88	88	4994.21410	0.325E−06	4
13	7	8	8	0.165734137E-12	0.453E−06	4
14	10	12	12	0.128729169E-08	0.916E−06	4
15	2830	2929	2929	1.92401599	0.936E−06	4
16	176	184	184	−427.391653	0.348E−04	2
17	1007	1032	1032	−0.379921091E-01	0.876E−06	4
18	1449	1474	1474	−0.245741193E-01	0.862E−06	4
19	1150	1183	1183	1654.94525	0.908E−05	2
20	2211	2274	2274	−1.00013520	0.886E−06	4
21	1280	1303	1303	2.41354873	0.997E−06	4
22	1562	1598	1598	1.00000000	0.786E−06	4
Σ	20789	22029	22029		TIME = 11.37	

Table 4: Results obtained by program TLISS

Problem	NIT	NFV	NFG	F	G	ITERM
1	5383	5417	5417	0.601022658E−13	0.599E−06	4
2	530	557	557	3.57276719	0.124E−05	2
3	125	128	128	0.338270284E−12	0.518E−06	4
4	109	114	114	269.499543	0.669E−06	4
5	26	27	27	0.710072396E−11	0.951E−06	4
6	35	36	36	0.142942272E−10	0.737E−06	4
7	36	41	41	336.937181	0.956E−06	4
8	33	36	36	761774.954	0.192E−02	2
9	15	18	18	316.436141	0.264E−06	4
10	2003	2030	2030	−124.950000	0.116E−04	2
11	157	175	175	10.7765879	0.299E−06	4
12	337	350	350	982.273617	0.145E−04	2
13	9	10	10	0.230414406E−14	0.642E−07	4
14	8	10	10	0.128834241E−08	0.977E−06	4
15	1226	1256	1256	1.92401599	0.970E−06	4
16	237	246	246	−427.404476	0.501E−04	2
17	598	604	604	−0.379921091E−01	0.908E−06	4
18	989	998	998	−0.245741193E−01	0.975E−06	4
19	1261	1272	1272	59.5986241	0.410E−05	2
20	2045	2058	2058	−1.00013520	0.911E−06	4
21	2175	2196	2196	2.13866377	0.996E−06	4
22	1261	1292	1292	1.00000000	0.927E−06	4
Σ	18598	18871	18871		TIME = 8.82	

Table 5: Results obtained by program TLIPU

Problem	NIT	NFV	NFG	F	G	ITERM
1	5263	5321	5321	0.530131995E−13	0.370E−05	2
2	2293	2447	2447	3930.43962	0.251E−04	2
3	127	132	132	0.210550150E−12	0.437E−06	4
4	70	72	72	269.522686	0.794E−06	4
5	26	27	27	0.710072396E−11	0.951E−06	4
6	35	36	36	0.142942272E−10	0.737E−06	4
7	37	43	43	336.937181	0.133E−05	2
8	59	65	65	761925.725	0.399E−03	2
9	508	510	510	428.056916	0.776E−06	4
10	1253	1277	1277	−82.5400568	0.120E−04	2
11	13	19	19	96517.2947	0.150E−04	2
12	95	102	102	4994.21410	0.790E−04	2
13	9	10	10	0.230414406E−14	0.642E−07	4
14	8	10	10	0.128834241E−08	0.977E−06	4
15	1226	1256	1256	1.92401599	0.970E−06	4
16	227	228	228	−427.391653	0.952E−05	2
17	598	604	604	−0.379921091E−01	0.908E−06	4
18	989	998	998	−0.245741193E−01	0.975E−06	4
19	1367	1383	1383	1654.94525	0.105E−04	2
20	2274	2303	2303	−1.00013520	0.798E−06	4
21	1196	1211	1211	2.41354873	0.975E−06	4
22	1361	1381	1381	1.00000000	0.962E−06	4
Σ	19034	19435	19435		TIME = 9.42	

Table 6: Results obtained by program TLIPS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1481	1656	26037	0.117631766E−15	0.354E−06	4
2	132	387	7945	0.153382199E−15	0.988E−08	4
3	19	20	110	0.421204156E−09	0.353E−06	4
4	19	20	230	269.499543	0.779E−07	4
5	12	13	49	0.465606821E−11	0.364E−06	4
6	13	14	76	0.366783327E−11	0.404E−06	4
7	9	10	37	336.937181	0.248E−06	4
8	11	12	58	761774.954	0.155E−07	4
9	7	11	28	316.436141	0.158E−07	4
10	75	153	3213	−133.610000	0.777E−08	4
11	33	45	181	10.7765879	0.414E−07	4
12	23	30	457	982.273617	0.591E−08	4
13	7	8	16	0.533593908E−15	0.327E−07	4
14	1	2	1005	0.120245125E−08	0.879E−07	4
15	14	15	4033	1.92401599	0.468E−07	4
16	13	17	295	−427.404476	0.800E−08	4
17	4	5	810	−0.379921091E−01	0.537E−06	4
18	4	5	1146	−0.245741193E−01	0.425E−06	4
19	10	11	1986	59.5986241	0.423E−06	4
20	18	39	3051	−1.00013520	0.712E−07	4
21	7	8	4901	2.13866377	0.120E−08	4
22	55	145	4760	1.00000000	0.206E−08	4
Σ	1967	2626	60424		TIME = 6.95	

Table 7: Results obtained by program TNETU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1611	1793	28524	0.00000000	0.000E+00	3
2	259	259	4418	3930.43956	0.230E−07	4
3	17	18	98	0.158634811E−08	0.954E−06	4
4	12	13	105	269.522686	0.103E−07	4
5	12	13	49	0.465606821E−11	0.364E−06	4
6	13	14	76	0.366783327E−11	0.404E−06	4
7	9	10	37	336.937181	0.248E−06	4
8	40	41	248	761925.725	0.281E−06	4
9	553	555	2056	428.056916	0.850E−07	4
10	112	137	2109	−84.1426617	0.732E−06	4
11	7	8	17	96517.2947	0.112E−11	4
12	133	136	2689	4994.21410	0.180E−06	4
13	7	8	16	0.533593908E−15	0.327E−07	4
14	1	2	1005	0.120245125E−08	0.879E−07	4
15	14	15	4033	1.92401599	0.468E−07	4
16	12	13	294	−427.391653	0.594E−06	4
17	4	5	810	−0.379921091E−01	0.537E−06	4
18	4	5	1146	−0.245741193E−01	0.425E−06	4
19	8	9	1902	1654.94525	0.690E−07	4
20	16	25	3254	−1.00013520	0.836E−08	4
21	4	5	1211	2.41354873	0.135E−06	4
22	52	137	4843	1.00000000	0.657E−06	4
Σ	2900	3221	58940		TIME = 8.56	

Table 8: Results obtained by program TNETS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1421	1425	5688	0.465831486E−25	0.418E−12	3
2	39	45	200	0.231406390E−14	0.350E−06	4
3	17	18	108	0.839782900E−09	0.933E−06	4
4	24	25	100	269.499543	0.666E−10	4
5	11	12	72	0.795109456E−10	0.473E−06	4
6	13	16	196	0.125944855E−10	0.815E−06	4
7	12	13	78	336.937181	0.300E−06	4
8	4	5	90	761774.954	0.216E−06	4
9	7	9	16	316.436141	0.146E−06	4
10	69	75	630	−135.290000	0.291E−11	4
11	67	68	408	10.7765879	0.199E−06	4
12	127	128	512	982.273617	0.495E−09	4
13	6	7	28	0.598998674E−10	0.693E−06	4
14	2	3	18	0.129013604E−08	0.792E−06	4
15	9	10	40	1.92401599	0.414E−06	4
16	7	8	48	−427.404476	0.565E−07	4
17	8	9	54	−0.379921091E−01	0.314E−10	4
18	7	8	48	−0.245741193E−01	0.218E−09	4
19	6	7	42	59.5986241	0.952E−08	4
20	14	15	90	−1.00013520	0.139E−08	4
21	11	12	72	2.13866377	0.331E−08	4
22	30	34	186	1.00000000	0.164E−08	4
Σ	1911	1952	8724		TIME = 3.00	

Table 9: Results obtained by program TNEDU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1420	1424	5680	0.00000000	0.000E+00	3
2	128	130	640	1980.05047	0.911E−10	4
3	17	19	108	0.189355864E−09	0.340E−06	4
4	10	12	44	269.522686	0.328E−09	4
5	13	15	84	0.391905635E−12	0.536E−06	4
6	13	14	196	0.136396633E−11	0.901E−06	4
7	30	32	186	336.920046	0.151E−05	2
8	37	38	684	761925.725	0.119E−06	4
9	507	508	1016	428.056916	0.347E−13	4
10	109	127	990	−80.4518214	0.639E−06	4
11	6	8	42	72291.4951	0.178E−08	4
12	519	520	2080	4994.21410	0.236E−06	4
13	3	4	16	0.660542076E−23	0.363E−11	3
14	2	3	18	0.129013604E−08	0.792E−06	4
15	9	10	40	1.92401599	0.414E−06	4
16	15	18	96	−427.391653	0.342E−06	4
17	8	9	54	−0.379921091E−01	0.314E−10	4
18	7	8	48	−0.245741193E−01	0.218E−09	4
19	13	16	84	1654.94525	0.174E−08	4
20	14	15	90	−1.00013520	0.139E−08	4
21	9	10	60	2.41354873	0.388E−08	4
22	30	34	186	1.00000000	0.164E−08	4
Σ	2919	2974	12442		TIME = 6.56	

Table 10: Results obtained by program TNEDS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1447	1450	5792	0.173249493E−16	0.138E−06	3
2	79	89	400	0.169144088E−20	0.382E−09	3
3	18	19	114	0.180692317E−09	0.316E−06	4
4	24	25	100	269.499543	0.136E−08	4
5	11	12	72	0.990922474E−10	0.511E−06	4
6	17	21	252	0.166904871E−10	0.898E−06	4
7	11	12	72	336.937181	0.629E−06	4
8	6	11	126	761774.954	0.237E−05	2
9	7	8	16	316.436141	0.362E−08	4
10	70	74	639	−133.630000	0.221E−07	4
11	71	72	432	10.7765879	0.237E−10	4
12	133	134	536	982.273617	0.203E−07	4
13	7	8	32	0.402530175E−26	0.153E−13	3
14	2	3	18	0.129028794E−08	0.820E−06	4
15	10	11	44	1.92401599	0.217E−06	4
16	12	15	78	−427.404476	0.894E−09	4
17	8	9	54	−0.379921091E−01	0.391E−09	4
18	8	9	54	−0.245741193E−01	0.705E−10	4
19	7	8	48	59.5986241	0.106E−08	4
20	10	11	66	−1.00013520	0.277E−11	4
21	11	12	72	2.13866377	0.154E−06	4
22	46	51	282	1.00000000	0.376E−08	4
Σ	2015	2064	9299	NCG = 1182	TIME = 2.92	

Table 11: Results obtained by program TNECU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1436	1439	5748	3.98662385	0.138E−08	4
2	79	89	400	0.169144088E−20	0.382E−09	3
3	18	19	114	0.180692317E−09	0.316E−06	4
4	24	25	100	269.499543	0.136E−08	4
5	11	12	72	0.990922474E−10	0.511E−06	4
6	17	21	252	0.166904871E−10	0.898E−06	4
7	11	12	72	336.937181	0.629E−06	4
8	6	11	126	761774.954	0.237E−05	2
9	7	8	16	316.436141	0.362E−08	4
10	70	74	639	−133.630000	0.221E−07	4
11	27	31	168	86.8673060	0.416E−06	4
12	133	134	536	982.273617	0.203E−07	4
13	7	8	32	0.402530175E−26	0.153E−13	3
14	2	3	18	0.129028794E−08	0.820E−06	4
15	10	11	44	1.92401599	0.217E−06	4
16	12	15	78	−427.404476	0.894E−09	4
17	8	9	54	−0.379921091E−01	0.391E−09	4
18	8	9	54	−0.245741193E−01	0.705E−10	4
19	7	8	48	59.5986241	0.106E−08	4
20	10	11	66	−1.00013520	0.277E−11	4
21	11	12	72	2.13866377	0.154E−06	4
22	46	51	282	1.00000000	0.376E−08	4
Σ	1960	2012	8991	NCG = 1127	TIME = 2.88	

Table 12: Results obtained by program TNECS

Problem	NIT	NFV	NFG	F	G	ITERM
1	2654	3627	3627	0.794789730E−16	0.213E−06	3
2	105	179	179	83.3161404	0.498E−06	4
3	40	45	45	0.267007684E−12	0.823E−06	4
4	37	45	45	269.499543	0.605E−06	4
5	16	17	17	0.106026711E−11	0.728E−06	4
6	38	40	40	0.546961387E−11	0.882E−06	4
7	22	26	26	335.252624	0.105E−06	4
8	26	40	40	761774.954	0.295E−04	2
9	193	202	202	316.436141	0.155E−05	2
10	227	258	258	−125.810000	0.351E−04	2
11	100	127	127	10.7765879	0.566E−06	4
12	28	29	29	982.273617	0.102E−06	4
13	1	2	2	0.00000000	0.000E+00	3
14	25	28	28	0.104289352E−08	0.927E−06	4
15	8	15	15	1.92401599	0.482E−07	4
16	25	35	35	−427.404476	0.130E−06	4
17	15	17	17	−0.379921091E−01	0.141E−06	4
18	5	11	11	−0.245741193E−01	0.311E−07	4
19	19	23	23	59.5986241	0.466E−06	4
20	37	97	97	−1.00013520	0.212E−08	4
21	37	40	40	2.13866377	0.767E−06	4
22	55	211	211	1.00000000	0.610E−07	4
Σ	3713	5114	5114		TIME = 4.27	

Table 13: Results obtained by program TSEDU

Problem	NIT	NFV	NFG	F	G	ITERM
1	2591	3322	3322	0.00000000	0.000E+00	3
2	344	347	347	35.1211309	0.107E−06	4
3	39	43	43	0.441691821E−12	0.425E−06	4
4	21	22	22	269.522686	0.105E−06	4
5	16	17	17	0.783032535E−11	0.279E−06	4
6	32	33	33	0.959526458E−11	0.801E−06	4
7	19	21	21	337.722479	0.247E−06	4
8	52	56	56	761925.725	0.780E−04	2
9	1001	1003	1003	428.056916	0.192E−06	4
10	191	222	222	−86.7038382	0.225E−05	2
11	13	18	18	72291.4951	0.285E−08	4
12	228	235	235	4994.21410	0.304E−06	4
13	1	2	2	0.00000000	0.000E+00	3
14	25	28	28	0.104289352E−08	0.927E−06	4
15	8	15	15	1.92401599	0.534E−07	4
16	21	22	22	−427.391653	0.759E−06	4
17	15	17	17	−0.379921091E−01	0.299E−06	4
18	5	10	10	−0.245741193E−01	0.193E−07	4
19	20	25	25	1654.94525	0.351E−06	4
20	78	130	130	−1.00013520	0.196E−06	4
21	27	31	31	2.41354873	0.202E−06	4
22	52	190	190	1.00000000	0.418E−06	4
Σ	4799	5809	5809		TIME = 7.68	

Table 14: Results obtained by program TSEDS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1469	1640	4580	0.190717059E−15	0.188E−07	4
2	137	406	958	0.733479616E−22	0.408E−10	3
3	16	17	85	0.302968461E−09	0.400E−06	4
4	13	14	70	269.499543	0.697E−08	4
5	13	14	42	0.705564252E−12	0.599E−06	4
6	13	14	98	0.136525612E−11	0.901E−06	4
7	12	17	43	336.937181	0.260E−09	4
8	5	8	38	761774.954	0.127E−06	4
9	5	9	39	316.436141	0.996E−12	4
10	60	106	411	−124.690000	0.102E−08	4
11	30	38	193	10.7765879	0.419E−06	4
12	24	25	75	982.273617	0.161E−09	4
13	3	4	12	0.660547868E−23	0.363E−11	3
14	2	4	10	0.787241903E−12	0.492E−09	4
15	4	6	16	1.92401599	0.864E−06	4
16	9	19	39	−427.404476	0.114E−12	4
17	3	4	12	−0.379921091E−01	0.158E−07	4
18	2	4	10	−0.245741193E−01	0.482E−09	4
19	2	5	11	59.5986241	0.316E−07	4
20	15	33	65	−1.00013520	0.408E−09	4
21	7	8	24	2.13866377	0.909E−06	4
22	44	107	197	1.00000000	0.435E−07	4
Σ	1888	2502	7028	NCG = 9154	TIME = 2.95	

Table 15: Results obtained by program TSECU

Problem	NIT	NFV	NFG	F	G	ITERM
1	2598	3347	3347	0.00000000	0.000E+00	3
2	352	361	361	35.1211309	0.853E−05	2
3	39	43	43	0.441691822E−12	0.425E−06	4
4	21	22	22	269.522686	0.105E−06	4
5	16	17	17	0.783032535E−11	0.279E−06	4
6	32	33	33	0.959526458E−11	0.801E−06	4
7	19	21	21	337.722479	0.162E−05	2
8	46	49	49	761925.725	0.792E−04	2
9	1001	1003	1003	428.056916	0.348E−08	4
10	203	233	233	−86.7188428	0.288E−04	2
11	21	38	38	72291.4951	0.135E−10	4
12	223	230	230	4994.21410	0.303E−06	4
13	1	2	2	0.00000000	0.000E+00	3
14	25	28	28	0.104289348E−08	0.927E−06	4
15	17	27	27	1.92401599	0.553E−07	4
16	21	22	22	−427.391653	0.759E−06	4
17	15	17	17	−0.379921091E−01	0.299E−06	4
18	8	12	12	−0.245741193E−01	0.358E−11	4
19	20	25	25	1654.94525	0.351E−06	4
20	33	46	46	−1.00013520	0.959E−10	4
21	27	31	31	2.41354873	0.202E−06	4
22	51	185	185	1.00000000	0.834E−06	4
Σ	4789	5792	5792	NCG = 15187	TIME = 6.32	

Table 16: Results obtained by program TSECS

Problem	NIT	NFV	NFG	F	G	ITERM
1	3124	3134	3134	0.287703261E−08	0.582E−08	4
2	286	287	287	0.379499216E−08	0.203E−06	2
3	71	71	71	0.233196848E−09	0.100E−07	4
4	40	40	40	126.863549	0.699E−08	4
5	282	282	282	0.732927514E−07	0.400E−08	4
6	344	344	344	0.836329152E−08	0.326E−08	4
7	286	287	287	2391.16999	0.673E−04	2
8	610	611	611	0.317244739E−05	0.548E−08	4
9	2514	2516	2516	552.380551	0.448E−08	4
10	907	907	907	131.888476	0.579E−08	4
11	269	271	271	0.173668302E−09	0.266E−08	4
12	1805	1810	1810	621.128947	0.906E−02	2
13	680	681	681	2940.50943	0.140E−03	2
14	370	370	370	112.314954	0.622E−08	4
15	364	364	364	36.0935676	0.986E−08	4
16	1004	1004	1004	13.2000000	0.904E−08	4
17	380	380	380	0.268534232E−01	0.871E−09	4
18	15319	15321	15321	0.589970806E−08	0.925E−08	4
19	3972	4056	4056	0.565862690E−08	0.887E−08	4
20	774	988	988	0.406495193E−08	0.468E−08	4
21	247	248	248	264.000000	0.364E−03	2
22	1191	1192	1192	593.360762	0.145E−03	2
Σ	34839	35164	35164		TIME = 13.49	

Table 17: Results obtained by program TSENU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1377	1379	1379	0.697391982E−22	0.130E−09	3
2	41	46	46	0.216572157E−16	0.154E−06	3
3	11	12	14	0.136731713E−09	0.233E−06	4
4	13	16	21	134.749772	0.279E−06	4
5	4	5	7	0.111058357E−10	0.887E−06	4
6	6	7	13	0.742148235E−26	0.303E−12	3
7	10	12	23	60734.8551	0.648E−07	4
8	21	26	24	0.253357740E−08	0.800E−06	4
9	15	16	36	2216.45871	0.104E−10	4
10	12	18	21	191.511336	0.524E−07	4
11	2587	2593	2649	0.647358980E−27	0.359E−12	3
12	16	20	23	19264.6341	0.513E−10	4
13	17	21	28	131234.018	0.784E−08	4
14	5	8	18	108.517888	0.227E−08	4
15	6	7	15	18.1763146	0.290E−06	4
16	15	21	40	2.51109677	0.724E−06	4
17	15	20	19	0.257973699E−16	0.275E−08	3
18	42	44	45	0.151517993E−24	0.122E−10	3
19	15	16	23	0.354943701E−14	0.255E−06	4
20	26	27	29	0.378161520E−10	0.407E−07	4
21	10	11	17	647.828517	0.773E−11	4
22	26	32	45	4486.97024	0.602E−07	4
Σ	4290	4357	4535		TIME = 4.56	

Table 18: Results obtained by program TGADU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1011	1013	1013	0.00000000	0.000E+00	3
2	260	273	508	1959.28649	0.439E−12	4
3	10	12	13	0.784354965E−09	0.868E−06	4
4	14	18	19	134.761343	0.827E−08	4
5	4	5	7	0.438081882E−11	0.697E−06	4
6	6	7	13	0.791460684E−17	0.934E−08	3
7	22	23	61	145814.000	0.000E+00	4
8	25	32	28	0.978141069E−06	0.782E−06	4
9	44	45	153	2220.17880	0.181E−09	4
10	12	19	21	191.511336	0.301E−07	4
11	3977	2992	2990	0.00000000	0.000E+00	3
12	29	30	50	67887.2385	0.438E−12	4
13	19	20	36	147906.000	0.000E+00	4
14	1	2	6	126.690556	0.000E+00	4
15	24	27	81	18.1763146	0.203E−10	4
16	46	50	135	3.59074140	0.470E−10	4
17	11	12	15	0.969524252E−21	0.171E−10	3
18	0	1	3	0.00000000	0.000E+00	3
19	26	30	34	0.202602070E−14	0.193E−06	4
20	929	930	2780	498.800124	0.359E−05	2
21	20	21	33	649.598077	0.280E−08	4
22	24	31	55	4488.96148	0.242E−07	4
Σ	6514	5593	8054		TIME = 7.77	

Table 19: Results obtained by program TGADS

Problem	NIT	NFV	NFG	F	G	ITERM
1	1108	1110	1110	0.00000000	0.000E+00	3
2	624	640	649	66.4089431	0.283E−07	4
3	11	12	14	0.202412411E−09	0.210E−06	4
4	11	13	17	134.749772	0.592E−07	4
5	4	5	7	0.116836300E−10	0.908E−06	4
6	6	7	13	0.787821743E−26	0.311E−12	3
7	17	40	29	60734.8551	0.428E−05	6
8	22	25	25	0.127726626E−07	0.160E−06	4
9	13	15	38	2216.45871	0.846E−06	4
10	129	147	176	191.511336	0.104E−07	4
11	3010	3016	3012	0.402368464E−24	0.902E−11	3
12	205	226	236	22287.9069	0.449E−08	4
13	123	132	152	131234.018	0.743E−09	4
14	7	8	32	108.517888	0.148E−07	4
15	13	20	42	18.1763146	0.445E−05	2
16	14	15	35	2.51109677	0.103E−09	4
17	29	34	33	0.139780007E−09	0.238E−06	4
18	49	53	52	0.119511868E−21	0.344E−09	3
19	15	16	23	0.339085307E−13	0.788E−06	4
20	17	18	32	0.336618309E−11	0.137E−07	4
21	15	18	23	647.696136	0.262E−06	4
22	47	59	98	4486.97024	0.663E−08	4
Σ	5489	5629	5848	NCG = 8282	TIME = 5.01	

Table 20: Results obtained by program TGACU

Problem	NIT	NFV	NFG	F	G	ITERM
1	1028	1031	1030	0.00000000	0.000E+00	3
2	263	262	511	1959.28649	0.819E−10	4
3	10	12	13	0.814133878E−09	0.897E−06	4
4	11	14	17	134.761343	0.206E−10	4
5	4	5	7	0.438081882E−11	0.697E−06	4
6	6	7	13	0.791460667E−17	0.934E−08	3
7	15	16	42	145814.000	0.000E+00	4
8	17	18	20	0.201167216E−07	0.162E−06	4
9	54	55	203	2220.17880	0.614E−10	4
10	95	105	126	191.511336	0.527E−06	4
11	4577	3591	3587	0.00000000	0.000E+00	3
12	49	50	84	67887.2385	0.351E−07	4
13	18	19	41	147906.000	0.000E+00	4
14	1	2	6	126.690556	0.000E+00	4
15	33	72	85	18.1763146	0.182E−04	6
16	8	12	29	3.59074140	0.542E−06	4
17	25	29	29	0.716457302E−10	0.922E−07	4
18	0	1	3	0.00000000	0.000E+00	3
19	28	32	36	0.209831733E−13	0.620E−06	4
20	937	938	2806	498.800124	0.572E−12	4
21	21	22	34	649.598077	0.324E−08	4
22	44	51	89	4488.96148	0.645E−10	4
Σ	7244	6344	8811	NCG = 11206	TIME = 6.56	

Table 21: Results obtained by program TGACS

Problem	NIT	NFV	NFG	C	G	ITERM
1	53	66	54	0.260681654E−14	0.120E−07	4
2	107	160	108	0.502441303E−12	0.262E−06	4
3	33	41	34	0.535439934E−08	0.814E−06	4
4	51	93	52	0.540217976	0.383E−06	4
5	23	24	24	0.132910215E−08	0.482E−06	4
6	46	52	47	0.216701250E−08	0.436E−06	4
7	48	113	49	0.260162540	0.430E−06	4
8	21	58	22	282.380956	0.806E−06	4
9	59	146	60	0.185849706	0.287E−06	4
10	159	215	160	−0.251638288	0.250E−06	4
11	70	96	71	0.538829394E−01	0.139E−07	4
12	136	245	137	0.941962422	0.207E−06	4
13	2	4	3	0.456380111E−19	0.304E−11	3
14	5	6	6	0.162409086E−08	0.671E−06	4
15	116	120	117	0.199003538E−01	0.436E−06	4
16	110	214	111	−0.388943896E−02	0.518E−05	2
17	33	49	34	−0.110417781E−06	0.764E−06	4
18	62	77	63	0.744234285E−09	0.611E−07	4
19	9	23	10	42.6746811	0.373E−09	4
20	25	32	26	−0.497512435E−02	0.422E−06	4
21	16	23	17	0.298487756E−01	0.595E−06	4
22	32	82	33	0.577532726E−02	0.972E−07	4
Σ	1216	1939	1238		TIME = 0.75	

Table 22: Results obtained by program TMAXU

Problem	NIT	NFV	NFG	C	G	ITERM
1	337	355	338	0.193178806E−13	0.254E−05	3
2	127	151	128	0.120336380E−12	0.424E−05	3
3	25	28	26	0.383710546E−09	0.331E−06	4
4	67	77	68	126.863549	0.358E−02	2
5	6	7	7	0.494049246E−14	0.666E−07	3
6	13	17	14	0.663150090E−13	0.141E−06	3
7	73	108	74	2391.16999	0.209E+00	2
8	241	243	242	0.383133726E−07	0.708E−06	4
9	209	251	209	552.682636	0.267E+01	−6
10	84	106	85	131.888475	0.242E−05	2
11	732	751	733	0.799693645E−12	0.581E−05	3
12	203	237	204	612.723020	0.601E−04	2
13	90	111	91	2940.50941	0.245E−03	2
14	84	107	85	112.314955	0.480E−05	2
15	39	64	40	36.0935678	0.163E−01	2
16	67	108	67	13.2000005	0.139E−03	6
17	337	344	338	0.100472795E−13	0.167E−06	3
18	3637	3794	3638	0.199840144E−14	0.419E−09	3
19	23	24	24	0.938360500E−12	0.217E−04	3
20	22	44	22	0.121058719E−11	0.398E−05	6
21	65	90	66	262.921649	0.108E−05	2
22	608	627	609	593.367735	0.525E−02	2
Σ	7089	7644	7108		TIME = 2.80	

Table 23: Results obtained by program TSUMU

Problem	NIT	NFV	NFG	F	G	ITERM
1	10	41	0	0.224531E−22	0.168207E−07	3
2	9	46	0	0.106897E−22	0.163517E−06	3
3	3	19	0	0.333989E−19	0.223053E−06	3
4	7	23	0	0.348196E−17	0.177085E−02	3
5	12	63	0	0.117206E−16	0.694210E−06	3
6	17	52	0	0.110919E−16	0.167579E−11	3
7	13	41	0	0.339913E−19	0.457009E−03	3
8	13	73	0	0.125748E−25	0.193922E−04	3
9	13	99	0	0.432936E−21	0.201706E−03	3
10	5	41	0	0.803846E−25	0.415983E−03	3
11	12	37	0	0.189327E−25	0.423583E−05	3
12	18	55	0	0.129272E−16	0.713317E−13	3
13	18	39	0	0.105290E−16	0.341327E−13	3
14	4	13	0	0.774783E−20	0.441968E−05	3
15	5	36	0	0.182567E−17	0.471251E−03	3
16	53	319	0	0.462169E−17	0.153957	3
17	14	48	0	0.449140E−22	0.105525E−03	3
18	27	82	0	0.249708E−20	0.571681E−05	3
19	2	7	0	0.309324E−21	0.370062E−09	3
20	13	43	0	0.428279E−20	0.203421E−07	3
21	12	37	0	0.200623E−20	0.255404E−10	3
22	7	50	0	0.195350E−19	0.106707E−05	3
23	29	262	0	0.390327E−17	0.200697E−10	3
24	6	31	0	0.822526E−23	0.812457E−09	3
25	9	46	0	0.147127E−23	0.395357E−09	3
26	12	61	0	0.608837E−17	0.420862E−07	3
27	10	51	0	0.275078E−20	0.121824E−06	3
28	10	60	0	0.229532E−16	0.213811E−05	3
29	4	53	0	0.124549E−19	0.130673E−05	3
30	12	162	0	0.222959E−21	0.107876E−07	3
Σ	379	1990	0	TIME = 4.77		

Table 24: Results obtained by program TEQNU

Problem	NIT	NFV	NFG	F	G	ITERM
1	30	64	0	0.326079E−18	0.154142E−03	3
2	17	57	0	0.720058E−19	0.261551E−07	3
3	5	11	0	0.861220E−16	0.366389E−03	3
4	11	19	0	0.115060E−18	0.358897E−01	3
5	20	56	0	0.335602E−16	0.121910E−06	3
6	22	31	0	0.167377E−16	0.898624E−08	3
7	25	42	0	0.137004E−20	0.185851E−05	3
8	21	60	0	0.496243E−28	0.183782E−07	3
9	32	71	0	0.220876E−21	0.800603E−05	3
10	9	24	0	0.202316E−20	0.162996E−03	3
11	16	23	0	0.116022E−21	0.130018E−02	3
12	23	40	0	0.861690E−16	0.190460E−08	3
13	24	32	0	0.234892E−16	0.204525E−08	3
14	8	13	0	0.596974E−21	0.811563E−05	3
15	12	28	0	0.124901E−17	0.305897	3
16	22	78	0	0.984840E−20	0.125407E−03	3
17	17	43	0	0.130235E−20	0.154659E−04	3
18	46	61	0	0.224793E−17	0.116353E−01	3
19	2	5	0	0.704403E−18	0.221630E−06	3
20	18	30	0	0.158787E−16	0.312477E−03	3
21	25	34	0	0.233925E−16	0.135133E−05	3
22	14	45	0	0.189862E−17	0.128826E−01	3
23	23	106	0	0.194742E−18	0.550497E−08	3
24	20	53	0	0.737500E−17	0.611156E−08	3
25	29	50	0	0.208794E−17	0.413643E−08	3
26	36	67	0	0.132055E−17	0.481013E−08	3
27	40	75	0	0.659356E−17	0.862034E−08	3
28	27	83	0	0.461856E−18	0.268680E−08	3
29	12	95	0	0.206962E−16	0.754042E−08	3
30	18	145	0	0.740533E−16	0.167985E−07	3
Σ	624	1541	0	TIME = 4.13		

Table 25: Results obtained by program TEQLU

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., Matonoha C., Vlček J.: Primal interior-point method for large sparse minimax optimization. Technical Report V-941, Prague, ICS AS CR, 2005.

[17] Lukšan L., Matonoha C., Vlček J.: Trust-region interior point method for large sparse l_1 optimization. Technical Report V-942, Prague, ICS AS CR, 2005.

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

[19] Lukšan L., Tůma M., Hartman J., Vlček J., Ramešová N., Šiška M., Matonoha C.: Interactive System for Universal Functional Optimization (UFO). Version 2006. Technical Report V-977. Prague, ICS AS CR 2006.

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

[21] 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.

[22] Lukšan L., Vlček J.: Variable metric method for minimization of partially separable nonsmooth functions. Pacific Journal on Optimization 2 (2006).

[23] 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.

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

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

[26] 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.

[27] Schlick T., Fogelson A.: TNPACK – A Truncated Newton Minimization Package for Large-Scale Problems (Parts I and II). ACM Transactions on Mathematical Software 18 (1992) 46-111.

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

[29] 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.

[30] Toint P.L.: Subroutine VE08. Harwell Subroutine Library. Specifications (Release 12), Vol, 2, AEA Technology, December 1995, 1162-1174.

[31] Toint P.L.: Subroutine VE10. Harwell Subroutine Library. Specifications (Release 12), Vol, 2, AEA Technology, December 1995, 1187-1197.

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

[33] Vlček J., Lukšan L.: Shifted limited-memory variable metric methods for large-scale unconstrained minimization. J. of Computational and Applied Mathematics, 186 (2006) 365-390.

[34] Watson L.T., Melville R.C., Morgan A.P., Walker H.F: Algorithm 777. HOMPACK90: A Suite of Fortran 90 Codes for Globally Convergent Homotopy Algorithms. ACM Transactions on Mathematical Software 23 (1997) 514-549.

[35] Zhu C., Byrd R.H., Lu P., Nocedal J.: Algorithm 778. L-BFGS-B: Fortran Subroutines for Large-Scale Bound Constrained Optimization. ACM Transactions on Mathematical Software 23 (1997) 550-560.