*Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 1, 13–18*

# NEWTON'S ITERATION WITH A CONJUGATE GRADIENT BASED DECOMPOSITION METHOD FOR AN ELLIPTIC PDE WITH A NONLINEAR BOUNDARY CONDITION

JONAS KOKO*

* LIMOS, Université Blaise Pascal, CNRS UMR 6158
ISIMA, Campus des Cézeaux – BP 10125, F–63173 Aubière cedex, France
e-mail: koko@sp.isima.fr

Newton's iteration is studied for the numerical solution of an elliptic PDE with nonlinear boundary conditions. At each iteration of Newton's method, a conjugate gradient based decomposition method is applied to the matrix of the linearized system. The decomposition is such that all the remaining linear systems have the same constant matrix. Numerical results confirm the savings with respect to the computational cost, compared with the classical Newton method with factorization at each step.

**Keywords:** Newton's method, conjugate gradient method, nonlinear PDE

## 1. Introduction

Newton's method is a standard tool for numerical solution of nonlinear equations (Dennis and Schnabel, 1996; Hughes *et al.*, 1987; Ortega and Rhainboldt, 1970; Sonnenveld *et al.*, 1985), replacing the original problem with a sequence of linearized problems. The matrix of the linearized problem changes at each iteration, and using direct linear solvers results in too many matrix factorizations. Except for very special cases (see, e.g., (Golub and Van Loan, 1989, §12.6; Golub *et al.*, 1974)), a factorization ($LU$, $LDL^T$, $RR^T$, etc.) cannot be updated even though only few entries of the matrix have changed.

In this paper we propose a Newton/Conjugate Gradient (NCG) method for an elliptic PDE with a nonlinear boundary condition. The nonlinear boundary condition implies that only a small number of matrix entries change at Newton's iteration. To avoid matrix factorizations at each iteration, we apply a conjugate gradient based decomposition to treat the varying part of the matrix for the linearized problem. Then the remaining linear systems have the same constant matrix. If a Cholesky factorization is performed at the initialization step, the solutions to all the linear systems during the iterative process then reduce to backward-forward substitutions.

The paper is organized as follows: In Section 2 we recall Newton's method using a model problem. The conjugate gradient decomposition method is presented in Section 3, followed by its application to the model problem in Section 4. Finally, some numerical experiments are presented in Section 5.

## 2. Newton's Method for a Model Problem

Let $\Omega$ be a bounded open set in $\mathbb{R}^2$ with boundary $\Gamma$. We assume that $\Gamma = \Gamma_D \cup \Gamma_N$, with $\Gamma_D \cap \Gamma_N = \emptyset$, $\mathrm{meas}(\Gamma_D) > 0$ and $\mathrm{meas}(\Gamma_N) > 0$. Consider the model problem

$$-\Delta u = f \qquad \text{in} \quad \Omega, \tag{1}$$

$$u = 0 \qquad \text{on} \quad \Gamma_D, \tag{2}$$

$$\frac{\partial u}{\partial n} = -\varphi(u) \qquad \text{on} \quad \Gamma_N, \tag{3}$$

where $\varphi(u) = |u|^\alpha u$, for $\alpha \geq 2$.

Let $H^1(\Omega)$ denote the standard Hilbert space with respect to the domain $\Omega$. To write down the variational formulation of (1)–(3), we define the subspace

$$V = \left\{ v \in H^1(\Omega),\ v = 0 \text{ on } \Gamma_D \right\}$$

and introduce the following notation:

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x,$$

$$(f,v) = \int_\Omega f v \,\mathrm{d}x,$$

$$(u,v)_{\Gamma_N} = \int_{\Gamma_N} uv \,\mathrm{d}\Gamma.$$

The weak formulation of (1)–(3) is then given by the nonlinear variational problem: Find $u \in V$ such that

$$a(u,v) + \big(\varphi(u), v\big)_{\Gamma_N} = (f,v), \quad \forall v \in V. \tag{4}$$

We can introduce the functional $J$ defined by

$$J(u) = \frac{1}{2}a(u,u) + \frac{1}{\alpha+2}\int_{\Gamma_N} |u|^{\alpha+2}\,\mathrm{d}\Gamma - (f,u). \quad (5)$$

Then the variational equation (4) characterizes stationary points of (5), i.e., solving the nonlinear variational equation (4) is equivalent to finding $u \in V$ such that

$$J(u) \le J(v), \quad \forall v \in V. \quad (6)$$

Since the functional $J$ is convex and Gâteaux differentiable, the minimization problem (6) (and therefore the variational equation (4)) has a unique solution.

To solve the nonlinear equation (4), we can use Newton's iterative method. Starting from an initial guess $u^0$, Newton's method applied to (4) results in the following iteration for $n \ge 0$:

$$\begin{aligned} &w^n \in V,\\ &a(w^n,v) + \big(\varphi'(u^n)w^n, v\big)_{\Gamma_N} = b(u^n,v),\\ &\qquad\qquad \forall v \in V, \quad (7)\\ &u^{n+1} = u^n + w^n, \quad (8) \end{aligned}$$

where

$$\big(\varphi'(u^n)w^n, v\big)_{\Gamma_N} = \int_{\Gamma_N} (\alpha+1)|u^n|^\alpha w^n v\,\mathrm{d}\Gamma,$$

$$b(u^n,v) = (f,v) - a(u^n,v) - \big(\varphi(u^n), v\big)_{\Gamma_N}.$$

Consider a discretization of (4) over $\Omega$, assuming that $V_h \subset H^1(\Omega)$ is a linear finite element space. The finite element approximation of (4) is

$$u_h \in V_h;\ a(u_h,v_h) + \big(\varphi(u_h), v_h\big)_{\Gamma_n} = (f,v_h), \quad \forall v_h \in V_h. \quad (9)$$

Newton's iteration (7)–(8) becomes

$$\begin{aligned} &w_h^n \in V_h,\\ &a(w_h^n,v_h) + \big(\varphi'(u_h^n)w_h^n, v_h\big)_{\Gamma_N} = b(u_h^n,v_h),\\ &\qquad\qquad \forall v_h \in V_h, \quad (10)\\ &u_h^{n+1} = u_h^n + w_h^n, \quad (11) \end{aligned}$$

and Newton's iteration (10)–(11) is equivalent to the following sequence of algebraic problems for $n \ge 0$:

$$A\boldsymbol{w}^n + D(\boldsymbol{u}^n)\boldsymbol{w}^n = b(\boldsymbol{u}^n), \quad (12)$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \boldsymbol{w}^n, \quad (13)$$

where $A$ is a symmetric positive definite matrix and $D(\boldsymbol{u}^n)$ is a diagonal matrix with a small number of nonzero entries (corresponding to the nodes of $\Gamma_N$). Since the matrix of the linear system (12) changes from iteration to iteration, using a direct linear solver may make Newton's method prohibitively expensive to use. Moreover, classical solvers do not take into account the fact that only a small number of entries of the matrix in (12) change at each iteration due to the structure of $D(\boldsymbol{u}^n)$.

## 3. Decomposition Method

Let $A$ be an $N \times N$ nonsingular matrix, $D$ an $N \times N$ diagonal matrix and $b \in \mathbb{R}^N$. We first consider the following linear system:

$$Ax + Dx = b. \quad (14)$$

The main idea of our method is to solve (14) using only the factorization of $A$. Now, consider the linear system

$$Ax = b - Dy, \quad (15)$$

keeping in mind that in (12), only the matrix $D$ varies from iteration to iteration. For an arbitrary $y$ in $\mathbb{R}^N$, the solution of (15) is not that of (14). However, we know that for $y = x$, the solutions of (15) and (14) coincide.

Let $P$ be a symmetric, positive definite matrix. With $P$ we can define the energy norm

$$|x|_P = \sqrt{(Px,x)_{\mathbb{R}^N}}, \quad (16)$$

where $(\cdot,\cdot)_{\mathbb{R}^N}$ denotes the canonical Euclidean inner product in $\mathbb{R}^N$. In our decomposition method we choose $y$ in (15) as a minimizer of the real-valued function

$$F_\varepsilon(y) = \frac{1}{2}|y-x|_P^2 + \frac{\varepsilon}{2}|y|_P^2, \quad (17)$$

with $\varepsilon \in ]0,1[$ as the penalty parameter. The function $F_\varepsilon$ above is quadratic and coercive since

$$F_\varepsilon(y) \ge \frac{\varepsilon}{2}|y|_P^2.$$

Instead of the original problem (14), we solve the following minimization problem: Find $y^* \in \mathbb{R}^N$ such that

$$F_\varepsilon(y^*) \le F_\varepsilon(y), \quad \forall y \in \mathbb{R}^N. \quad (18)$$

Note that the solution of (18) is only an approximate solution of the original problem (14). Then it remains to show that the solution of (18) converges to that of (14) as $\varepsilon \to 0$.

**Theorem 1.** (Convergence with vanishing penalty parameter) *Let $y^\varepsilon$ be the solution of (18) and $x^\varepsilon$ the corresponding solution of (15) for a penalty parameter $\varepsilon \in ]0,1[$. Let $x$ be the solution of (14). Then $y^\varepsilon \to x$ and $x^\varepsilon \to x$ as $\varepsilon \to 0$.*

*Proof.* Since $y^\varepsilon$ is a minimizer of $F_\varepsilon$, for all $\varepsilon > 0$ we have $F_\varepsilon(y^\varepsilon) \le F_\varepsilon(x)$, i.e.,

$$|y^\varepsilon - x^\varepsilon|_P^2 + \varepsilon|y^\varepsilon|_P^2 \le \varepsilon|x|_P^2 < |x|_P^2. \tag{19}$$

Then the sequence $(y^\varepsilon, x^\varepsilon)$ is uniformly bounded in $\mathbb{R}^N \times \mathbb{R}^N$. Thus there exists a subsequence, also denoted by $(y^\varepsilon, x^\varepsilon)$, which converges to an element $(\bar{y}, \bar{x})$ in $\mathbb{R}^N \times \mathbb{R}^N$ as $\varepsilon \to 0$. From (19) we deduce that $\bar{y} = \bar{x}$. By passing to the limit in (14), it follows that $\bar{x}$ solves (14), and thus $\bar{x} = x$ by the uniqueness. ■

The mapping $y \mapsto x = x(y)$ is linear and continuous, so we have

$$x(y + td) = x + tw,$$

where $w$ is the solution of the sensitivity system

$$Aw = -Dd. \tag{20}$$

The first directional derivative of $F_\varepsilon$ is then given by

$$\frac{\partial}{\partial y}F_\varepsilon(y)\cdot d = \big(P(y-x), d-w\big)_{\mathbb{R}^N} + \varepsilon(Py, d)_{\mathbb{R}^N}, \quad \forall y, d \in \mathbb{R}^N. \tag{21}$$

In order to construct descent directions for $F_\varepsilon$, we need the gradient vector $g := \nabla F_\varepsilon(y)$ defined by

$$\frac{\partial}{\partial y}F_\varepsilon(y)\cdot d = \big(\nabla F_\varepsilon(y), d\big)_{\mathbb{R}^N}.$$

To this end, we can, e.g., compute $w$ explicitly in (20) by inverting $A$, i.e.,

$$w = -A^{-1}Dd. \tag{22}$$

Substituting (22) into (21), we get

$$\frac{\partial}{\partial y}F_\varepsilon(y)\cdot d = \big(P(y-x), d + A^{-1}Dd\big)_{\mathbb{R}^N} + \varepsilon(Py, d)_{\mathbb{R}^N}.$$

After elementary algebraic calculations, we get

$$\frac{\partial}{\partial y}F_\varepsilon(y)\cdot d = \big((I + DA^{-1})P(y-x), d\big)_{\mathbb{R}^N} + \varepsilon(Py, d)_{\mathbb{R}^N},$$

where $I$ is the identity matrix. Therefore we deduce that

$$g = \nabla F_\varepsilon(y) = (I + DA^{-1})P(y-x) + \varepsilon Py. \tag{23}$$

In practice, to avoid the computation of $A^{-1}$, the gradient $g = \nabla F_\varepsilon(y)$ is obtained in two steps as follows:

$$Az = P(y-x), \tag{24}$$

$$g = Dz + P(\varepsilon_1 y - x), \tag{25}$$

where $\varepsilon_1 = 1 + \varepsilon$.

To solve (18), we now consider methods of the general type, for $y^0$ arbitrarily given,

$$y^{k+1} = y^k + t_k d^k, \quad k = 0, 1, 2, \dots . \tag{26}$$

The *descent direction* $d^k$ will, in general, be constructed using the gradient obtained with (24) and (25). If at each step $k$, setting $d^k = -g^k = -\nabla F_\varepsilon(y^k)$, we obtain a gradient based algorithm. As $F_\varepsilon$ is a quadratic function, a conjugate gradient algorithm is the best optimization algorithm in terms of computational efforts and storage requirements because (theoretically) it converges in a finite number of iterations. For detailed studies of conjugate gradient methods, see, e.g., (Luenberger, 1989; Polak, 1971). At each step $k$, the conjugate gradient direction (in the Fletcher-Reeves version) is given by

$$d^k = -g^k + \beta_k d^{k-1},$$

$$\beta_k = \frac{|g^k|^2}{|g^{k-1}|^2},$$

with $|\cdot|$ being the Euclidean norm.

For a given descent direction $d^k$, the step size $t_k$ is computed in such a way so as to minimize the real-valued function

$$\psi(t) = F_\varepsilon(y^k + td^k), \quad t > 0.$$

Since $F_\varepsilon$ is a quadratic function, $t_k$ is the unique solution of the linear equation

$$\psi'(t) = \frac{\partial}{\partial y}F_\varepsilon(y^k + td^k)\cdot d^k = \big(\nabla F_\varepsilon(y^k + td^k), d^k\big)_{\mathbb{R}^N} = 0.$$

Using (23), after a straightforward calculation, we obtain

$$t_k = -\frac{(g^k, d^k)_{\mathbb{R}^N}}{|d^k - w^k|_P^2 + \varepsilon|d^k|_P^2}, \tag{27}$$

where $w^k$ is the solution of

$$Aw^k = -Dd^k.$$

We can now present our conjugate gradient based decomposition algorithm:

**Algorithm CG:**

**Step 0.** Initialization: $y^0$ arbitrarily given.
$Ax^0 = b - Dy^0$,
$Az^0 = P(y^0 - x^0)$,
$g^0 = Dz^0 + P(\varepsilon_1 y^0 - x^0)$,
$d^0 = -g^0$.

**Step $k\geq 0$.** Assume that $y^k$, $x^k$ and $d^k$ are known.

Descent:

$Aw^k = -Dd^k$.

Compute $t_k$ with (27):

$y^{k+1} = y^k + t_k d^k$,

$x^{k+1} = x^k + t_k w^k$.

New direction:

$Az^{k+1} = P(y^{k+1} - x^{k+1})$,

$g^{k+1} = Dz^{k+1} + P(\varepsilon_1 y^{k+1} - x^{k+1})$,

$\beta_k = |g^{k+1}|^2/|g^k|^2$,

$d^{k+1} = -g^{k+1} + \beta_k d^k$.

We stop iterating as soon as $|g^k|/|g^0|$ is sufficiently small. At each step of Algorithm CG, we solve two linear systems (20) and (24). Noting that the two linear systems have the same matrix which does not change during the iterative process, a Cholesky factorization can be performed only once. The solution to (20) and (24) during the iterative process then reduces to backward-forward substitutions.

An important point that remains to be dealt with is the choice of the matrix $P$ used to define the energy norm (16). The matrix $P$ is a preconditioner. A suitable choice for $P$ guarantees a fast and numerically scalable[1] conjugate gradient algorithm. Our choice for numerical experiments of Section 5 is $P = \mathrm{diag}(P_{11}, \ldots, P_{NN})$ with

$$P_{ii} = 1 + D_{ii}/A_{ii}, \quad i = 1, \ldots, N,$$

derived from (23) since the conjugate gradient algorithm solves $\nabla F_\varepsilon(y) = 0$.

## 4. Application to Newton's Iteration

Applying the conjugate gradient decomposition algorithm of the previous section to Newton's iteration (12)–(13) is quite straightforward. It suffices to solve (12) with Algorithm CG.

### Algorithm NCG:

**Step 0.** Guess $\boldsymbol{u}^0$.

**Step $n\geq 0$.** Compute $D^n := D(\boldsymbol{u}^n)$ and $b^n := b(\boldsymbol{u}^n)$.

Solve the system

$A\boldsymbol{w}^n + D^n\boldsymbol{w}^n = b^n$

using Algorithm CG.

Update the approximate solution: $\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \boldsymbol{w}^n$.

[1] An iterative method is said to be *numerically scalable* if its convergence properties, e.g., the number of iterations needed for convergence are asymptotically independent of the size of the problem to be solved.

It is now clear that one factorization (at the initialization step) of the matrix $A$ suffices to solve all linear systems of Algorithm NCG.

Note that since $D^n$ has a small number of nonzero entries, operations involving $D^n$ in Algorithm CG must be adapted to reduce the computational cost.

## 5. Numerical Results

Algorithm NCG outlined in the previous sections was implemented in Fortran 90, in double precision arithmetic, on an SGI Origin200 Server. In this section we report some numerical results. The stopping criterion of Newton's iterations is

$$\frac{|\boldsymbol{w}^n|}{|\boldsymbol{u}^n|} < 10^{-6}. \tag{28}$$

Note that we do not need high accuracy in solving (12) with the conjugate gradient decomposition algorithm. Then, for the conjugate gradient decomposition algorithm, we use the following stopping criterion:

$$\frac{|g^k|}{|g^0|} < \varepsilon_n = \max\left\{0.1\varepsilon_{n-1},\ 10^{-4}\right\},$$

where the subscript $n$ indicates the index of Newton's iteration, with $\varepsilon_0 = 0.1$.

We choose the domain $\Omega = (0,1)\times(0,1)$ and the right-hand side

$$f(x,y) = \frac{2c''(y)}{c(y)+\frac{x}{\sqrt{2}}} - \frac{2c'(y)}{\left(c(y)+\frac{x}{\sqrt{2}}\right)^2} - e^{-1}\left(c(y)+\frac{x}{\sqrt{2}}\right)^2,$$

$$c(y) = e^{-y}$$

taken from (Abbasian and Carey, 1998).

The domain was first discretized by a nonuniform mesh consisting of 177 nodes, 312 triangles and 11 nodes on $\Gamma_N$. This initial mesh was successively refined to produce meshes with 665, 2577, 10145 and 40257 nodes, respectively. Performances of the classical Newton-Raphson method (NR method in what follows), with a factorization at each step, are reported in Tables 1–3. Performances of Algorithm NCG are reported in Tables 4–6. The CPU times reported in all tables were obtained by the Fortran 90 intrinsic subroutine `cpu_time`, called just before and after Newton's loop.

First, we notice that, in Algorithm NCG, the conjugate gradient decomposition algorithm does not alter the number of Newton's iterations. We also notice that the

Table 1. Performances of the Newton-Raphson algorithm for $\alpha = 2$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Number of iterations | 4 | 4 | 4 | 4 | 4 |
| CPU time (in sec.) | 0.01 | 0.057 | 0.378 | 4.371 | 53.209 |

Table 2. Performances of the Newton-Raphson algorithm for $\alpha = 3$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Number of iterations | 4 | 4 | 4 | 4 | 4 |
| CPU time (in sec.) | 0.01 | 0.055 | 0.379 | 4.404 | 53.121 |

Table 3. Performances of the Newton-Raphson algorithm for $\alpha = 5$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Number of iterations | 4 | 4 | 4 | 4 | 4 |
| CPU time (in sec.) | 0.01 | 0.058 | 0.379 | 4.369 | 53.141 |

Table 4. Performances of Algorithm NCG for $\alpha = 2$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Newton iterations | 4 | 4 | 4 | 4 | 4 |
| CG iterations | 10 | 10 | 10 | 10 | 10 |
| Speed up | 1.42 | 1.5 | 1.58 | 1.52 | 1.72 |

Table 5. Performances of Algorithm NCG for $\alpha = 3$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Newton iterations | 4 | 4 | 4 | 4 | 4 |
| CG iterations | 6 | 6 | 6 | 6 | 6 |
| Speed up | 2 | 1.96 | 2.08 | 2.02 | 2.18 |

Table 6. Performances of Algorithm NCG for $\alpha = 5$.

| Number of $\Gamma_N$ nodes | 11 | 21 | 41 | 81 | 161 |
|---|---|---|---|---|---|
| Newton iterations | 4 | 4 | 4 | 4 | 4 |
| CG iterations | 6 | 6 | 6 | 6 | 6 |
| Speed up | 2 | 2.07 | 2.07 | 2.02 | 2.18 |

number of Newton's and conjugate gradient iterations required to satisfy (28) was virtually independent of the mesh size for a fixed $\alpha$. Therefore the number of solutions to the linear systems was independent of the mesh size. The speed up reported in Tables 4–6 show that the savings in the computational cost, obtained with Algorithm NCG, were significant. Even for small size problems, our algorithm is faster than the classical Newton algorithm.

To complete the study of the numerical behaviour of our algorithm, we examined the case when the Cholesky solver was replaced by the Preconditioned Conjugate Gradient (PCG) solver. Since iterative solvers are particularly useful for large sparse linear systems, we restricted our study to the problem with 40257 mesh nodes.

The preconditioner matrix was obtained by incomplete Cholesky factorization with drop tolerance (see, e.g., (Meurant, 1999, §8.5; Saad, 1990)). The Preconditioned Conjugate Gradient algorithm stopped with a relative precision of $10^{-5}$ in both the NR and NCG methods. Tables 7–9 summarize the performances of the two algorithms for drop tolerances $10^{-2}$, $10^{-4}$ and $10^{-6}$. The classical Newton algorithm stopped after 4 iterations in all tests presented whereas Algorithm NCG required 5 iterations for $\alpha \in \{2, 3\}$ and 4 iterations for $\alpha = 5$. We notice that for small drop tolerances, our algorithm is faster than the classical NR algorithm. This property can be useful for general elliptic problems since dropping small elements can more likely produce better preconditioners than dropping large elements.

Table 7. Performances of the Newton-Raphson (NR) and NCG algorithms with a preconditioned conjugate gradient solver for $\alpha = 2$.

| Drop tolerance | $10^{-2}$ | $10^{-4}$ | $10^{-6}$ |
|---|---|---|---|
| NR CPU time (in sec.) | 221.140 | 289.722 | 2184.271 |
| Speed-up with NCG | 0.50 | 1.53 | 3.42 |

Table 8. Performances of the Newton-Raphson (NR) and NCG algorithms with a preconditioned conjugate gradient solver for $\alpha = 3$.

| Drop tolerance | $10^{-2}$ | $10^{-4}$ | $10^{-6}$ |
|---|---|---|---|
| NR CPU time (in sec.) | 219.562 | 289.990 | 2185.254 |
| Speed-up with NCG | 0.50 | 1.55 | 3.43 |

Table 9. Performances of the Newton-Raphson (NR) and NCG algorithms with a preconditioned conjugate gradient solver for $\alpha = 5$.

| Drop tolerance | $10^{-2}$ | $10^{-4}$ | $10^{-6}$ |
|---|---|---|---|
| NR CPU time (in sec.) | 219.122 | 289.987 | 2191.885 |
| Speed-up with NCG | 0.64 | 1.82 | 3.55 |

## 6. Conclusion

We have shown that our Newton/Conjugate Gradient method is more efficient, in terms of speed, than the classical Newton method with factorization at each step. The implementation of the method is straightforward. Even with an iterative solver, the savings in the computational time can be significant with our method, if the preconditioner is obtained by an incomplete factorization with small drop tolerance.

## References

Dennis J.E. and Schnabel R.B. (1996): *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. — Philadelphia: SIAM.

Golub G.H. and Van Loan C.F. (1989): *Matrix Computations*. — Baltimore: The John Hopkins University Press.

Luenberger D. (1989): *Linear and Nonlinear Programming*. — Reading, MA: Addison Wesley.

Meurant G. (1999): *Computer Solution of Large Systems*. — Amsterdam: Elsevier.

Ortega J.M. and Rhainboldt W.C. (1970): *Iterative Solution of Nonlinear Equations in Several Variables*. — New York: Academic Press.

Polak E. (1971): *Computational Methods in Optimization*. — New York: Academic Press.

Abbasian R.O. and Carey G.F. (1998): *Hybrid* MPE-*iterative schemes for linear and nonlinear systems*. — Appl. Math. Comput., Vol. 26, pp. 277–291.

Golub G.H., Murray W. and Saunders M.A. (1974): *Methods for modifying matrix factorizations*. — Math. Comp., Vol. 28, No. 126, pp. 505–535.

Hughes J.T., Ferency R.M. and Halquist J.O. (1987): *Large-scale vectorized implicit calculations in solid mechanics on a* C*ray* X-MP*/48 utilizing* EBE *preconditioned conjugate gradient*. — Comput. Meth. Appl. Mech. Eng., Vol. 61, pp. 215–248.

Saad Y. (1990): *SPARSKIT: A basic tool kit for sparse matrix computation*. — Tech. Rep. CSRD TR 1029, University of Illinois, Urbana, IL.

Sonnenveld P., Wesseling P. and De Zeeuw P.M. (1985): *Multigrid and conjugate gradient methods as convergence acceleration technique*, In: Multigrid Meth. Integr. Diff. — pp. 117–167, Clarendon Press.

Received: 15 May 2003
Revised: 29 December 2003