ISSN 0965-5425, Computational Mathematics and Mathematical Physics, 2018, Vol. 58, No. 8, pp. 1247–1258. © Pleiades Publishing, Ltd., 2018.
Original Russian Text © N.G. Burago, I.S. Nikitin, 2018, published in Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 2018, Vol. 58, No. 8.

Dedicated to the memory of A.S. Kholodov

Matrix-Free Conjugate Gradient Implementation of Implicit Schemes

N. G. Burago[a],* and I. S. Nikitin[b],[c],**

[a] *Institute for Problems of Mechanics, Russian Academy of Sciences, Moscow, 119526 Russia*
[b] *Institute for Computer-Aided Design, Russian Academy of Sciences, Moscow, 123056 Russia*
[c] *Moscow Aviation Institute (National Research University), Moscow, 125080 Russia*
**e-mail: buragong@yandex.ru*
***e-mail: i_nikitin@list.ru*

Received March 5, 2018

Abstract—Matrix-free conjugate gradient algorithms are described as applied to large systems of algebraic equations arising in the implementation of implicit schemes intended for problems in continuum mechanics. The algorithms are shown to reduce the computer memory requirements and provide a higher computer performance. Additional advantages include the robustness of the algorithms and the simplicity of their implementation and debugging.

Keywords: matrix-free conjugate gradient method, large systems of algebraic equations, implicit schemes, continuum mechanics, robustness.

DOI: 10.1134/S0965542518080043

1. DESCRIPTION OF THE METHOD

The conjugate gradient method was proposed by Hestenes and Stiefel [1]. The idea behind the method is to iteratively minimize the quadratic functional

$$F(\mathbf{x}) = (C\mathbf{x}, \mathbf{x})/2 - (\mathbf{d}, \mathbf{x}),$$

which has a minimum on the solution of the original system of algebraic equations

$$C\mathbf{x} = \mathbf{d},$$

where C is a positive symmetric $N \times N$ matrix, N is the number of unknowns, $\mathbf{x}$ is the N-dimensional vector of unknowns, and $\mathbf{d}$ is the right-hand side vector. At every iteration step n, the search direction $\mathbf{s}_n$ is a linear superposition of the search direction $\mathbf{s}_{n-1}$ used at the preceding iteration and the direction of the gradient $\mathbf{g}_n = C\mathbf{x}_n - \mathbf{d}$ of the functional F:

$$\alpha_n = (\mathbf{g}_n, \mathbf{s}_n)/(C\mathbf{s}_n, \mathbf{s}_n), \quad \mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \mathbf{s}_n, \quad \mathbf{g}_{n+1} = \mathbf{g}_n - \alpha_n C\mathbf{s}_n, \tag{1.1}$$

$$\beta_n = (\mathbf{g}_{n+1}, C\mathbf{s}_n)/(C\mathbf{s}_n, \mathbf{s}_n), \quad \mathbf{s}_{n+1} = \mathbf{g}_{n+1} - \beta_n \mathbf{s}_n, \tag{1.2}$$

where $\mathbf{s}_0 = \mathbf{g}_0 = C\mathbf{x}_0 - \mathbf{d}$. Expressions for the coefficients α_n and β_n are obtained by minimizing the functional in the search direction. It has been proved (see, e.g., [2]) that the conjugate gradient method generates a C-orthogonal basis in the solution space:

$$(C\mathbf{s}_i, \mathbf{s}_j) = c_i \delta_{ij},$$

where $c_i > 0$ and δ_{ij} is the Kronecker delta. The projection of the solution onto the currently generated basis vector is determined at every iteration step. To implement the process, the initial residual of the equations is calculated at $n = 0$ and the homogeneous part of the residual Cs_n is then calculated at every iteration step. Overall, the algorithm makes use of four N-dimensional vectors:

$$\mathbf{a}_1 = \mathbf{x}_n, \quad \mathbf{a}_2 = \mathbf{s}_n, \quad \mathbf{a}_3 = \mathbf{g}_n, \quad \mathbf{a}_4 = C \cdot \mathbf{s}_n.$$

When the matrix of the original system of algebraic equations is nonsymmetric and/or neutral, the conjugate gradient method is applied to the modified system of equations $C^{\mathrm{T}}(C\mathbf{x}-\mathbf{d})=0$ and takes the form

$$\alpha_n=(\mathbf{g}_n,C\mathbf{s}_n)/(C\mathbf{s}_n,C\mathbf{s}_n),\quad \mathbf{x}_{n+1}=\mathbf{x}_n-\alpha_n\mathbf{s}_n,\quad \mathbf{g}_{n+1}=\mathbf{g}_n-\alpha_nC\mathbf{s}_n, \tag{1.3}$$

$$\beta_n=(C\mathbf{g}_{n+1},C\mathbf{s}_n)/(C\mathbf{s}_n,C\mathbf{s}_n),\quad \mathbf{s}_{n+1}=\mathbf{g}_{n+1}-\beta_n\mathbf{s}_n, \tag{1.4}$$

where $\mathbf{s}_0=\mathbf{g}_0=C\mathbf{x}_0-\mathbf{d}$. In this algorithm, the homogeneous part of the residual is computed twice (Cs_n and Cg_{n+1}) rather than once at every iteration step. Implementation requires five N-dimensional vectors: $\mathbf{a}_1=\mathbf{x}_n$, $\mathbf{a}_2=\mathbf{s}_n$, $\mathbf{a}_3=\mathbf{g}_n$, $\mathbf{a}_4=C\cdot\mathbf{s}_n$, and $\mathbf{a}_5=C\cdot\mathbf{g}_n$.

Note that the iterative process of the conjugate gradient method is implemented without forming or storing the matrix C of the system or performing any operations with it. It is only necessary to have an algorithm for computing the residual $C\mathbf{x}_n-\mathbf{d}$ and its homogeneous part $C\mathbf{x}_n$. Such an algorithm is significantly simpler than the formation of C. In the absence of roundoff errors, the exact solution is determined after N iterations (a basis of the finite-dimensional solution space is constructed, and solution projections are found). In other words, the iterative process converges after a finite number of iterations equal to the dimension of the vector of unknowns.

The stopping rule for the iterative process is specified by the condition

$$(g_n,g_n)\le\varepsilon\vee(s_n,s_n)\alpha_n^2\le\varepsilon,$$

where ε is a small number close to the machine epsilon (i.e., a number that, when added to 1, results in 1), which depends on the word length of the computer. For four-byte representation of real numbers, $\varepsilon\approx10^{-6}$.

The case $(As_n,s_n)\le\varepsilon$ $(A>0)$ or $(As_n,As_n)\le\varepsilon$ indicates that the formulation of the problem is incorrect (the matrix of the system is singular).

The computations are performed with a limited number of significant figures (digits) and the condition number of C can be much greater than 1. Accordingly, the errors in the determined basis vectors and the corresponding projections of the desired solution are accumulated at iteration. As a result, when conditioning is very poor (i.e., the condition number is very large), the iterations of the conjugate gradient method may not converge. For this reason, the system of equations to be solved is preconditioned by multiplying it by an approximate inverse matrix $\tilde{C}^{-1}$:

$$\tilde{C}^{-1}(C\mathbf{x}-\mathbf{d})=0,$$

where $\tilde{C}^{-1}C\approx E$ and E is the identity matrix. In the ideal case of $\tilde{C}^{-1}C=E$, this multiplication exactly solves the problem.

The most popular preconditioning methods are based on band approximations of C inverted by the Cholesky method [3], which preserves the width of band matrices. However, when the number of unknowns is large ($N>10^5\div10^6$), even the application of band approximations of C is associated with considerable difficulties, especially in three-dimensional problems with an inevitably large width of bands of nonzero elements.

It will be shown later that matrix operations can be completely eliminated from conjugate gradient algorithms. For initial–boundary value problems of mathematical physics in the case of finite-difference and finite-element approximations, the use of the conjugate gradient method has shown that its convergence is ensured by applying the simplest approximate inverse matrix composed of inverse diagonal elements of the original matrix, which is equivalent to rescaling the unknowns. The computation, storage, and inversion of the diagonal elements of C are easy to do.

That is, by analogy with Gaussian elimination, the conjugate gradient method requires a finite number of operations for finding a solution, but does not involve any operations with the matrix of the system of linear algebraic equations, including its formation and storage. To implement the conjugate gradient method, it is sufficient to construct an algorithm for determining the gradient $\mathbf{g}_n=C\mathbf{x}_n-\mathbf{d}$ and its homogeneous part $C\mathbf{x}_n$ for the approximate solution $\mathbf{x}_n$. Such an algorithm does not require matrices at all.

With a growing number of unknowns, the required computer memory increases linearly and is independent of the number of spatial coordinates. The required computer memory is proportional to $4N$–$5N$ depending on whether the operator C is symmetric and positive definite. An attractive quality of the method is also the ability (inherited from gradient methods) to capture the basic (in value) projections of the desired solution at the first iterations. As a result, solutions with machine precision (i.e., up to small values that, when added to unity, do not change its machine representation) are achieved after $\sqrt{N}$ iterations. Since the amount of computations at a single iteration step is proportional to N (as at time steps of explicit schemes), the amount of computations required for finding the solution grows with the number of unknowns at a rate proportional to $N^{3/2}$. For example, when the number of unknowns is equal to one million, the number of iterations required for determining the solution is approximately one thousand. Moreover, it will be shown below that the computation of gradients (in other words, the residuals of the equations) at every iteration step is entirely identical to the computation of solutions at a new time level for two-level explicit schemes.

It should also be noted that matrix-free iterative methods are preferable for parallelization of computations on multiprocessor computers.

There are numerous modifications of the conjugate gradient method (see, e.g., [2]). Version (1.1), (1.2) stems from gradient descent (the original quadratic functional is minimized at every iteration step). Version (1.3), (1.4) stems from the gradient minimal residual method (the residual norm is minimized at every iteration step).

2. MATRIX-FREE IMPLEMENTATION OF THE CONJUGATE GRADIENT METHOD

To implement conjugate gradient iterations, it is sufficient to have algorithms for computing the residual vectors of the homogeneous and inhomogeneous systems of equations, which are implemented by analogy with explicit schemes. Consider these algorithms as applied to continuum mechanics problems and variational grid methods.

Since continuum mechanics problems are generally nonlinear, quasilinearization is usually used for their solution. Nonstationary problems are quasi-linearized at every time step, while stationary problems are quasi-linearized at every iteration step of a quasi-Newtonian iterative process (versions of the Newton–Kantorovich method or differentiation with respect to a parameter). Next, at every time step or every iteration step with respect to nonlinearity, there arise auxiliary linear boundary value problems, which, in our case, are solved by the conjugate gradient method.

It is important to note that quasilinearization has to be performed on the original integrodifferential equations, since discrete nonlinear equations, as a rule, have no analytical representation and the only way of specifying nonlinear equations on a computer is based on algorithms for computing the residuals of such equations.

2.1. Formulation of a typical initial–boundary value problem. The formulation of continuum mechanics problems involves balance relations for the basic processes. In an arbitrarily moving frame of reference, they are generally written as follows [4]:

$$\frac{\partial}{\partial t}\int_V A\delta A dV + \int_V (\mathbf{B} - A\boldsymbol{\Omega})\cdot\nabla\delta A dV = \int_V f\delta A dV + \int_S (\mathbf{B} - A\boldsymbol{\Omega})\cdot\mathbf{n}\delta A dS. \tag{2.1}$$

Here, A is the preserved quantity, $\mathbf{B}$ is the flux of A caused by the interaction and chaotic motion of the molecules (elasticity, viscosity, diffusion), $A\boldsymbol{\Omega}$ is the convective flux of A (caused by ordered motion of the material medium relative to the coordinate medium), f is the source/sink of A, V is the solution domain with boundary S, $\boldsymbol{\Omega} = \mathbf{u} - \mathbf{w}$ is the convective velocity, and $\mathbf{u}$ and $\mathbf{w}$ are the velocities of the material and coordinate media. The time derivative $\partial/\partial t$ is calculated along the trajectories of arbitrarily moving coordinates (along the trajectories of nodes of the moving grid). Specifically, a given point is a Lagrangian node if $\mathbf{w} = \mathbf{u}$ and an Eulerian node if $\mathbf{w} = 0$. Here and below, the use of the term "coordinate medium" is justified by the fact that the control of moving coordinates (the motion of nodes and cells) is conveniently implemented in practice by representing the set of nodes and cells as a solid ("coordinate") medium.

The fluxes $\mathbf{B}$ are related to the gradients of the preserved dependent variables by continuous medium constitutive relations. For media of the differential type (media with short memory), they can be represented in a fairly general form as

$$\mathbf{B} = \mathbf{K}\cdot\nabla A + \mathbf{L}, \tag{2.2}$$

where the tensors $\mathbf{K}$ and $\mathbf{L}$ are given functions of coordinates, time, and the desired solution A and the dot denotes the inner product. The motion of the grid medium is assumed to be given or determined by solving a separate auxiliary "grid problem." Problem (2.1), (2.2) is supplemented with the initial conditions

$$t = 0: \quad A = A^0(\mathbf{x}) \tag{2.3}$$

and the boundary conditions

$$\mathbf{x} \in S_A: \quad A = A_*(\mathbf{x}, t), \tag{2.4}$$

$$\mathbf{x} \in S_B = S \backslash S_A: \quad \mathbf{B} \cdot \mathbf{n} = \mathbf{B}_*(\mathbf{x}, t). \tag{2.5}$$

The given functions are marked with a star. To discuss matrix-free conjugate gradient algorithms for implicit schemes, a further elaboration of the formulation of the typical continuum mechanics problem is not required.

2.2. Approximation of the solution and the equations. Suppose that a grid has been generated. It consists of N_1 nodes with coordinates $\mathbf{x}_i$ $(i = 1, \dots, N_1)$. The grid nodes are joined in N_2 cells specified by an integer information array $J_2(k, j)$ $(k = 1, \dots, N_2;\ j = 1, \dots, M_2)$, where $J_2(k, j)$ is the global index of local node j in cell k, N_2 is the number of cells, and M_2 is the number of nodes in a cell. The boundary grid is specified by an information array of boundary cells $J_3(k, j)$ $(k = 1, \dots, N_3;\ j = 1, \dots, M_3)$, where $J_3(k, j)$ is the global index of local node j in the boundary cell k, N_3 is the number of boundary cells, and M_3 is the number of nodes in a boundary cell.

In what follows, grid cells (in this case, Dirichlet cells) are referred to as finite elements. The value of the sought function at the node i at the time t_n (n is the time level number) is denoted by A_i^n. We use the simplest piecewise linear approximation of the solution. The values of the sought functions $[A]_k$ at the center of each element k are calculated using the formula

$$[A]_k = \sum_{j=1}^{M} \frac{1}{M} A_{J_2(k,j)}, \tag{2.6}$$

where M is the number of nodes in the element and $J_2(k, j)$ is the global index of the node with local index j in element k. Formula (2.6) is identical for internal and boundary cells (elements). In the general case, grids can have cells with a different number of nodes, so that M can be variable. However, this is of no importance, and the element index on the number M is omitted to simplify the notation.

The spatial derivatives are calculated using the formula

$$[\nabla A]_k = \sum_{j=1}^{M} \nabla_{kj} A_{J_2(k,j)}. \tag{2.7}$$

Expressions for the coefficients of the spatial differentiation operator ∇_{kj} can be found in handbooks, so they are not presented.

The integrals in the balance equations are computed using the simplest Gaussian quadrature rules (the sum of integrals over cells determined by the products of the integrands at the element center and the length, area, or volume of the cell, depending on the number of dimensions) with a Gaussian point at the center of the element, for example,

$$\int_{V_k^n} (\mathbf{B} - A\boldsymbol{\Omega}) \cdot \nabla \delta A d\tilde{V} = ([\mathbf{B}]_k^n - [A]_k^n [\boldsymbol{\Omega}]_k^n) \cdot [\nabla \delta A]_k^n V_k^n. \tag{2.8}$$

In the one-dimensional case, this is the rectangle rule.

For terms with time derivatives of the sought function, we use quadrature rules with Gaussian points at the nodes of the element:

$$\int_{V_k^n} \frac{\partial A}{\partial t} \delta A dV = \sum_{j=1}^{M} \frac{A_{J_2(k,j)}^{n+1} - A_{J_2(k,j)}^{n}}{M \Delta t_n} \delta A_{J_2(k,j)} V_k^n. \tag{2.9}$$

In the one-dimensional case, this formula corresponds the integral trapezoidal rule. Here, the time derivative is represented in difference form to compute the solution A_i^{n+1} at the new time level t_{n+1}. The variation of the solution is independent of time, so the superscript on the variation of the solution is omitted. Due to the used representation of the integrals of terms with time derivatives, they can be evaluated without inverting band mass matrices, since they are diagonal (so-called inconsistent mass matrices [5]).

Note that it was previously believed in the literature on the finite element method that inconsistent mass matrices lead to the instability of finite-element algorithms as applied to nonstationary problems, while consistent mass matrices obtained, for example, by integration using formulas with Gaussian points at the centers of elements, on the contrary, provide more stable computations. However, the theory of difference schemes revealed that the cause of instability is the same as in explicit central-difference schemes, namely, the presence of diffusion terms in the first differential approximations with negative viscosity coefficients. Accordingly, an additional explicit or implicit viscosity has to be introduced for stability. Numerous ways of introducing such stabilizing terms or approximations represent a separate problem that lies beyond the scope of this work (for more detail, see [6]).

The simplest and most efficient approach to solving the variational balance equation (2.1) is based on the explicit–implicit scheme

$$\frac{\partial}{\partial t}\int\limits_{V^n}\frac{A^{n+1}-A^n}{\Delta t_n}\delta A dV + \int\limits_{V^n}(\tilde{\mathbf{B}}^{n+1} - A^n\mathbf{\Omega}^n)\cdot\nabla^n\delta A dV = \int\limits_{V^n} f^{n+1}\delta A dV + \int\limits_{S_B^n}(\tilde{\mathbf{B}}_*^{n+1} - A^n\mathbf{\Omega}^n\cdot\mathbf{n}^n)\delta A dS, \tag{2.10}$$

where artificial viscosity is added to the flux terms $\mathbf{B}$ for stability:

$$\tilde{\mathbf{B}}^{n+1} = \mathbf{B}^{n+1} + \nu_{art}^n\mathbf{I}\cdot\nabla^n A^{n+1}. \tag{2.11}$$

Here, the additional stabilizing viscosity ν_{art}^n is introduced explicitly or implicitly (e.g., by applying noncentered approximations of the derivatives) according to the finite-element scheme used. Here and below, the unit tensor of the second rank is denoted by $\mathbf{I}$. Recall that the functions given on the boundary are marked with a star.

Note that Eq. (2.10) takes into account the natural boundary conditions (constraints on the boundary fluxes). Since the sought function A is given on the boundary portion $S_A^n \subset S^n$, its variation on S_A^n are equal to zero, so the boundary integral is taken only over the rest of the boundary $S_B^n = S^n \backslash S_A^n$. The superscript n is used, since the positions of the boundaries and the boundary conditions vary with time.

Substituting the spatial approximations of the derivatives and the integrals yields the following approximate representation of the variational equation:

$$\begin{gathered}\sum_{k=1}^{N_2}\left[\sum_{j=1}^{M_2}\left(\frac{A_{J_2(k,j)}^{n+1} - A_{J_2(k,j)}^n}{\Delta t_n} - f_{J_2(k,j)}^n\right)\frac{1}{M_2}\delta A_{J_2(k,j)}V_k^n\right] \\ + \sum_{k=1}^{N_2}\left[\left(\tilde{K}_k^n\sum_{j=1}^{M_2}\nabla_{kj}^n A_{J_2(k,j)}^{n+1}\right)\sum_{l=1}^{M_2}\nabla_{kj}^n\delta A_{J_2(k,j)}V_k^n\right] + \sum_{k=1}^{N_2}\left[L_k^n\sum_{l=1}^{M_2}\nabla_{kj}^n\delta A_{J_2(k,j)}V_k^n\right] \\ - \sum_{k=1}^{N_2}\left[\left(\sum_{j=1}^{M_2}\frac{1}{M_2}A_{J_2(k,j)}^n\right)\sum_{l=1}^{M}\nabla_{kj}^n\delta A_{J_2(k,j)}V_k^n\right] - \sum_{k=1}^{N_3}\left[\left(B_{*k}^{n+1} - \Omega_k^n\sum_{l=1}^{M_3}\frac{A_{J_3(k,l)}^n}{M_3}\right)\sum_{j=1}^{M_3}\frac{\delta A_{J_3(k,j)}}{M_3}S_k^n\right] = 0.\end{gathered} \tag{2.12}$$

Here, $\tilde{K}_k^n = (K + \nu_{art}\mathbf{I})_k^n$. In computer codes, sums correspond to loops over cells, which involve loops over local nodes in cells. The discrete variations of the sought functions at nodes do not take values (the basic property of variations is that they are arbitrary), so that the sum of like terms of each discrete variation at the solution yields zero (the Euler lemma). That is, such sums are the equations that make up a linear algebraic system for the given problem. Like terms are combined in loops over cells, and the discrete variations indicate the component of the gradient (residual) from which the current multiplier of the given variation is summed. The algorithm for combining like terms coincides with the computation of a time step in explicit two-level schemes. The calculation of the gradients and their homogeneous parts can be briefly written as the formulas

$$\begin{gathered}\mathbf{g} = C\mathbf{y} - \mathbf{d} = G_i^n\frac{A_i^{n+1} - A_i^n}{\Delta t_n} - P_i^n - Q_i^{n+1}, \\ \mathbf{g}_0 = C\mathbf{y} = G_i^n\frac{A_i^{n+1}}{\Delta t_n} - Q_i^{n+1},\end{gathered} \tag{2.13}$$

where the vector of unknowns $\mathbf{y} = \{A_i^{n+1}\}_{i=1}^{N}$ consists of the desired node values of A, while the multiplier G_i^n and the terms P_i^n and Q_i^{n+1} are given by the formulas

$$G_i^n = \sum_{k=1}^{N_2} V_k^n M_2^{-1} \sum_{j=1}^{M_2} \tilde{H}(i - J_2(k,j)),$$

$$\begin{gathered} P_i^n = \sum_{k=1}^{N_2} V_k^n M_2^{-1} \sum_{j=1}^{M_2} \left[f^n_{J_2(k,j)} \tilde{H}(i - J_2(k,j)) \right] - \sum_{k=1}^{N_2} V_k^n L_k^n \sum_{l=1}^{M_2} \left[\nabla^n_{kj} \tilde{H}(i - J_2(k,j)) \right] \\ + \sum_{k=1}^{N_2} V_k^n \frac{1}{M_2} \left(\sum_{l=1}^{M_2} A^n_{J_2(k,l)} \right) \sum_{j=1}^{M} \left[\nabla^n_{kj} \tilde{H}(i - J_2(k,j)) \right] + \sum_{k=1}^{N_3} S_k^n \left(B^{n+1}_{*k} - \Omega_k^n \sum_{l=1}^{M_3} \frac{A^n_{J_3(k,l)}}{M_3} \right) \frac{1}{M_3} \sum_{j=1}^{M_3} \left[\tilde{H}(i - J_3(k,j)) \right], \end{gathered} \tag{2.14}$$

$$Q_i^n = -\sum_{k=1}^{N_2} V_k^n (K + \nu_{art} I)_k^n \left(\sum_{l=1}^{M_2} \nabla^n_{kl} A^{n+1}_{J(k,l)} \right) \sum_{l=1}^{M_2} \left[\nabla^n_{kj} \tilde{H}(i - J_2(k,j)) \right],$$

where the function $\tilde{H}$ is equal to unity if the argument is zero and is equal to zero otherwise.

The main boundary conditions, i.e., the sought function given at the boundary nodes, are taken into account in specifying an initial approximation and, then, at the end of every iteration. For this purpose, the residuals at the nodes with main boundary conditions are set to zero, i.e., the correction terms to the given boundary values of the sought function are set to zero. The natural boundary conditions (constraints on boundary fluxes) are taken into account in the sense of a weak solution in the variational equation. Thus, the boundary conditions are easy to take into account as compared with many other numerical methods.

In traditional matrix methods used to implement algorithms for computing the coefficients of the matrix C, one would have to collect like terms of common factors—the products $A_i^{n+1}\delta A_j$. Moreover, the matrix coefficients would require considerable storage space, which would grow nonlinearly with the number of space variables and the number of discrete unknowns.

In the considered version, the matrix C is not necessary for determining residuals and the computational algorithm becomes simpler, more visual, and more efficient in terms of storage space and performance. Programming and debugging simplify considerably, since the code is obtained, in fact, by writing the original equations in discrete form. The algorithms are easy to modify and adapt to (at first glance) completely different continuum mechanics problems.

The application of the considered iterative processes not only leads to significant savings of machine resources, but also (which is especially valuable) reduces the effort related to the development of algorithms and debugging, which is inevitable for complicated algorithms, and makes it possible to focus on more interesting work, namely, the study of solutions for a wide range of continuum mechanics problems.

2.3. Allowance for additional conditions (constraints). In shock-capturing computations (see [7]), any additional conditions (constraints), for example, contact ones, breakage/consolidation, phase transitions, incompressibility, and other conditions are easily taken into account by modifying the variational equations with the use of Lagrange multipliers or the penalty function method (in various versions). In such modifications, there is no need to use the conjugate gradient method. They are well-known and well-elaborated and are widely used in variational formulations of continuum mechanics problems.

2.4. Splitting with respect to physical processes. Note that, at every time step, the system of continuum mechanics equations is generally split with respect to physical processes, and a complete solution at a new time level is obtained by sequentially solving auxiliary boundary value problems for each balance equation of the problem (e.g., for momentum, continuity, energy, tracer balance, and other equations).

Rather frequently, there is no need to solve the auxiliary boundary value problems to high accuracy. In this case, the conjugate gradient method in the implicit schemes can be replaced with simpler iterative time marching algorithms by artificially inverting elliptic equations into parabolic ones, which can be integrated by explicit schemes. For this purpose, a nonstationary term with derivatives with respect to fictitious time is added to the original elliptic equation. Examples of such auxiliary problems are boundary value problems for equations for controlling moving adaptive grids (see [8]). By time marching, an approximate steady-state solution is obtained after a preset fixed number of iterations (determined by enumeration).

The validity of this simplification of the iterative algorithms is checked by computations. The simplified algorithms are valid if an improvement in the accuracy of solutions of the auxiliary problems nearly does not influence the solution of the main problem.

2.5. Conservativeness of the algorithms. This property is ensured by the variational Galerkin–Petrov form of the conservation laws (2.10) and by a finite-element approximation of the solution. Local conservativeness or the balance of fluxes between the nodal volumes is checked trivially and follows from the variational form of the equations and the fact that the derivatives of a constant are zero. Global conservativeness is provided according to boundary conditions.

2.6. Specification of artificial viscosity. The time-stepping schemes (in physical or fictitious time at iteration) are entirely similar to two-level central-difference schemes. Since convection is calculated using an explicit central-difference scheme, its stability is ensured by introducing artificial viscosity. From the variety of recipes available for this purpose [6], after performing numerous tests, we chose the stabilized upwind Petrov–Galerkin (SUPG) scheme [9, 10]. However, this scheme was used in a simplified form that corresponds to a central finite difference scheme with artificial viscosity. According to the SUPG scheme, the artificial viscosity was defined so that the contribution made by the diffusion terms on the boundaries between the nodal volumes was balanced in the norm by the contribution of the other terms from the homogeneous part of the equation. Note that "on the boundaries between the nodal volumes" means at Gaussian points of the quadrature rules used to integrate the variational equation. These Gaussian points are placed at the centers of finite elements. This simplified version of the stabilized Galerkin–Petrov scheme is called the balancing-viscosity scheme. The formulas for computing the artificial viscosity ν_{art} are given by

$$[\nu_{art}]_k = \alpha_k \frac{d_{1\mathrm{k}}}{d_{2\mathrm{k}}}$$

(see [11]). Here, k is the finite-element index, the time level index is omitted from all variables to simplify the notation, and the parameter α_k is defined as $\alpha_k = 0.5$ if $[\nabla \cdot \mathbf{u}]_k \Delta t < 0.1$ and $\alpha_k = 1.0$ otherwise. The time step Δt is determined by the stability condition (see below). The coefficients $d_{1k}^{(j)}$ and $d_{2k}^{(j)}$ ($j = 1, 2, 3$) are given by the formulas

$$(d_{1k})^2 = ([\nabla \cdot (A\Omega)]_k)^2, \quad (d_{2\mathrm{k}})^2 = \sum_{l=1}^{m} ([\nabla A]_k \cdot \nabla_{kl})^2.$$

For spatial differentiation coefficients ∇_{kl}, the property $\sum_{l=1}^{m} \nabla_{kl} = 0$ follows from the fact that the derivative of a constant is zero. This property ensures that the exchange of the preserved quantity between the nodal volumes is conservative. In zones of high rarefaction degree caused by the flow acceleration for $[\nabla \cdot \mathbf{u}]_k \Delta t \geq 0.1$, the balancing viscosity coefficient is doubled, which is necessary for the stability of the computations at long times.

2.7. Correction of physical viscosity. To improve the accuracy of the solution in boundary layers, the physical viscosity k is corrected (reduced with growing artificial viscosity) by using the exponential adjustment method, which was proposed by Samarskii in 1960s and was developed in [12]. The simplest correction has the form

$$\tilde{k} = \frac{k^2}{k + \nu_{art}}.$$

2.8. Elimination of small-scale spurious perturbations. Since all elements of the algorithm are justified only as applied to simple test problems in a loose manner, the solutions of problems based on complete equations exhibit spurious oscillations with a wavelength proportional to the spatial mesh size. Such nonphysical (saw-toothed) perturbations can be immediately detected by changes in the sign of the second derivative in the checked coordinate direction at neighboring nodes belonging to a grid edge. The relative orientations of the edge and the coordinate direction are of no matter. Such perturbations do not violate the stability of the method (do not grow), but deteriorate the plots of the solution (the contour line patterns become nonsmooth). These perturbations are eliminated by applying the simplest conservative coordinatewise smoothing procedure, which is used at a few nodes at the end of every step.

2.9. Determination of second derivatives. In the case of piecewise linear approximations of the solution, the second derivatives A_{xx} of some solution component w with respect to x are determined as follows. The obvious variational equation for the second derivatives,

$$\int_V (A_{xx} - \partial^2 A/\partial x^2)\delta A_{xx} dV = 0,$$

is integrated by parts to obtain

$$\int_V A_{xx}\delta A_{xx} dV + \int_V \partial A/\partial x\, \partial A_{xx}/\partial x dV = \int_S \partial A/\partial x\, \delta A_{xx} n_x dV,$$

where S is the boundary of the solution domain V. For simplicity, the right-hand side of the equation is set to zero, i.e., either the first or the second derivative with respect to x is assumed to vanish on the boundary. If Gaussian quadrature rules with Gaussian integration points placed at grid nodes are applied to the first integral on the left-hand side, then the second derivatives are explicitly determined by the variational equation, so there is no need to convert nondiagonal matrices. In this way, the second derivatives are quickly calculated using piecewise linear functions of the approximate solution. The second derivatives with respect to y and z are calculated in a similar fashion.

2.10. Stability conditions. For explicit–implicit schemes, the time step Δt is determined by the stability condition, which has the usual form (recall that an explicit time approximation is used only for convection with velocity $\mathbf{\Omega}_k$)

$$\Delta t \le \beta \min_k \left(\frac{h_k}{|\mathbf{\Omega}_k| + 10^{-6}}, \frac{10^{-1}\varepsilon_S}{\|\mathbf{e}_k\| + 10^{-6}} \right),$$

where h_k is the size of the kth finite element, β is a safety factor (usually equal to 0.3–0.6), $\mathbf{e}$ is the strain rate tensor, and ε_S is the strain corresponding to the yield point or the ultimate tensile strength. The convection constraint (the term $|\mathbf{\Omega}_k|$ in the denominator of the stability condition) is caused by the explicit approximation of convective fluxes. The accuracy constraint (second constraint) is necessary in solid mechanics problems. It expresses the requirement that the norm of the strain increment over a time step be small.

It is important to note that explicit–implicit schemes can successfully be applied to problems involving rapid processes, for which the time step constraint caused by the accuracy conditions nearly coincides with the time step constraints caused by the stability conditions in explicit schemes:

$$\Delta t \le \beta \min_k \left(\frac{h_k}{|\mathbf{\Omega}_k| + c + 10^{-6}} \right),$$

where c is the speed of sound (the velocity of propagation of small perturbations). In such problems, since the solution at the preceding time level is a good initial approximation to the solution at a new level, the number of iterations in the conjugate gradient method decreases sharply to two or three. In other words, implicit schemes with the conjugate gradient method automatically begin to perform as quickly as explicit schemes. Small numbers added to the denominators of the stability conditions prevent division by zero.

2.11. Difficulties disappearing in matrix-free algorithms. Note additional advantages of matrix-free iterations: (1) there is no need to store nonzero elements of the system's matrix and their locations; (2) there is no need to calculate elements of the system's matrix; and (3) there is no need to search for an optimal indexing of grid nodes ensuring that the band width of the system's matrix is minimal. These advantages are inherent in any matrix-free iterative methods (stationary iteration, the Gauss–Seidel method, relaxation methods, and others), but only conjugate gradient methods guarantee that the solution is determined in a finite number of iterations.

3. RESULTS

Below, typical problems are solved by applying explicit–implicit iterative algorithms based on the conjugate gradient method.

3.1. Supersonic ideal gas flows with shock waves on moving adaptive grids. Figure 1 shows the results obtained for the supersonic flow over a wedge (plane problem). The ratio of specific heats is 1.4, and the

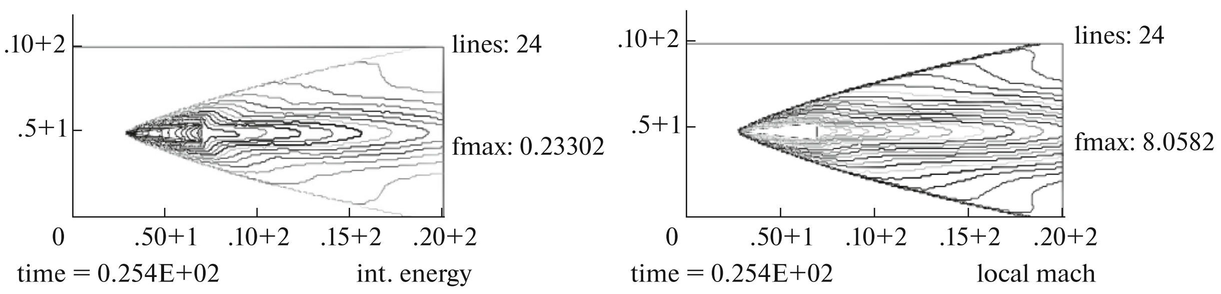

Fig. 1. Flow over a wedge at M = 8: internal energy and the local Mach number.

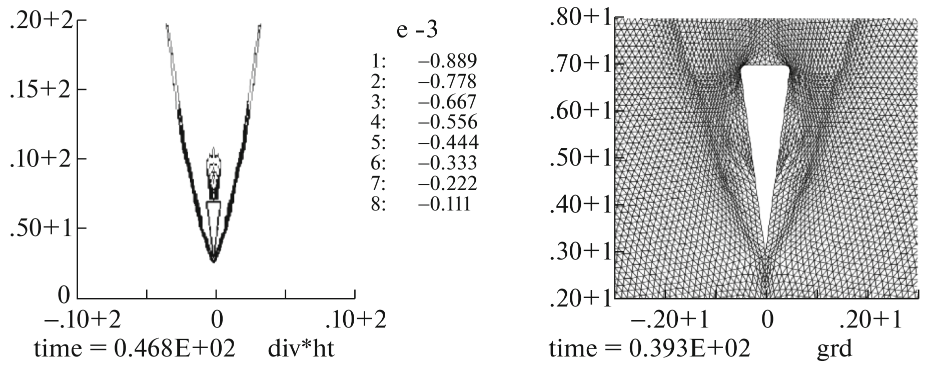

Fig. 2. Flow over a cone at M = 134: monitor function and a grid fragment.

free-stream Mach number is M = 8. For the supersonic flow over a cone (axisymmetric problem) at M = 134, Fig. 2 presents the monitor function $\tilde{T} = -\Delta t \nabla \cdot \mathbf{v}$, which controls mesh adaptation, and a fragment of an adaptive grid near the cone.

Of course, in the range of hypersonic flows (M > 8) the problem formulation has to be considerably extended by taking into account a variety of physicochemical effects in a high-temperature plasma. Here, we demonstrate only the potential applicability of the algorithm to problems of this type. The number M = 134 was random. To increase the Mach number, the temperature in the incoming flow was specified at random by a small number.

For the flow over a wedge, the wedge was specified by the method of overlapping meshes [11]. The computations were performed on a basic (bordering) structured mesh consisting of initially rectangular cells (elements) in a rectangular flow region. The wedge had its own (overlapping) mesh. The velocities were set to zero at nodes of the basic mesh covered with the wedge mesh. Under mesh adaptation, the covered nodes of the basic mesh did not move (the wedge was fixed). In shock-capturing computation, the variational formulation of the problem and the finite element approximation of the solution automatically approximated the necessary conditions on the boundaries of the overlapped domain—the equality of the pressure and temperature derivatives normal to the boundary.

In the case of boundary conditions of other type, for example, when we specify the functions themselves or mixed boundary conditions, they are easy to take into account as additional constraints via a modification of the variational balance equations by applying the penalty function method or Lagrange multipliers.

3.2. Collapse of a heavy viscous fluid column in a closed tank. Initially, a heavy viscous fluid (water) column in a closed cubic tank is at rest. Under downward gravity, the water begins to move and the column collapses. Under viscous friction forces, the water oscillations in the tank are damped and the fluid reaches a new equilibrium. Shock-capturing computations were performed with the use of a continuous marker function equal to 1 in water-filled cells and to 0 in empty cells. The marker function was calculated using the transport equation. The computations were performed in terms of velocity and pressure. Both

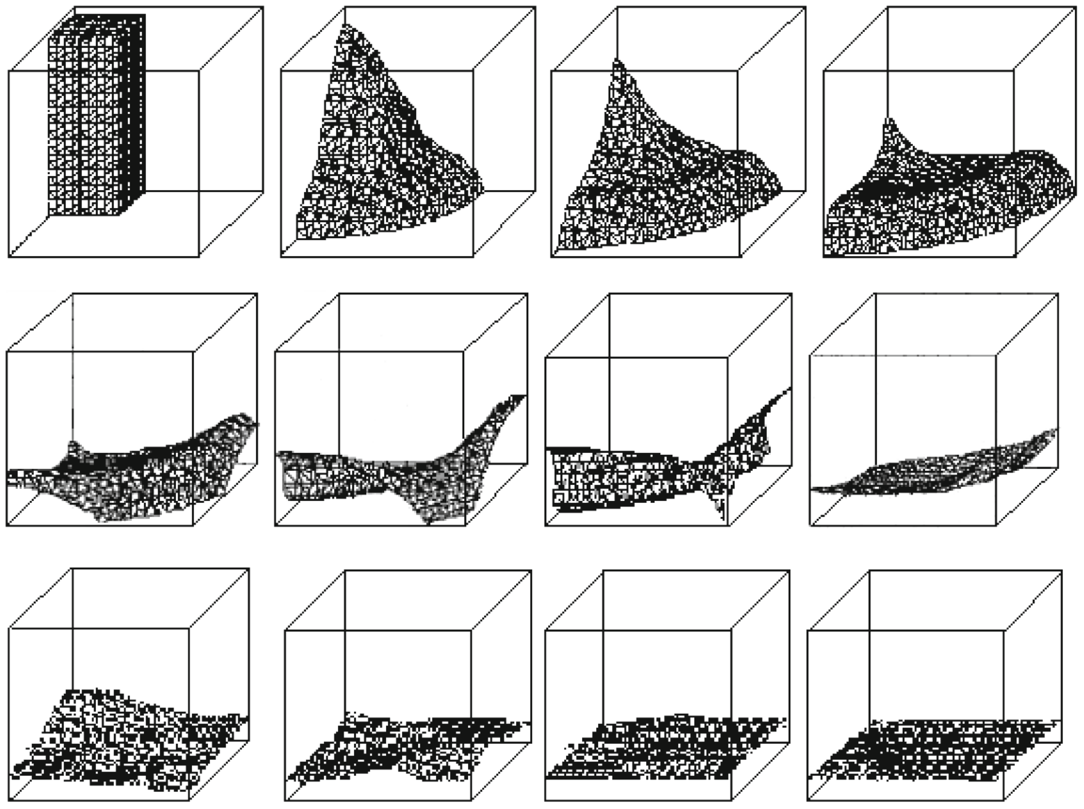
Fig. 3. Collapse of a heavy viscous column in a closed tank.

variables were determined at grid nodes, and a piecewise linear approximation of the solution was applied, so that the LBB condition was not satisfied, but this did not stop us from solving the problem. Conservativeness with respect to mass was not violated. In the case of a continuous marker function, this requires the monotonicity of the computational method and antidiffusion of the marker function near the free boundary.

3.3. Failure of an elastoplastic plate under heating and tension. Figure 4 shows a quarter of a flat elastoplastic specimen. The left and lower boundaries are the axes of symmetry. The entire upper boundary is free of external load. The right boundary is slowly shifted to the right, so that the specimen is stretched. The corner at the upper boundary leads to a concentration of stress and strain, and the failure begins from this corner. The evolution of crack propagation is shown with allowance for only elasticity (left panels), elasticity and plasticity (middle panels), and with additional allowance for local heating of the vertical zone from the corner (right panels). The failure patterns remain unchanged under mesh refinement. It can be seen that the failure patterns are completely different depending on the properties of the material and applied external heating. The computations were based on the shock-capturing approach, i.e., the intact and cracked plates were both computed by applying a unified algorithm. The elastic moduli of the material and the yield point depended on the degree of failure of the material. The degree of failure was determined using a kinetic equation under the failure condition [13], which was switched on after the maximum tensile strain reached a certain limit.

3.4. Stamping of an aluminum cup. Figure 5 shows the numerical results obtained for the stamping of an aluminum cup. The lower rigid die was at rest. The upper die moved downward, converting a circular aluminum plate into a cup. The duration of the process was much longer than the time required for elastic waves to travel along the radius of the plate, so that the process was quasistatic. It was computed by applying an implicit scheme with the use of the matrix-free conjugate gradient method. The contact interaction was computed by the Lagrange multiplier method with the Lagrange multipliers being normal contact loads. The contact friction was assumed to be negligibly low.

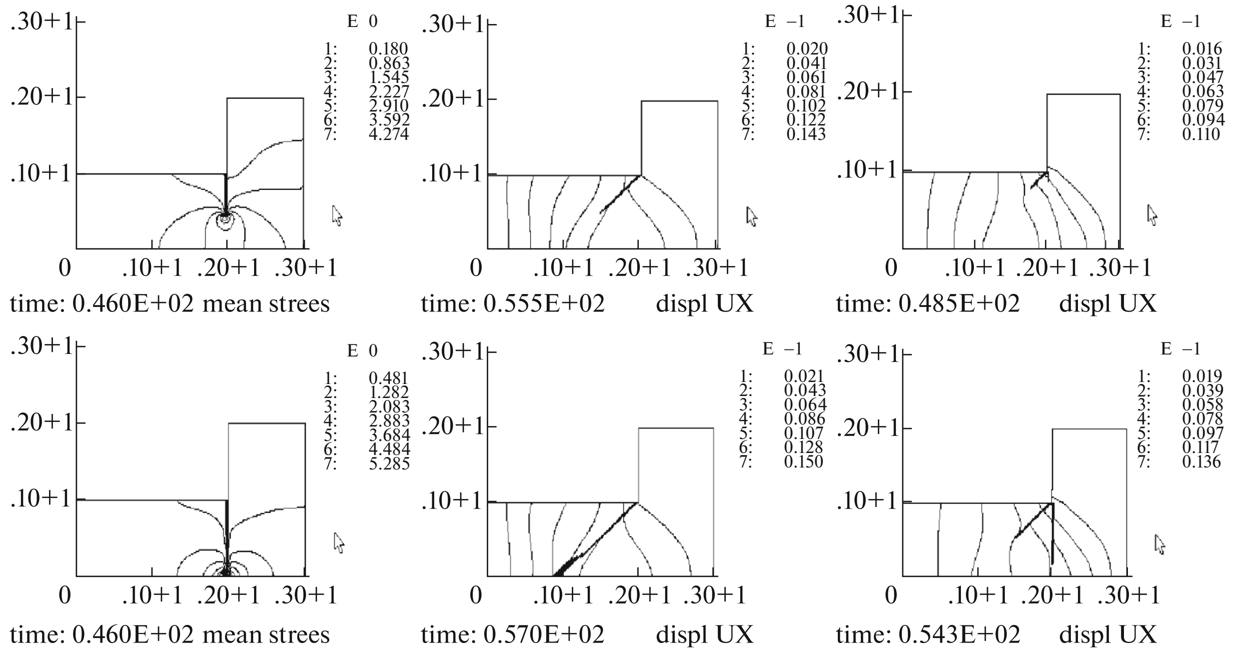

Fig. 4. Failure of an elastoplastic plate under tension and heating.

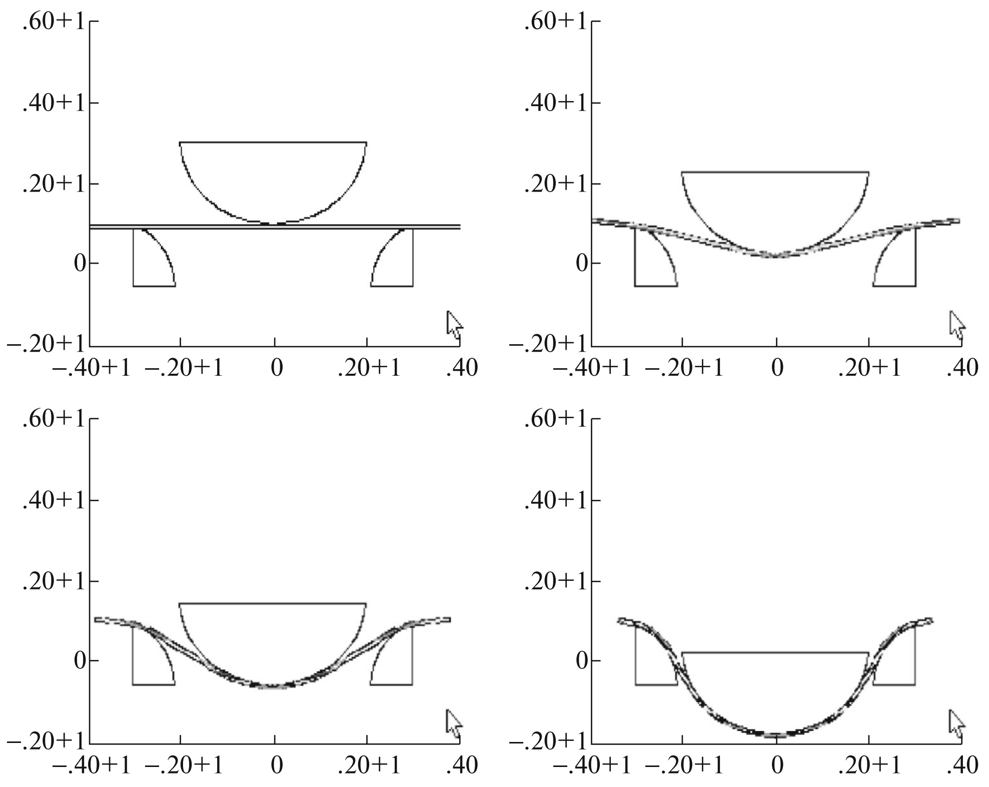

Fig. 5. Stamping an aluminum cup.

CONCLUSIONS

A detailed description of simple and low-cost effective algorithms for the implementation of explicitly implicit schemes using conjugate gradient methods without matrices has been given. Typical examples of the application of the methods in implicit schemes were presented for gas dynamics problems (computation of an adaptive moving grid by an implicit scheme), mechanics of a viscous incompressible fluid (implicit scheme for computation of viscous effects and incompressibility) and mechanics of solids (implicit scheme for calculating contact interaction and destruction of elastoplastic bodies). The methods used in the calculations are well known. Here, it was shown that their matrix-free implementation leads to very simple and effective algorithms.

REFERENCES

1. M. R. Hestenes and E. Stiefel, "Method of conjugate gradients for solving linear systems," J. Res. Natl. Bur. Stand. **49**, 409–436 (1952).
2. E. Polak, *Computational Methods in Optimization: A Unified Approach* (Academic, New York, 1971).
3. J. H. Wilkinson and C. Reinsch, *Handbook for Automatic Computation*, Vol. 2: *Linear Algebra* (Springer-Verlag, Berlin, 1971).
4. N. G. Burago, "Formulation of basic equations of continuum mechanics in moving adaptive coordinates," in *Numerical Methods in Computational Solid Mechanics*, Ed. by G. I. Pshenichnov (Vychisl. Tsentr Akad. Nauk SSSR, Moscow, 1984) [in Russian].
5. G. Strang, and G. Fix, *An Analysis of the Finite Element Method* (Prentice-Hall, Englewood Cliffs, 1973).
6. A. G. Kulikovskii, N. V. Pogorelov, and A. Yu. Semenov, *Mathematical Aspects of Numerical Solution of Hyperbolic Systems* (Fizmatlit, Moscow, 2001; Chapman and Hall/CRC, London, 2001).
7. N. G. Burago and V. N. Kukudzhanov, "A review of contact algorithms," Mech. Solids. **40** (1), 35–71 (2005).
8. V. D. Liseikin, "A survey of methods for constructing structured adaptive grids," Comput. Math. Math. Phys. **36** (1), 1–32 (1996).
9. A. N. Brooks and T. J. R. Hughes, "Streamline upwind Petrov–Galerkin formulation for convection dominated flows," Comput. Methods Appl. Mech. Eng. **32**, 199–259 (1982).
10. G. J. Le Beau, S. E. Ray, S. K. Aliabadi, and T.-E. Tezduyar, "SUPG finite element computation of compressible flows with the entropy and conservation variables formulations," Comput. Methods Appl. Mech. Eng. **104**, 397–422 (1993).
11. N. G. Burago, I. S. Nikitin, and V. L. Yakushev, "Hybrid numerical method with adaptive overlapping meshes for solving nonstationary problems in continuum mechanics," Comput. Math. Math. Phys. **56** (6), 1065–1074 (2016).
12. E. P. Doolan, J. J. H. Miller, and W. H. A. Schilders, *Uniform Numerical Methods for Problems with Initial and Boundary Layers* (Boole, Dublin, 1980).
13. N. G. Burago, A. I. Glushko, and A. N. Kovshov, "Thermodynamic method for constructing constitutive relations for models of continuous media," Mech. Solids **35** (6), 1–9 (2000).

Translated by I. Ruzanova