A batch of applied programs for numerical solution of convection-diffusion boundary-value problem

Kireev I.V., Pyataev S.F., Shaidurov V.V.

Table of Contents

A batch of applied programs for numerical solution of convection-diffusion boundary-value problem

Kireev I.V., Pyataev S.F., Shaidurov V.V.

A batch of applied programs for numerical solution of convection-diffusion boundary-value problem

Kireev I.V., Pyataev S.F., Shaidurov V.V.

Introduction

The work consists in development of an economical algorithm based on the classic variant of finite element method and intended for numerical solution of convection-diffusion boundary-value problem

$$-\varepsilon\triangle u + b_1\frac{\partial u}{\partial x} + b_2\frac{\partial u}{\partial y} = f \quad \text{in} \quad \Omega, \tag{1}$$

$$u = g \quad \text{on} \quad \Gamma. \tag{2}$$

Here two-dimensional domain Ω is limited by piecewise smooth boundary Γ; ε is a small positive number; b_1, b_2, f, g are smooth enough functions.

A good adaptation to the conditions of this problem is required from the algorithm, which would ensure high-accurate solution of boundary-value problem under linear approximation of the function $u(x, y)$ on each finite element. This means that an automatic division of the initial domain into finite elements oriented along the characteristics should be anticipated in the algorithm, and the requirement of economy indispensably leads to the use of the technology of embedded grids.

The idea of this algorithm is as follows: for construction of a new grid it is sufficient to analyze the behavior of piecewise linear and Hermitian cubic interpolations of the approximate solution obtained within the framework of standard finite element approach, and on the basis of this analysis to construct a partition of edges of finite elements, which automatically leads to

construction of a new embedded grid accounting for more subtle peculiarities of the desired solution.

For realization of this idea, it is necessary to have an algorithm of determination of partial derivatives of the function $u(x, y)$ from its given node values. From a number of algorithms of determination of partial derivatives we have chosen a method described below, which, in our opinion, most organically matches this class of boundary-value problems. Unfortunately, theoretical substantiation of this statement is very problematic, but the approach being proposed has made a good showing in a great number of numerical experiments.

The testing of algorithms and programs has been performed for a simpler boundary- value problem; it was assumed that the domain $\Omega = [0,1] \times [0,1]$ is unit square, $b_1 \equiv 1$, $b_2 \equiv 0$, and $f \equiv c$ is constant, i.e., the following equation was considered

$$-\varepsilon \triangle u + \frac{\partial u}{\partial x} = c \quad \text{in} \quad (0,1) \times (0,1).$$

It is easy to verify that this equation admits solutions of the form

$$u(x, y) = \left(c_1 e^{\lambda_1 (x-1)} + c_2 e^{\lambda_2 x}\right) \sin n\pi y + ay + b + cx,$$

where a, b, c_1, c_2 are certain constants, and

$$\lambda_1 = \frac{1}{2\varepsilon}(1 + \sqrt{1 + (2n\varepsilon)^2}) > 0, \quad \lambda_2 = \frac{1}{2\varepsilon}(1 - \sqrt{1 + (2n\varepsilon)^2}) < 0,$$

at that $\lambda_1 \cong \varepsilon^{-1} + n^2\varepsilon$ and $\lambda_2 \cong -n^2\varepsilon$ under $\varepsilon \ll 1$.

If we are interested in solutions which do not depend on y, then, as it is easy to show, the solution of the equation is of the form

$$u(x) = c_1 \exp \frac{x-1}{\varepsilon} + c_2 + cx.$$

Just in this class of functions the debugging and testing of the algorithm of edge division have been carried out.

1 An algorithm of determination of partial derivatives

Let approximate the partial derivatives of a function $u(x, y)$ determined by numerical values u_J in nodes $M_J(x_J, y_J) : u(x_J, y_J) = u_J$ in the vicinity of the point M_0 as a linear combination

$$\frac{\partial u(x, y)}{\partial x} = u^0_x + (x - x_0)u^0_{xx} + (y - y_0)u^0_{xy},$$

$$\frac{\partial u(x,y)}{\partial y} = u_y^0 + (x - x_0)u_{xy}^0 + (y - y_0)u_{yy}^0,$$

where $u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0$ are certain unknown constants to be determined for each node of the grid.

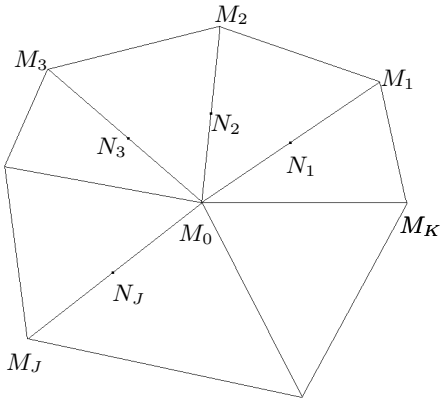

Fig. 1:
The nondegenerate case of d_{M_0}.

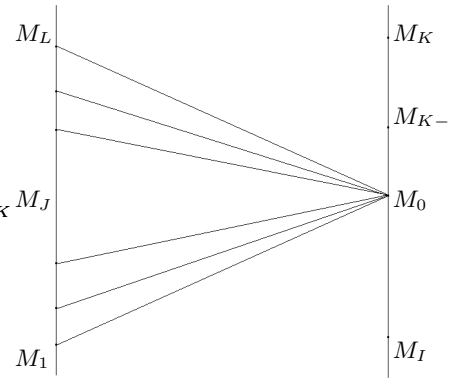

Fig. 2:
The degenerate case of d_{M_0}.

Then the central difference $u_J - u_0$ approximates the derivative of the function

$$u_J(t) = u(x_0 + t(x - x_0), y_0 + t(y - y_0)),\ t \in [0,1]$$

in the point N_J $(t = 0.5)$ with error $O(h_J^3)$, where

$$h_J = \sqrt{(x_J - x_0)^2 + (y_J - y_0)^2},$$

$u_J(0) = u_0$, and $u_J(1) = u_J$. Therefore for smooth enough function $u(x,y)$ the following relations should be valid:

$$d_J(u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0)/h_J^3 = O(1)$$

where

$$\begin{aligned} &d_J(u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0) = (u_J - u_0) \\ &- (x_J - x_0)[u_x^0 + 0.5(x_J - x_0)u_{xx}^0 + 0.5(y_J - y_0)u_{xy}^0] \\ &- (y_J - y_0)[u_y^0 + 0.5(x_J - x_0)u_{xy}^0 + 0.5(y_J - y_0)u_{yy}^0]. \end{aligned}$$

Grouping together neighbouring to M_0 nodes M_1, $M_2, \ldots, M_J, \ldots, M_K$, (see Fig. 1) construct the functional

$$d_{M_0}(u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0) = \sum_{J=1}^{J=K} \{d_J(u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0)\}^2 / h_J^6.$$

It is easy to verify that the components u_x^0, u_y^0 of solution of the least square problem

$$d_{M_0}(u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0) \underset{u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0}{\longrightarrow} \inf$$

give an approximation to partial derivatives

$$\frac{\partial u(x_0, y_0)}{\partial x}, \quad \frac{\partial u(x_0, y_0)}{\partial y}$$

of a smooth enough function $u(x, y)$ to within $\max\{h_J^2\}$. At that, total number of nodes M_J necessary for calculation of partial derivatives in the point M_0 must be not less than 5 ($K \geq 5$).

However, arbitrary sequences of points close to M_0 cannot be used for such procedure. So, for instance, a sequence of points similar to that shown in Fig. 2 gives a functional d_{M_0} degenerate with respect to u_x^0, u_y^0, u_{xx}^0, u_{xy}^0, u_{yy}^0, whose minimization problem has infinite number of solutions. Therefore, if in the process of calculation it appears that quadratic functional d_{M_0} generates a linear system of algebraic equations with singular matrix, then additional nodes M_J immediately neighbouring the node M_0 are consequently taken into consideration, till the functional d_{M_0} will become non-degenerate. The highest accuracy of the derivatives calculated in such a way is reached in internal points of the domain.

2 Construction of a sequence of embedded grids

The described above algorithm of determination of partial derivatives in vertices of finite elements from calculated node values of numerical solution $u(x, y)$ has been used for construction of a sequence of embedded grids.

A new grid was constructed on the basis of analysis of behaviour on each edge of the initial grid of both linear and Hermitian cubic interpolations of function $u(x, y)$ constructed from node values of function u and values of partial derivatives u_x, u_y calculated in the nodes of the grid.

An edge was not divided, if on the edge the module of maximal difference of values between linear and cubic splines did not exceed ε_a. Otherwise, a

new point was chosen inside the edge, proceeding from the following reasonings.

Denote by M'_J the desired point of division of the edge M_0M_J. Then, as it is shown in Fig. 3, on the edge M_0M_J a piecewise linear approximation of the function $u(x, y)$ appears; minimizing in some norm the residual between the latter and cubic approximations, we obtain an algorithm of construction of the point M'_J of the edge M_0M_J.

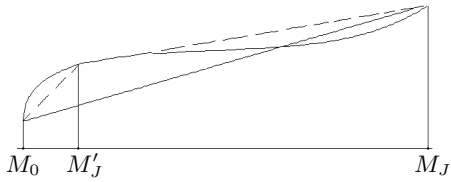

Fig. 3:
A piecewise linear approximation $u(x, y)$ on the edge M_0M_J.

Let give more details. Denote by $u(t)$ the cubic spline for the edge $[M_0, M_J]$; $t \in [0, 1]$ and the values

$$u(0) = u_0,\ u(1) = u_J,\ \frac{du}{dt}(0) = u'_0,\ \frac{du}{dt}(1) = u'_J$$

are given. Then

$$u(t) = a_{00} + a_{01}t + a_{02}t^2 + a_{03}t^3,$$
$$u(t) = a_{10} + a_{11}(1-t) + a_{12}(1-t)^2 + a_{13}(1-t)^3,$$

where

$$\begin{aligned}
a_{00} &= u_0; \quad a_{01} = u'_0;\\
a_{02} &= 3(u_1 - u_0) - 2u'_0 - u'_1;\\
a_{03} &= -2(u_1 - u_0) + u'_0 + u'_1;\\
a_{10} &= u_1; \quad a_{11} = -u'_1;\\
a_{12} &= -3(u_1 - u_0) + u'_0 + 2u'_1;\\
a_{13} &= 2(u_1 - u_0) - u'_0 - u'_1.
\end{aligned}$$

Let $v_0(t)$ and $v_1(t)$ be linear approximations of the function $u(t)$ on the intervals $[M_0, M'_J]$ and $[M'_J, M_J]$, where $t \in [0, \tau]$ and $t \in [\tau, 1]$, respectively;

$0 < \tau < 1$. Then the functional

$$\Phi_{L_2} = \int_0^{\tau} \big(u(t) - v_0(t)\big)^2 dt + \int_{\tau}^{1} \big(u(t) - v_1(t)\big)^2 dt$$

gives the square of norm of residual between $u(t)$ and its piecewise linear approximation $v_0(t)$, $v_1(t)$ in the space $L_2[0,1]$. Direct computations give the following expression for the functional:

$$\begin{aligned}\Phi_{L_2} &= (7a_{02}^2 + 21a_{02}a_{03}\tau + 16a_{03}^2\tau^2)\tau^5 \\ &\quad + (7a_{12}^2 + 21a_{12}a_{13}(1-\tau) + 16a_{13}^2(1-\tau)^2)(1-\tau)^5\end{aligned}$$

where the numbers a_{ij} are defined above. Minimizing this functional with respect to $\tau \in (0,1)$, we determine the coordinates of the point M_J' which is new node for the new grid; for this purpose it is necessary to solve an equation of the fifth power with respect to τ.

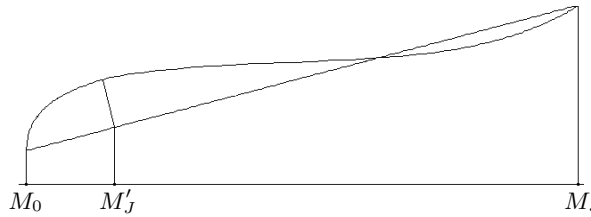

Fig. 4: The distance between graphs
of the cubic polynomial and its linear interpolation.

For numerical solution of algebraic equation the Newton method was used, and as an initial approximation the point of the edge $[M_0, M_J]$ was taken, in which the distance between the graphs of the cubic polynomial and its linear interpolation is maximal, as shown in Fig. 4. As a rule, this approximation is rather good, and for correction of it with a reasonable accuracy it is sufficient to make only several iterations of the Newton method.

Besides Φ_{L_2}, other functionals have been considered. So, for instance, a number of functionals have been considered which approximate residual functional from $C[0,1]$. However, test computations have shown that the results differ insignificantly, but the time of computation when constructing the embedded grid increases greatly. Apparently, this is connected with the fact that the node values u_J themselves are results of computations and have the accuracy of the order $\max\limits_J\{h_J^2\}$ where h_J is the length of $J-$th edge.

3 Program realization of the algorithm

A complex of programs in language **C** for numerical solution of convection-diffusion boundary-value problem (1)–(2) has been designed on the basis of the algorithm described above.

For numerical solution of convection-diffusion boundary-value problem a scheme with the second order of accuracy has been used, which generates a system of equations with M-matrix, satisfying discrete maximum principle.

The solution of the obtained system of linear algebraic equations has been carried out by iterative Gauss-Seidel procedure under special ordering of equations and unknowns. The number of iterations on each embedded grid was fixed and did not exceed 15.

In Fig. 5 – 20 some results of operation of the procedure of construction of embedded grid for solution of boundary-value problem under $x \in [0,1]$, $y \in [0,1]$ and $u(x,0) = u(x,1) = u(0,y) = u(1,y) = 0$, $\varepsilon = 10^{-3}$ are shown. The initial triangulation was generated by uniform division of the sides of the square into 8 equal intervals with subsequent diagonal division of each elementary square into triangles. An edge was not divided, if the module of the maximal on the edge difference between linear and cubic splines did not exceed $\varepsilon_a = 10^{-3}$; for solution of finite-dimensional problem on each of the grids the Seidel method with fixed number of iterations (=50) was used.

Fig. 5 – 8 show the character of arising grids under $\varepsilon_a = \varepsilon$. In figures 9 – 20 the information on the sequence of grids arising under $\varepsilon_a = 50\varepsilon$ is reflected;

Fig. 9 – 12 show general dynamics of the sequence of grids;

Fig. 13 – 14 show the changes of grid at two last steps in the square $[0.0,\ 0.125] \times [0.0,\ 0.125]$, eightfold enlarged;

Fig. 15 –16 show the changes of grid at two last steps in the square $[0.5,\ 0.625] \times [0.0,\ 0.125]$, eightfold enlarged;

Fig. 17 – 18 show the changes of grid at two last steps in the square $[0.875,\ 1.0] \times [0.0,\ 0.125]$, eightfold enlarged;

Fig. 19 – 20 show the changes of grid at two last steps in the square $[0.875,\ 1.0] \times [0.375,\ 0.5]$, eightfold enlarged.

In the course of joint researches in Augsburg Technical University series of test computations on different computers and under several operation systems has been carried out. Main objective of these computations was to estimate real time necessary for solving the considered boundary-value problem.

In our opinion, some of these results are rather interesting; they are represented below in the form of a table.

HOSTNAME	TYPE	MHz	MB	SYSTEM	1	2	3
MALAGA	R4000	150	96	IRIX	58.9	3.75	187.0
SEVILLA	R8000	75	512	IRIX64	42.8	3.88	171.0
MARBELLA	PENTIUM PRO	200	128	LINUX	20.2	2.40	56.7
ALCALA	PENTIUM II	266	128	LINUX	13.8	1.68	71.7
ZARAGOZA	ALPHA	500	128	D.UNIX	9.7	1.10	36.6
LACORUNA	ALPHA	533	256	LINUX	11.5	1.30	47.6
BURGOS	ALPHA	533	256	LINUX	11.7	1.28	47.9

Here the first five columns contain general information about the computers which were used in the computational experiment; this information has been kindly granted to us by professor U. Rüde.

C-version of the program contains four main parts:
(I) procedures of construction of initial triangulation for Ω;
(II) procedures of formation of the global system of linear algebraic equations;
(III) procedures realizing the iterative Gauss-Seidel process with special ordering of equations and unknowns;
(IV) procedures which construct embedded grid by above method using the solution from (III).

The tests have shown that at the beginning of the computational process the time of execution of each of I-IV parts of the **C**-program is proportional to N_{point} which is the number of points of the initial grid. Therefore, the time of execution of each part of the program in the course of the test computations was divided by $N_{point} = 10^6$ after the statistical processing of several numerical experiments. Here "time of execution" means the user time obtained by the command "**time**" of UNIX operation system.

The column 1 contains the time of computations for the parts I, II on a regular grid. The column 2 represents the time spent on realization of one full iteration in the Gauss-Seidel method when solving the system of linear algebraic equations. And, finally, the column 3 contains total time of computation of the stages IV and II of **C**-program.

The computations have been performed for the case when

$$\Omega = [0,1] \times [0,1],\ \varepsilon = 0.001,\ b_1 \equiv 1,\ b_2 \equiv 0,\ f \equiv 1,\ g \equiv 0.$$

One can see from this table that the efficiency of a computational complex strongly depends on both the parameters of computer and the type of operation system.

Fig. 5: Changes of grid at step 1; $\varepsilon = 10^{-3}$.

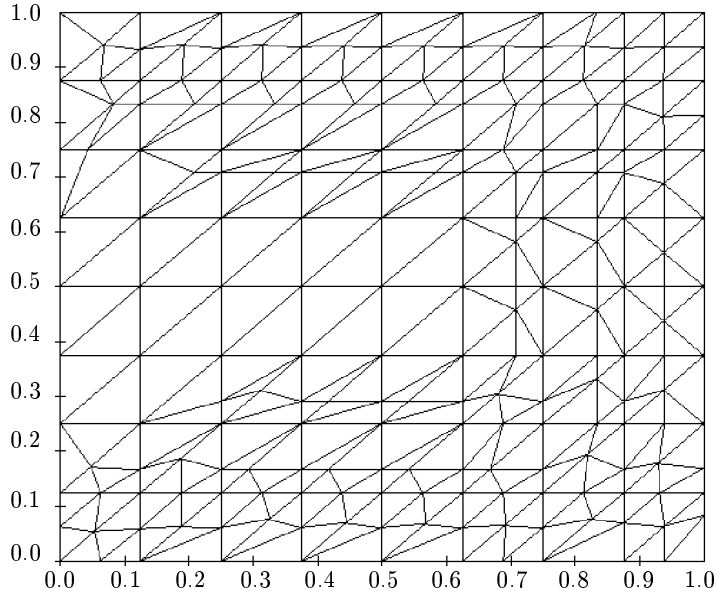

Fig. 6: Changes of grid at step 2; $\varepsilon = 10^{-3}$.

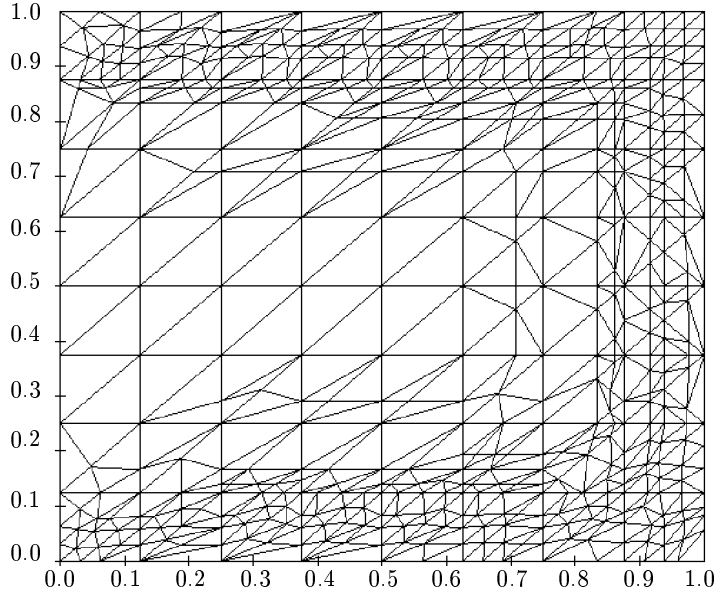

Fig. 7: Changes of grid at step 3; $\varepsilon = 10^{-3}$.

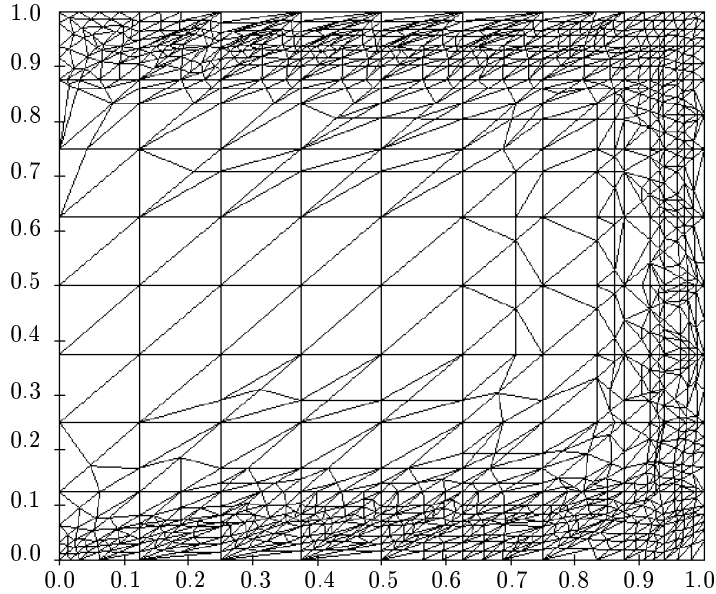

Fig. 8: Changes of grid at step 4; $\varepsilon = 10^{-3}$.

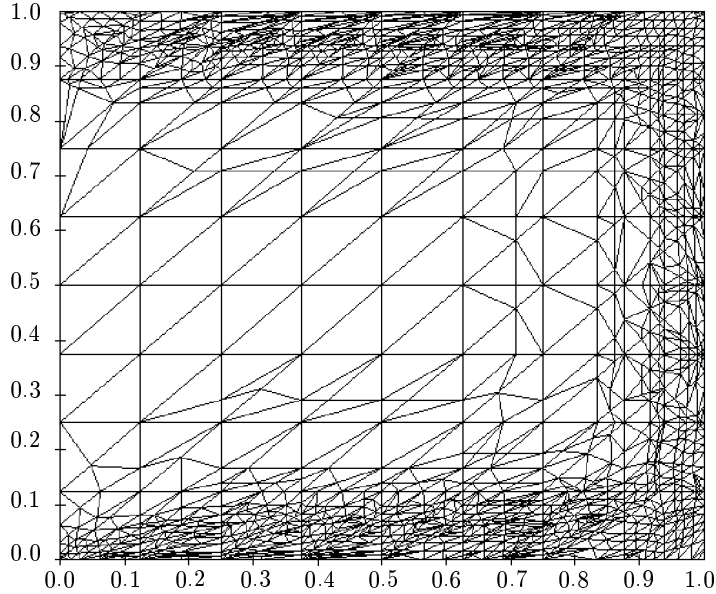

Fig. 9: Changes of grid at step 1; $\varepsilon = 0.05$.

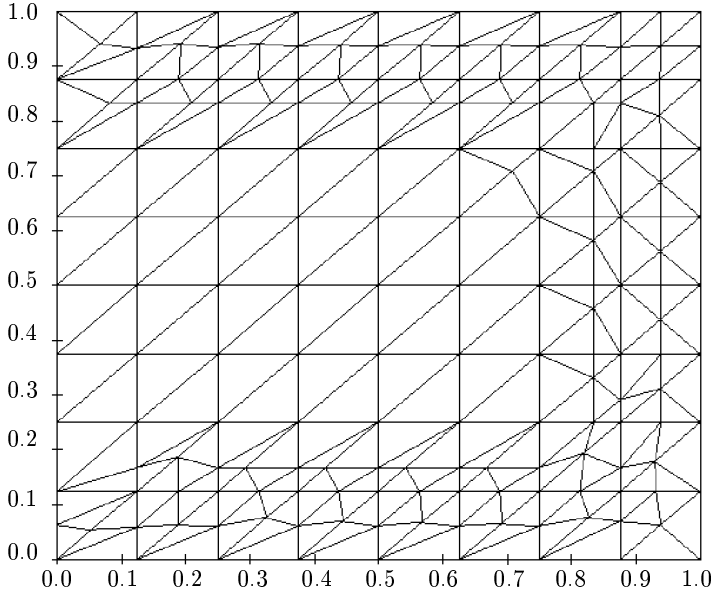

Fig. 10: Changes of grid at step 2; $\varepsilon = 0.05$.

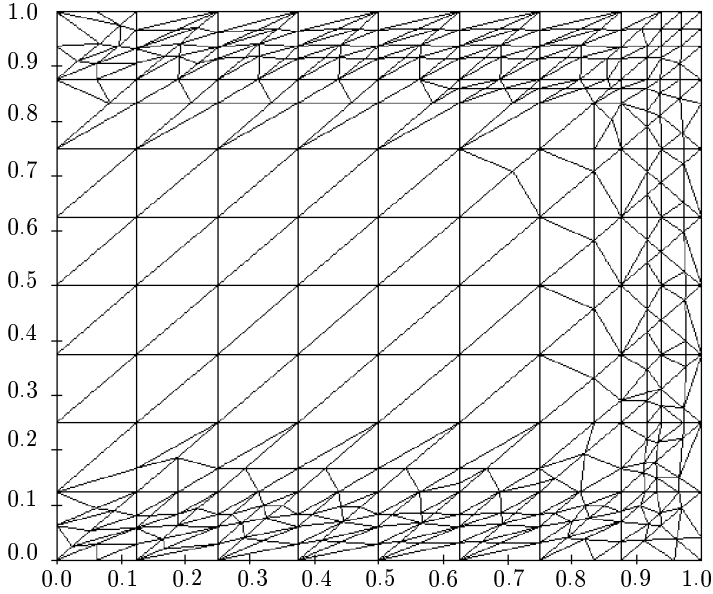

Fig. 11: Changes of grid at step 3; $\varepsilon = 0.05$.

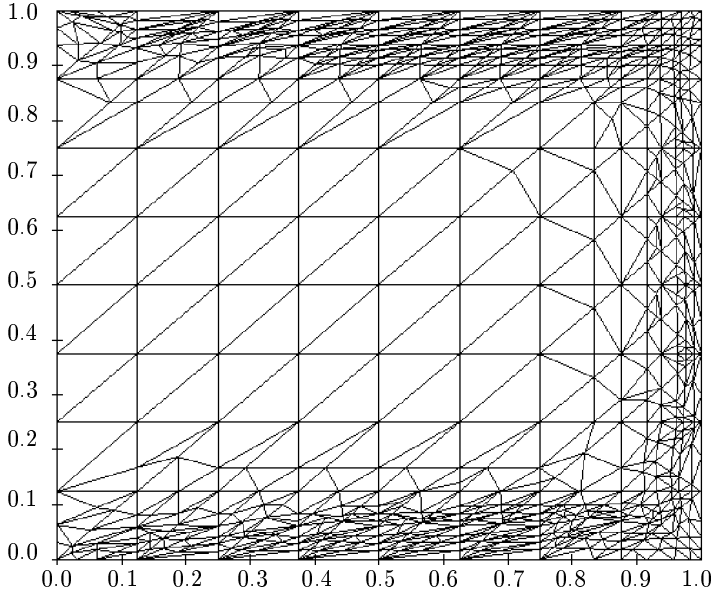

Fig. 12: Changes of grid at step 4; $\varepsilon = 0.05$.

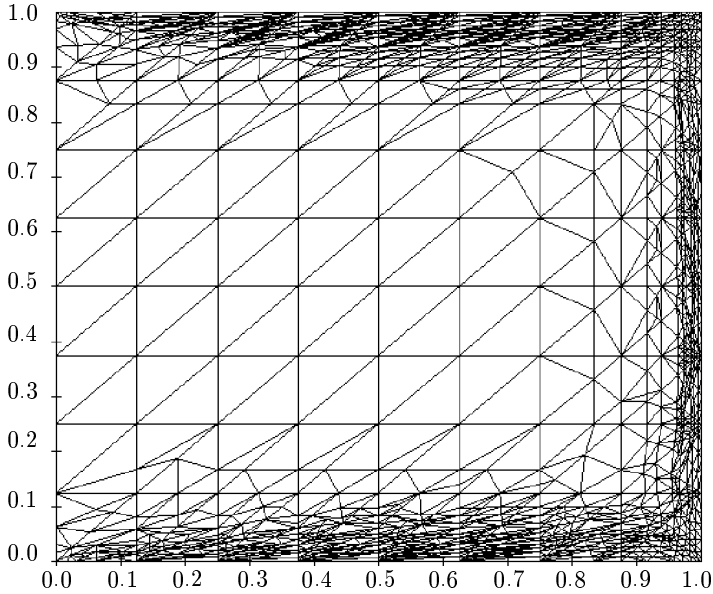

Fig. 13: Step 3; $\varepsilon = 0.05$ (vicinity of the origin).

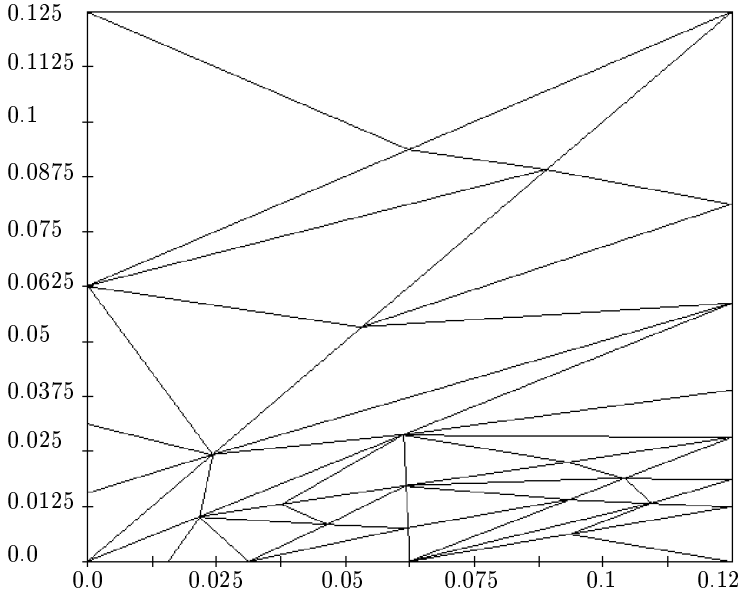

Fig. 14: Step 4; $\varepsilon = 0.05$ (vicinity of the origin).

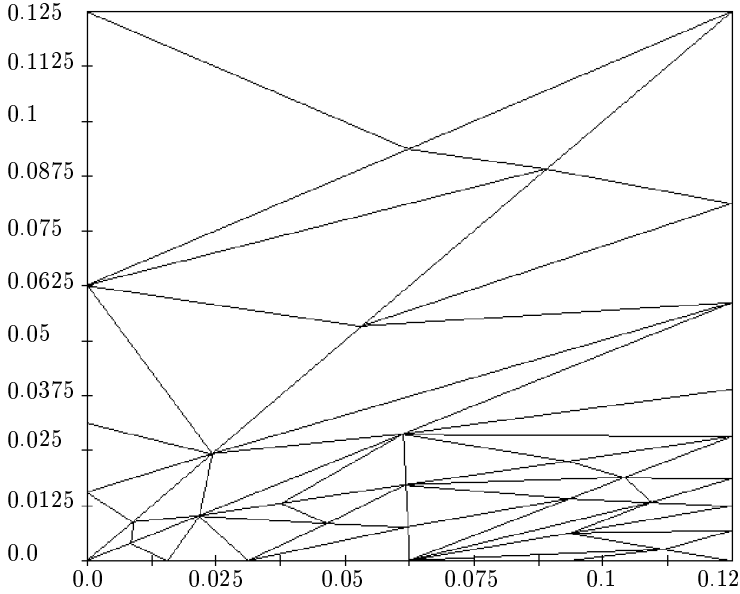

Fig. 15: Step 3; $\varepsilon = 0.05$ (at the bottom).

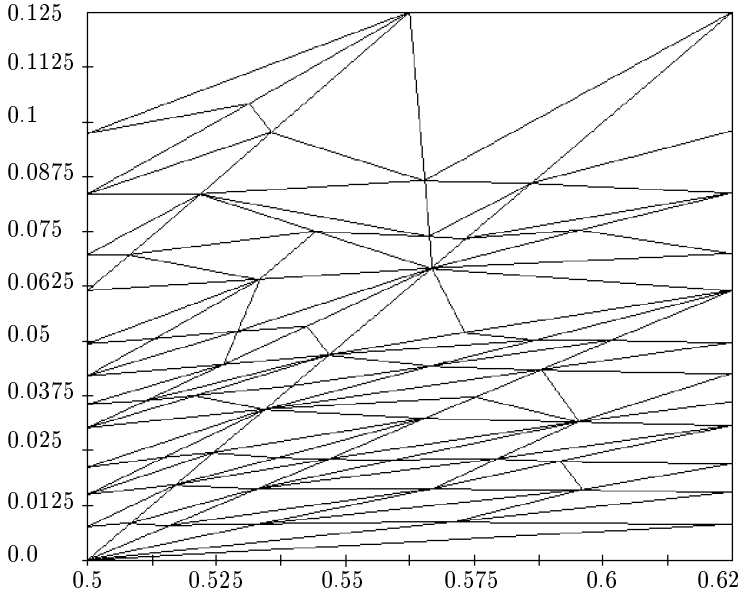

Fig. 16: Step 4; $\varepsilon = 0.05$ (at the bottom).

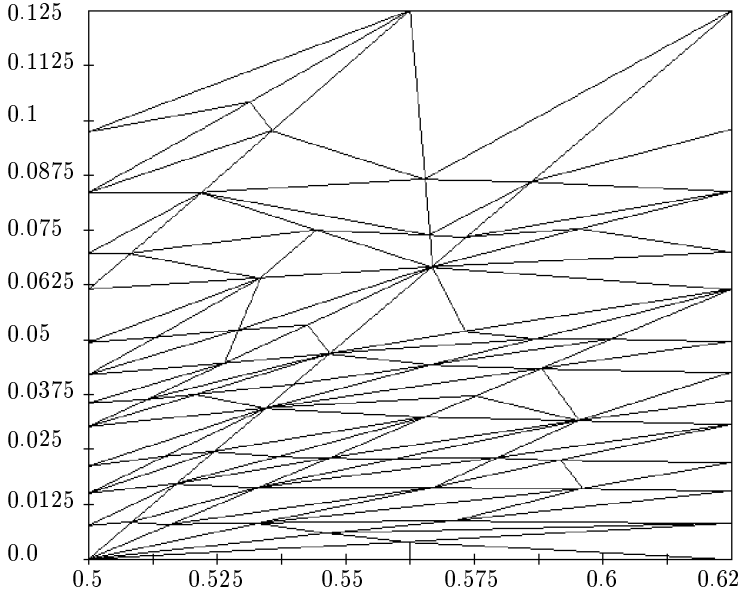

Fig. 17: Step 3; $\varepsilon = 0.05$ (the right-hand bottom corner).

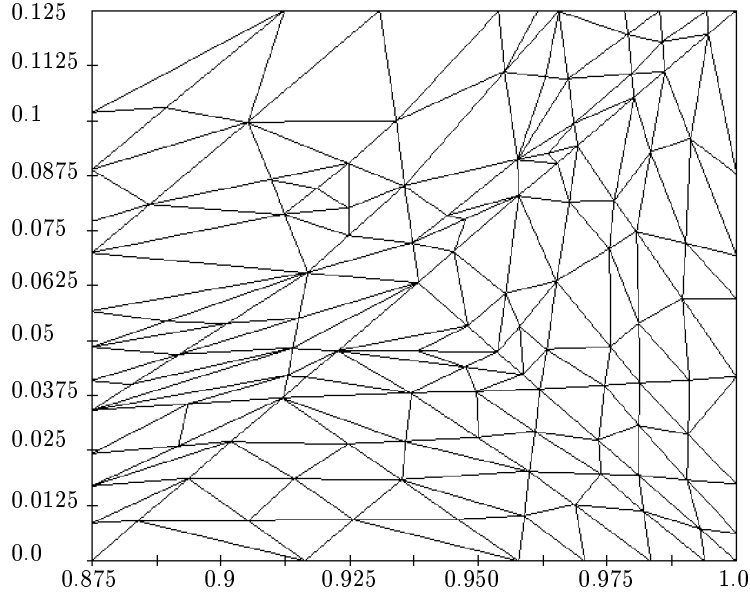

Fig. 18: Step 4; $\varepsilon = 0.05$ (the right-hand bottom corner).

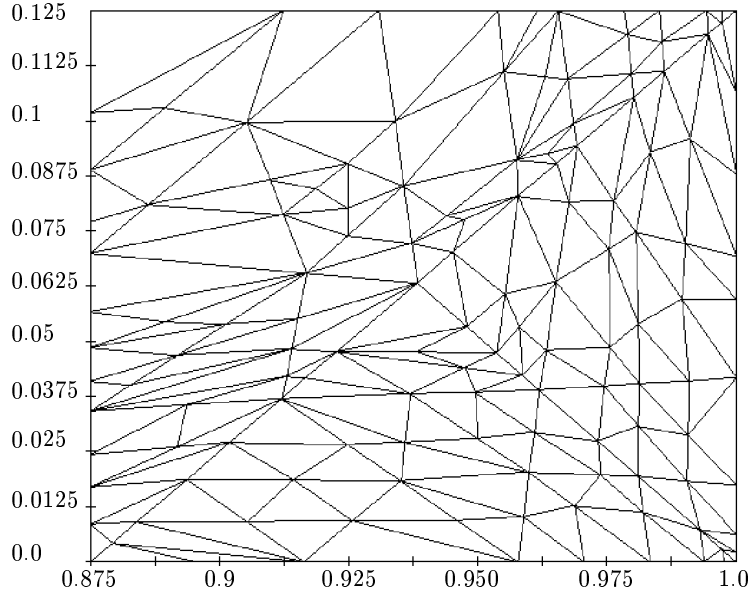

Fig. 19: Step 3; $\varepsilon = 0.05$ (the right-hand boundary).

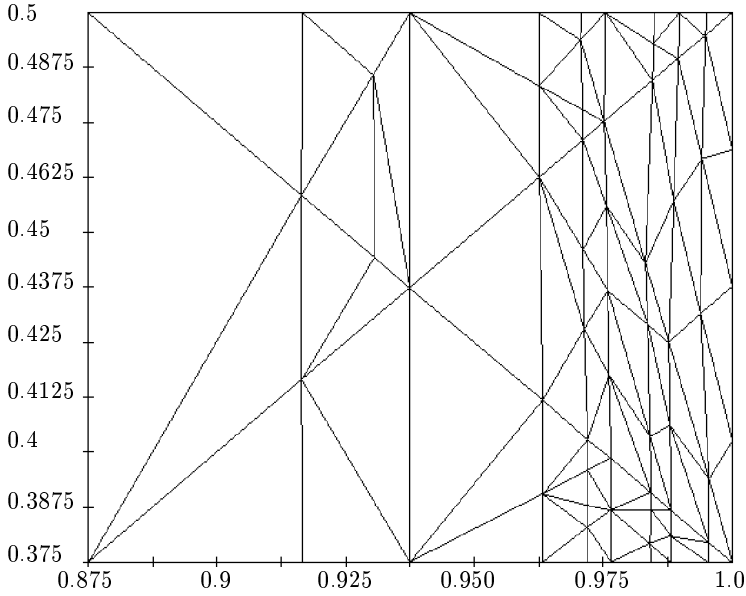

Fig. 20: Step 4; $\varepsilon = 0.05$ (the right-hand boundary).