ANNALS OF "DUNAREA DE JOS" UNIVERSITY OF GALATI
MATHEMATICS, PHYSICS, THEORETICAL MECHANICS
FASCICLE II, YEAR XVI (XLVII) 2024, No. 1
DOI: https://doi.org/10.35219/ann-ugal-math-phys-mec.2024.1.06

Algorithmic approach to the finite element approximation method

Nina N. Cazacu[1]

[1] *High School of Arts, United Principates Street, 030167 Bucharest, Romania*
[*]*Corresponding author: cazanina@yahoo.com*

Abstract

The paper provides a general method for obtaining the so-called "shape functions", which approximate the solution of systems of differential equations with boundary conditions. Unlike classic methods usually used for solving variational problems, the presented procedure approximates the unknown functions with piecewise linear functions, using a sequence of C++ code. The domain in which the system is defined is divided into finite elements, and the nodal values of the solution are obtained by solving the Fermat system associated with the specific boundary conditions of the defined domain. The proposed algorithm, based on C++ code, is designed to solve problems in two-dimensional cases or higher.

Keywords: FEM, numerical algorithm, shape functions, program, approximation method

1. INTRODUCTION

The finite element method (FEM-in short) is the most widely used approximation method ([3]) in numerical approximation methods for variational problems. The present paper provides an example of applying FEM, using C++ code, as opposed to the traditional MatLab or MathCAD programs ([4]).

In practice, the associated physical phenomena are diverse:

1. heat propagation
2. spreading the electric (or magnetic) potential
3. fluid movement through porous media
4. bending of the prismatic plate

We will work within a $\Omega \subset \Re^2$ bounded domain, with a smooth border $\partial\Omega$ along certain portions. Let's consider, in this mathematical context, the "quasi-harmonic" equation that represents the behavior of an undetermined quantity v, in two dimensions:

$$\frac{\partial}{\partial x}(k_x \frac{\partial v}{\partial x}) + \frac{\partial}{\partial y}(k_y \frac{\partial v}{\partial y}) + f = 0, (x, y) \in \Omega \quad (1)$$

The constants k_x, k_y, the function f, and also the boundary conditions are given. The approximate solution of the equation is required. In some cases, extreme conditions are also added :

$$k_x \frac{\partial v}{\partial x} l_x + k_y \frac{\partial v}{\partial y} l_y + b + av = 0; (x, y) \in \partial\Omega \quad (2)$$

with:

$$\frac{dv}{dn} = k_x \frac{\partial v}{\partial x} l_x + k_y \frac{\partial v}{\partial y} l_y; n = (l_x, l_y); (x, y) \in \partial\Omega \quad (3)$$

which represents the derivative along the direction *n*, together with the *Newmann* condition:

$$\frac{dv}{dn} = 0, (x, y) \in \partial\Omega$$

or the Dirichlet condition:

$$v = f(=0);(x,y) \in \partial\Omega$$

For two real functions $u, v \in C^1(\Omega)$, the scalar product was defined ([1]):

$$< u,v >= \iint_{\Omega} (uv + \nabla u \cdot \nabla v)dxdy \tag{4}$$

which induces the norm $\|u\| =< u,u >^{1/2}$. We have denoted by $H^1(\Omega)$, the Hilbert space obtained by completing $C^1(\Omega)$, in relation to this norm. The space of infinitely differentiable functions $C_0^{\infty}(\Omega)$, with the compact support contained in Ω, with respect to the norm $\|\cdot\|$, led to another closed subspace $H_0^1(\Omega) \subset H^1(\Omega)$. Let's consider the Hilbert space X, which verifies the relation: $H_0^1(\Omega) \subset X \subset H^1(\Omega)$. A quadratic function $I : X \to \Re$ was defined, with the variable $\nu(x, y)$ as follows:

$$I(v) = \iint_{\Omega} \left\{ \frac{1}{2} \left[k_X \left(\frac{\partial v}{\partial x} \right)^2 + k_y \left(\frac{\partial v}{\partial y} \right)^2 \right] - fv \right\} dxdy \tag{5}$$

where: $K = K(x, y)$ is assumed to be a continuous, positive function, not depending on v, and $f = f(x, y)$ is a given continuous function.

According to Euler's classical theory, the variation problem requires the determination of a function $u \in X$ that achieves the minimum of I on X, i.e.:

$$I(u) \le I(v), \forall v \in X. \tag{6}$$

2. FEM PRESENTATION -THE LINEAR APPROXIMATION METHOD

As it is known in the specialized literature, FEM approximates the unknown function with smooth functions on every element of the domain, so it results in the linear "shape functions", each finite element having associated only one single linear function.

The simplest example of a finite element in a two-dimensional case is the triangle, because the linear function in two dimensions has three coefficients that are determined by the conditions placed on the peaks of the triangle. Rectangular finite elements or, more generally, curve finite elements can also be used, with minor adaptations to improve domain coverage. It is required that some sides of the finite elements align with the domain's border $\partial\Omega$ ([4]).

It is assumed that Ω can be divided into a finite number M of triangles, such that the intersection of any two triangles is either empty, or has a common peak, or a common side.

For example, our purpose was to solve problem (6) using the finite element method, in the two dimensional case of an **octagonal domain**, and represent the solutions.

The linear approximation functions, on each element of the unknown function have the form *u(x, y) = A + Bx + Cy*, in which A,B,C are obtained using Cramer method, from a simple linear system:

$$\begin{aligned} u_i &= u(x_i, y_i) = A + Bx_i + Cy_i \\ u_j &= u(x_j, y_j) = A + Bx_j + Cy_j \\ u_m &= u(x_m, y_m) = A + Bx_m + Cy_m \end{aligned} \tag{7}$$

We did some notations, as follows:

$$\begin{array}{l} a_i = x_j y_m - x_m y_j; a_j = x_m y_i - x_i y_m; a_m = x_i y_j - x_j y_i \\ b_i = y_j - y_m = y_{jm}; b_j = y_m - y_i = y_{mi}; b_m = y_i - y_j = y_{ij}; \\ c_i = x_j - x_m = x_{jm}; c_j = x_m - x_i = x_{mi}; c_m = x_i - x_j = x_{ij} \end{array} \quad \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{vmatrix} = 2\Delta_{ijm} = \Delta \qquad (8)$$

Solving system (7), with the notations we have made, we have obtained:

$$\begin{aligned} A &= \frac{1}{2\Delta_{ijm}}(a_i u_i + a_j u_j + a_m u_m) = \frac{1}{2\Delta_{ijm}}(\sum a_i u_i) \\ B &= \frac{1}{2\Delta_{ijm}}(b_i u_i + b_j u_j + b_m u_m) = \frac{1}{2\Delta_{ijm}}(\sum b_i u_i) \\ C &= \frac{1}{2\Delta_{ijm}}(c_i u_i + c_j u_j + c_m u_m) = \frac{1}{2\Delta_{ijm}}(\sum c_i u_i) \end{aligned} \qquad (9)$$

which, by replacing in the system (7), leads to:

$$u = \frac{1}{2\Delta_{ijm}} \sum u_i (a_i + b_i x + c_i y) \qquad (10)$$

So, to write these functions, called "shape functions", we need the values of the unknown function in the network nodes, namely u_k, k = 1, ..,M, with M being the number of all finite elements in which we have decided to divide the domain. Therefore, the network has 3M nodes, if the finite element is a triangle, 4M nodes, if it is a square, and so on. Also, the coordinates of the network nodes (x_k, y_k) must be known, and also the area of one finite element, noted Δ_{ijm}. Using the standard notations, the so-called "nodal values" of the domain Ω were defined:

$$\{u\}^e = \begin{bmatrix} u_i \\ u_j \\ u_m \end{bmatrix}; \quad N_i = \frac{1}{2\Delta_{ijm}}(a_i + b_i x + c_i y); \quad N_j = \frac{1}{2\Delta_{ijm}}(a_j + b_j x + c_k y); \quad N_m = \frac{1}{2\Delta_{ijm}}(a_m + b_m x + c_m y)$$

As a consequence, the approximate solution can also be written in the standard form of the scalar product between the vector $\{u\}^e$ and the matrix $[h] = [N_i, N_j, N_m]$:

$$u = [N_i, N_j, N_m] \cdot \{u\}^e = \sum N_i \cdot u_i \qquad (11)$$

We used the addition property of the integral in conjunction with the decomposition of the domain into finite elements, obtaining a real function with M real variables u_k, where k = 1, ..,M, and M represents the number of triangles and polygon vertices. With the domain decomposed into triangles Δ_{ijm}, the function u is uniquely and continuously defined across the domain, with the help of the so-called "nodal values". A node is a common peak/vertex shared by several finite elements, allowing several elements to be linked within the $\frac{\partial I^e}{\partial u_i}$ expression. Since the finite element is, in this case, a triangle, each element contributes to only three of the differentials. The quadratic functional $I(u)$ can be minimized with respect to these values:

$$\left[\frac{\partial I}{\partial u}\right]^e = \left[\frac{\partial I^e}{\partial u_i}, \frac{\partial I^e}{\partial u_j}, \frac{\partial I^e}{\partial u_m}\right]$$

For one finite element aria, we have denoted the integral I^e, and considering $k_x = k_y = 1$, the following formula is obtained:

$$I^e = \frac{1}{2}\iint_e [(\frac{\partial N_i}{\partial x}u_i + \frac{\partial N_j}{\partial x}u_j + \frac{\partial N_m}{\partial x}u_m)^2 + (\frac{\partial N_i}{\partial y}u_i + \frac{\partial N_j}{\partial y}u_j + \frac{\partial N_m}{\partial y}u_m)^2 + fu]dxdy \quad (12)$$

The partial derivative, with respect to the u_i component, has to the general form:

$$\frac{\partial I^e}{\partial u_i} = \frac{1}{2}\iint_e \left\{ k_X \frac{\partial}{\partial u_i}\left(\frac{\partial u}{\partial x}\right) + k_y \frac{\partial}{\partial u_i}\left(\frac{\partial u}{\partial y}\right) + f\frac{\partial u}{\partial u_i}\right\} dxdy \quad (13)$$

and can be explicitly written on its components:

$$\frac{\partial I^e}{\partial u_i} = \iint_e [(\frac{\partial N_i}{\partial x}u_i + \frac{\partial N_j}{\partial x}u_j + \frac{\partial N_m}{\partial x}u_m)\frac{\partial N_i}{\partial x} + (\frac{\partial N_i}{\partial y}u_i + \frac{\partial N_j}{\partial y}u_j + \frac{\partial N_m}{\partial y}u_m)\frac{\partial N_i}{\partial y} + f\frac{\partial u}{\partial u_i}]dxdy =$$

$$= \frac{1}{4\Delta_{ijm}^2}[(b_i u_i + b_j u_j + b_m u_m)b_i + (c_i u_i + c_j u_j + c_m u_m)c_i]\iint_e dxdy + \frac{f}{2\Delta_{ijm}}\iint_e (a_i + b_i x + c_i y)dxdy = \quad (14)$$

$$= \alpha_i u_i + \alpha_j u_j + \alpha_m u_m + \frac{\Delta}{3}$$

We have denoted, for simplicity, the new coefficients $\alpha_i, \beta_i, \gamma_i$, like so:

$$\alpha_i = \frac{1}{4\Delta_{ijm}}(b_i^2 + c_i^2);\alpha_j = \frac{1}{4\Delta_{ijm}}(b_j b_i + c_j c_i);\alpha_m = \frac{1}{4\Delta_{ijm}}(b_m b_i + c_m c_i) \text{ and similarly:}$$

$$\beta_i = \frac{1}{4\Delta_{ijm}}(b_i b_j + c_i c_j);\beta_j = \frac{1}{4\Delta_{ijm}}(b_j^2 + c_j^2);\beta_m = \frac{1}{4\Delta_{ijm}}(b_m b_j + c_m c_j) \quad (15)$$

$$\gamma_i = \frac{1}{4\Delta_{ijm}}(b_i b_m + c_i c_m);\gamma_j = \frac{1}{4\Delta_{ijm}}(b_j b_m + c_j c_m);\gamma_m = \frac{1}{4\Delta_{ijm}}(b_m^2 + c_m^2)$$

In equations (12), we specified a particular notation for the last term of the integral, and also calculated its corresponding value; $\bar{x}, \bar{y}$ were the notations that represent the coordinates of the triangle's centre of gravity.

$$F = f \cdot u \Rightarrow F_i^e = u^e = \frac{f}{2\Delta_{ijm}}\iint_e (a_i + b_i x + c_i y)dxdy = \frac{1}{\Delta_{ijm}} \cdot \Delta_{ijm}(a_i + b_i \bar{x} + c_i \bar{y});$$

$$\bar{x} = \frac{x_i + x_j + x_m}{3}; \bar{y} = \frac{y_i + y_j + y_m}{3}$$

We also mention the relation: $\iint_e dxdy = 2\Delta_{ijm} = \Delta$, which has been used in the transformations we have maid in equations (14).

For minimization, all the partial derivatives of the function must be null, forming a system in the unknowns u_i:

$$\frac{\partial I}{\partial u} = \sum_e \frac{\partial I^e}{\partial u_i} = 0, i = 1,...,M \quad (16)$$

where $\boldsymbol{I^e}$ is the integral of a finite element, and the sum was calculated after all the finite elements. Accordingly, the Fermat system is:

$$\frac{\partial I^e}{\partial u_i} = \sum \alpha_i u_i + \frac{\Delta}{3} = 0, i = 1,...,M \tag{17}$$

Following the formula (10), the approximate solution is the sum of the linear functions multiplied with each peak's value u_k, for every element:

$$u = \frac{1}{\Delta}\sum_{k=1}^{M-2}\begin{bmatrix}(a_k + b_k x + c_k y)u_k \\ +(a_{k+1} + b_{k+1} x + c_{k+1} y)u_{k+1} \\ +(a_{k+1} + b_{k+2} x + c_{k+2} y)u_{k+2}\end{bmatrix}, k = 1,\overline{M-2} \tag{18}$$

3. CODIFICATION OF THE METHOD

For translating the procedure we have described, in the C++ code, that we need the coordinates of the triangles peaks in the entry file. The values are calculated in relation to the center of the octagon.

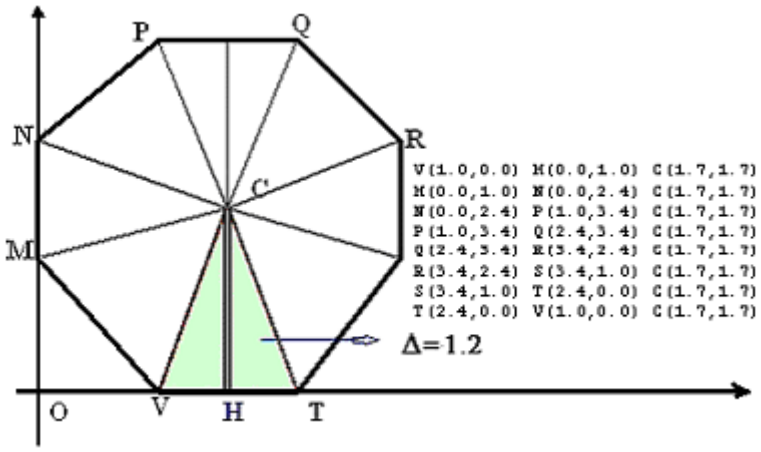


Fig.1. Applying FEM for a two dimensional domain

The numerical value of the finite element aria was calculated in the particular case of the octagon presented in figure 1:

$$\begin{aligned} &S_8 = \Delta_{CVT} = \frac{\sqrt{2}(1+\frac{\sqrt{2}}{2})}{2} = \frac{1+\sqrt{2}}{2} \cong 1,2; M = 8; \\ &\text{or}: VT = l_8 = \sqrt{2}; i_8 = \frac{\sqrt{2}}{2}\operatorname{ctg}\frac{\pi}{8} \Rightarrow S_8 = \frac{1}{2}\sqrt{2}\frac{\sqrt{2}}{2}\operatorname{ctg}\frac{\pi}{8} = \frac{1}{2}\operatorname{ctg}\frac{\pi}{8} = \frac{2,414}{2} = 1,207 \approx 1,21 \end{aligned} \tag{19}$$

The auxiliary coefficients are calculated as follows:

$$\begin{aligned} &b_1 = y_2 - y_3 = -0.7; c_1 = x_2 - x_3 = -1.7; && \alpha_1 = b_1^2 + c_1^2 = (-0.7)^2 + (-1.7)^2 = 0.7 \\ &b_2 = y_3 - y_1 = 1.7; c_2 = x_3 - x_1 = 0.7; && \rightarrow \alpha_2 = b_2 b_1 + c_2 c_1 = -(1.7)(0.7) - (-0.7)(-1.7) = -2.38 \\ &b_3 = y_1 - y_2 = -1; c_3 = x_1 - x_2 = 1 && \alpha_3 = b_3 b_1 + c_3 c_1 = 0.7 - 1.7 = -1 \end{aligned} \tag{20}$$

Consequently, for the first aria element, ΔVMC, we have obtained:

$$\frac{\partial I_{VMC}}{\partial u_1} = \frac{1}{4.8}[3.38 \cdot u_1 - 2.38u_2 - u_3] + 0.4 = 0.7u_1 - 0.5 \cdot u_2 - 0.21 \cdot u_3 + 0.4 = 0 \tag{21}$$

System (14) can also be written, with matrices coefficients, as follows:

$$\frac{\partial I(u)}{\partial u_i} = \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \cdot \begin{bmatrix} u_i \\ u_j \\ u_m \end{bmatrix} + \frac{\Delta}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = [H] \cdot [u] + F^e = [0] \tag{22}$$

which led to the associated Fermat system:

$0.7 \cdot v1 - 0.5 \cdot u2 - 0.21 \cdot u3 + 0 \cdot u4 + 0 \cdot u5 + 0 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$-0.5 \cdot u1 + 0.7 \cdot u2 - 0.21 \cdot u3 + 0 \cdot u4 + 0 \cdot u5 + 0 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$-0.21 \cdot u1 - 0.21 \cdot u2 + 0.42 \cdot u3 + 0 \cdot u4 + 0 \cdot u5 + 0 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$0 \cdot u1 + 0 \cdot u2 + 0 \cdot u3 + 0.7 \cdot u4 - 0.5 \cdot u5 - 0.2 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$0 \cdot u1 + 0 \cdot u2 + 0 \cdot u3 - 0.5 \cdot u4 + 0.7 \cdot u5 + -0.2 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$0 \cdot u1 + 0 \cdot u2 + 0 \cdot u3 - 0.2 \cdot u4 - 0.2 \cdot u5 + 0.41 \cdot u6 + 0 \cdot u7 + 0 \cdot u8 + 0.4 = 0$
$0 \cdot u1 + 0 \cdot u2 + 0 \cdot u3 + 0 \cdot u4 + 0 \cdot u5 + 0 \cdot u6 + 0.7 \cdot u7 - 0.5 \cdot u8 + 0.4 = 0$
$0 \cdot u1 + 0 \cdot u2 + 0 \cdot u3 + 0 \cdot u4 + 0 \cdot u5 + 0 \cdot u6 - 0.5 \cdot u7 + 0.7 \cdot u8 + 0.4 = 0$

solved using the eliminations method, and the solutions we have obtained are the following:

2.9, 1.7, 1.92, 2.91, 1.7, 1.96, 1.94, 1.14

It results inan approximate solution to the problem, with linear functions as components:

u=[0+(-0.7)*x+(-1.7)*y**]*(2.9)**+
[-1.7+(1.7)*x+(0.7)*y**]*(1.7)**+
[1+(-1)*x+(1)*y**]*(1.92)**+
[0+(0.7)*x+(-1.7)*y**]*(2.91)**+
[1.7+(0.7)*x+(1.7)*y**]*(1.7)**+
[0+(-1.4)*x+(0)*y**]*(1.96)**+
[0+(1.7)*x+(-0.7)*y**]*(1.94)**+
[4.08+(-0.7)*x+(1.7)*y**]*(1.14)**

4. THE PSEUDO CODE AND THE PROGRAM SOURCE CODE

The code sequences in C++ calculate the nodal values of the solution, where n represents the number of nodes, as well as the coefficients of the linear functions, and display the resulting approximate solution. The source code uses the following notations : u , v are vectors of the network nodes coordinates provided in the input file, the variables e,c,d represent the linear coefficients a_i used to calculate the elements of the matrix h, and the variable b represents the free term $\Delta/3$.The solution to the Fermat system of equations is represented by the variable x.

A repetitive cycle, conditioned at the beginning, reported to the number of nodes, reads, from the input file f, the nodes coordinates for every finite element, calculates the linear coefficients a_i, and the elements α_i of the matrix h, using the formulas in equations (20). The pseudo code of this sequence is presented below:

k←1; while(k<n) do begin
u_k, v_k; u_{k+1}, v_{k+1}; u_{k+2}, v_{k+2} ← f
$e_k \leftarrow u_{k+1} \cdot v_{k+2} - u_{k+2} \cdot v_{k+1}$; $e_{k+1} \leftarrow u_{k+2} \cdot v_k - u_k \cdot v_{k+2}$; $e_{k+2} \leftarrow u_k \cdot v_{k+1} - u_{k+1} \cdot v_k$;
$c_k \leftarrow v_{k+1} - v_{k+2}$; $d_k = u_{k+1} - u_{k+2}$;
$c_{k+1} \leftarrow v_{k+2} - v_k$; $d_{k+1} \leftarrow u_{k+2} - u_k$;
$c_{k+2} \leftarrow v_k - v_{k+1}$; $d_{k+2} \leftarrow u_k - u_{k+1}$;
$h_{k,k} \leftarrow c_k^2 + d_k^2$; $h_{k,k+1} \leftarrow c_k \cdot c_{k+1} + d_k \cdot d_{k+1}$; $h_{k,k+2} \leftarrow c_k \cdot c_{k+2} + d_k \cdot d_{k+2}$;
$h_{k+1,k} \leftarrow c_{k+1} \cdot c_k + d_{k+1} \cdot d_k$; $h_{k+1,k+1} \leftarrow c_{k+1}^2 + d_{k+1}^2$; $h_{k+1,k+2} \leftarrow c_{k+1} \cdot c_{k+2} + d_{k+1} \cdot d_{k+2}$;
$h_{k+2,k} \leftarrow c_{k+2} \cdot c_k + d_{k+2} \cdot d_k$; $h_{k+2,k+1} \leftarrow c_{k+2} \cdot c_{k+1} + d_{k+2} \cdot d_{k+1}$; $h_{k+2,k+2} \leftarrow c_{k+2}^2 + d_{k+2}^2$;
k←k+3
end while

Next code sequences were meant to solve the Fermat system, using the specific method of eliminations, with the following associated pseudo code:

```
i=n; while (h_ii=0) do begin
                    i←i-1; p←i;
                    end while
x_p←b_p/ h_pp;
for i=p-1, i>=1, i←i-1 do begin
                        temp←b_i
                        for j=i+1, j<=p, j←j+1 do begin
                                              temp←temp-h_ij·x_j
                                              end do
                         if h_ii <>0 then x_i←temp/h_ii  end if
                         end do
```

The source code for the the octagonal domain example is found below, in accordance with the theoretical results we have outlined:

```
typedef float matrix[100][100];
void main()
{ matrix h; float b[100],c[100],d[100],
e[100],x[100],u[100],v[100];
int p,i,j,k,l,n,vb;float aux,temp;
fstream f("DATE1.in",ios::in),
g("DATE2.out",ios::out);
n=8;
for(i=1;i<=n;i++){for(j=1;j<=n;j++)
h[i][j]=x[i]=c[i]=d[i]=0;}
for(i=1;i<=n;i++)b[i]=2*(1.2)/3;//f=1
p=1; while(p<n){
f>>u[p]>>v[p]>>u[p+1]>>v[p+1]>>u[p+2]>>
v[p+2];
e[p]=u[p+1]*v[p+2]-u[p+2]*v[p+1];
e[p+1]=u[p+2]*v[p]-u[p]*v[p+2];
e[p+2]=u[p]*v[p+1]-u[p+1]*v[p];
c[p]=v[p+1]-v[p+2];        c[p+1]=v[p+2]-v[p];
c[p+2]=v[p]-v[p+1];
d[p]=u[p+1]-u[p+2];        d[p+1]=u[p+2]-u[p];
d[p+2]=u[p]-u[p+1];
h[p][p]=pow(c[p],2)+pow(d[p],2);
h[p][p+1]=c[p]*c[p+1]+d[p]*d[p+1];
h[p][p+2]=c[p]*c[p+2]+d[p]*d[p+2];
h[p+1][p]= c[p+1]*c[p]+d[p+1]*d[p];
h[p+1][p+1]=pow(c[p+1],2)+pow(d[p+1],2);
h[p+1][p+2]=c[p+1]*c[p+2]+d[p+1]*d[p+2];h
[p+2][p]=c[p+2]*c[p]+d[p+2]*d[p];
h[p+2][p+1]=c[p+2]*c[p+1]+d[p+2]*d[p+1];
h[p+2][p+2]=pow(c[p+2],2)+pow(d[p+2],2);
p+=3;}
for(i=1;i<=n;i++)for(j=1;j<=n;j++)
h[i][j]/=(2*1.2);
g<<"matrix H elements:"<<endl;
for(i=1;i<=n;i++){ g<<endl;
for(j=1;j<=n;j++) g<<h[i][j]<<" " ;}
 //The solution's values are calculated using the
elimination method
i=n;while(h[i][i]==0)i--;p=i;
x[p]=b[p]/h[p][p];
for(i=p-1;i>=1;i--){ temp=b[i];
for(j=i+1;j<=p;j++) temp-=h[i][j]*x[j];
if (h[i][i]!=0) x[i]=temp/h[i][i];}
g<<endl;  g<<" Fermat system"<<endl;
for(i=1;i<=p;i++){for(j=1;j<=p;j++)
if (j!=1)g<<"+"<<h[i][j]<<"*v"<<j;else
g<<h[i][j]<<"*v"<<j; g<<"=0"<<endl;}
g<<endl;g<<"Fermat system solutions:"
<<endl; for(k=1;k<=p;k++) g<<x[k]<<" ";
g<<endl<<"The linear functions:"<<endl;
for(k=1;k<=p;k++){
g<<"["<<e[k]<<"+("<<c[k]<<")*x+("<<d[k]<
<")*y]*("<<x[k]<<")"<<endl;}
g<<"The solution with linear functions as
components:"<<endl;
for(k=1;k<p;k++){
g<<"["<<e[k]<<"+("<<c[k]<<")*x+("<<d[k]<
<")*y]*("<<x[k]<<")+"<<endl;}
g<<"["<<e[p]<<"+("<<c[p]<<")*x+("<<d[p]<
<")*y]*("<<x[p]<<")"<<endl;
                f.close();  g.close();}
```

ENTRY FILE

```
1.0 0.0  0.0 1.0  1.7 1.7
0.0 1.0  0.0 2.4  1.7 1.7
0.0 2.4  1.0 3.4  1.7 1.7
1.0 3.4  2.4 3.4  1.7 1.7
```

EXIT FILE

FERMAT SYSTEM

```
 0.7*V1-0.5*V2+-0.21*V3+0*V4+0*V5+0*V6+0*V7+0*V8=0
-0.5*V1+0.7*V2+-0.21*V3+0*V4+0*V5+0*V6+0*V7+0*V8=0
-0.21*V1-.21*V2+0.42*V3+0*V4+0*V5+0*V6+0*V7+0*V8=0
 0*V1+0*V2+0*V3+0.7*V4+-0.5*V5+-0.2*V6+0*V7+0*V8=0
```

2.4 3.4 3.4 2.4 1.7 1.7 0*V1+0*V2+0*V3+-0.5*V4+0.7*V5+-0.2*V6+0*V7+0*V8=0
3.4 2.4 3.4 1.0 1.7 1.7 0*V1+0*V2+0*V3+-0.2*V4-0.2*V5+0.41*V6+0*V7+0*V8=0
3.4 1.0 2.4 0.0 1.7 1.7 0*V1+0*V2+0*V3+0*V4+0*V5+0*V6+0.7*V7+-0.5*V8=0
2.4 0.0 1.0 0.0 1.7 1.7 0*V1+0*V2+0*V3+0*V4+0*V5+0*V6+-0.5*V7+0.7*V8=0

FERMAT SYSTEM SOLUTIONS:
2.9 1.7 1.92 2.91 1.7 1.96 1.94 1.14

THE SOLUTION WITH LINEAR FUNCTIONS AS COMPONENTS:
[0+(-0.7)*X+(-1.7)*Y]*(2.9)+
[-1.7+(1.7)*X+(0.7)*Y]*(1.7)+
[1+(-1)*X+(1)*Y]*(1.92)+
[0+(0.7)*X+(-1.7)*Y]*(2.91)+
[1.7+(0.7)*X+(1.7)*Y]*(1.7)+
[0+(-1.4)*X+(0)*Y]*(1.96)+
[0+(1.7)*X+(-0.7)*Y]*(1.94)+
[4.08+(-0.7)*X+(1.7)*Y]*(1.14)

THE APPROXIMATE SOLUTION: 5.38x-4.5y+6.57=0

5. RESULTS AND DISCUSSION

For the graphical visualization of the solution, the ordinates of the points have to be calculated with the aid of the linear "shape functions" formulas and the initial coordinate values.

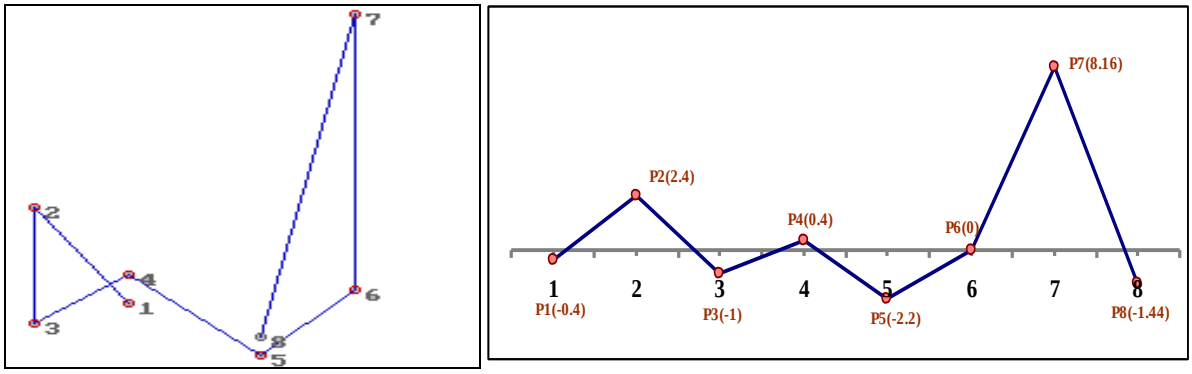
Table 1. Listed values for the domain peaks ordinates

	x_k	**y formulas**	y_k **values**
1	1	y=-0.4x	-0.4
2	0	y=-2.4x+2.4	2.4
3	0	y=x-1	-1
4	1	y=0.4x	0.4
5	2.4	y=-0.4x-1	-2.2
6	3.4	x=0	0
7	3.4	y=2.4x	8.16
8	2.4	y=0.4x-2.4	-1.44

For the graphical visualization of the linear components, using the new positions of the eight peaks based on the new/updated ordinate values, we have used the following sequence of code(in graphical mode):

```
u[1]=1;u[2]=0; u[3]=0; u[4]=1;u[5]=2.4; u[6]=3.4; u[7]=3.4; u[8]=2.4; v[1]=-0.4*u[1]; v[2]=-2.4*u[2]
+2.4; v[3]=u[3]-1; v[4]=0.4*u[4]; v[5]=-0.4*u[5]-1;   v[6]=0; v[7]=2.4*u[7];  v[8]=0.4*u[8]-2.4;
i=1;
while(i<8)  { line(getmaxx()/4+40*u[i],getmaxy()/2-20*v[i],
getmaxx()/4+40*u[i+1],getmaxy()/2-20*v[i+1]);i++;}
line(getmaxx()/4+40*u[1],getmaxy()/2-20*v[1], getmaxx()/4+40*u[8],getmaxy()/2-20*v[8]);
char s[10];    setcolor(8);
k=1;outtextxy(getmaxx()/4,2*getmaxy()/3,"Graphic of the shape functions");
while(k<8){ setcolor(4); circle(getmaxx()/4+40*u[k],getmaxy()/2-20*v[k],2); setcolor(8); itoa(k,s,10);
outtextxy(getmaxx()/4+40*u[k]+5,getmaxy()/2-20*v[k],s); k++; }
```

The result was the graphical representation in Figure 2a), the two-dimensional C++ graphical image of the solution components. In Figure 2b), a similar graphical representation was displayed, obtained using MS Excel tool, with the same values. The similarity between both forms can be seen/is obvious. They are not identical, due to the reference sets: according to the algorithm C++ code, the reference set consists of the first coordinates of the octagon nodes in two-dimensional space, as they were located in the domain; for the same purpose, MS Excel used a different scale and the reference set of eight equally spaced points is displayed along the horizontal axis, ordered relative to the origin.



a) ***b)***

Fig.2. Visualization of the linear solution components-the "shape functions"

CONCLUSIONS

1. The application to n-dimensional problems will follow the same procedure (so it can be applied in the same way for both planar and spatial problems). For three-dimensional problems, the algorithm will use three dimensional matrices.

2. Boundary conditions are included in the discrete problem.

3. The piecewise approximation polynomials are chosen so that they are uniquely determined by the nodes of the finite element network.

If the number of the finite elements increases, the approximate solution becomes more accurate. The number M of finite elements was obtained by dividing the total area of the region by the area of one element.

4. The C++ code does not depend on the dimension of the domain, though a regular shape, decomposable into triangles as finite elements, is suitable/ideal/fit. The coordinates are included in the entry file of the program.

References

1. Arnold V.I., *Ordinary differential equations*, Scientific and Encyclopedic Publishing House, Bucharest, 1978.
2. Barbu V., Lasiecka I., Triggiani. R., *Memoirs of the American Mathematical Society-Tangential Boundary Stabilization of the Navier-Stokes Equations*, Providence, Rhode Island, 2006.
3. Brezis H., *Functional analysis*, Romanian Academy Publishing House, 2002.
4. Keener J.P., *Principles of Applied Mathematics: Transformation and Approximation*, Westview Press, Cambridge, 2000.
5. Maksay S., Bistrian D., *Method of the finite elements*, Iasi, Cermi, 2008.
6. Mateescu G.D., Moraru P. F., *Informatics*, Donaris Publishing House, 2006.
7. Mateescu G.D., Mateescu I.C., *Numerical Analysis*, Petrion Publishing House, 1996.