AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

FACULTY OF METALS ENGINEERING AND INDUSTRIAL COMPUTER SCIENCE

Finite Element Method.

Developed by:

Kacper Suder

Name and surname

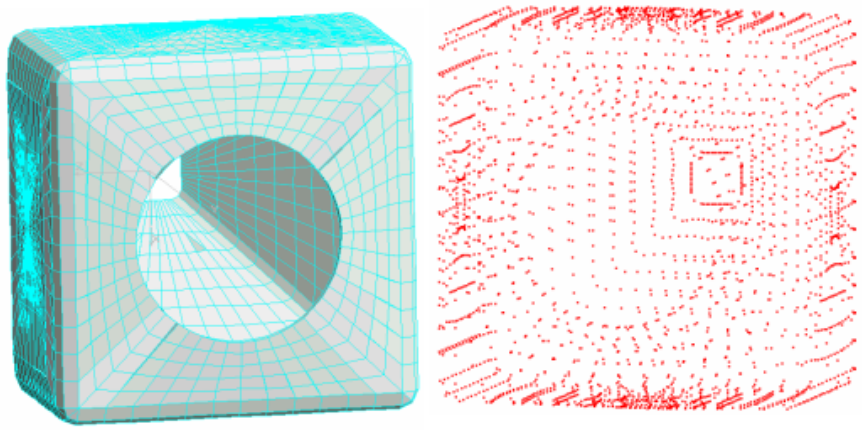
C++	20.01.23
Language	Data

1. The purpose of the software:

As part of laboratory exercises, software for the finite element method has been developed to simulate the unsteady heat transport process with a convective boundary condition. Additionally, numerical integration using Gauss-Legendre quadrature has been developed.

2. Theoretical introduction:

The Finite Element Method - a method of solving differential equations, used in industry for conducting simulations (e.g. stresses, strains). In order to perform a simulation on the tested object (e.g. sheet metal)**,** a **Finite Element Mesh (FEM)** is applied to it.

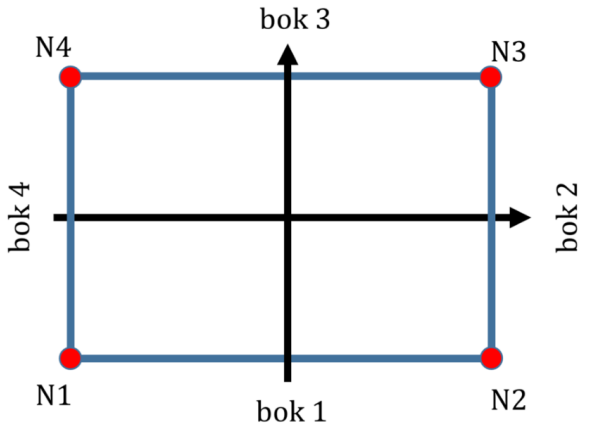


/Finite element model of a mixer tank prepared for solving using the Finite Element Method. On the left - the structure is divided into elements, on the right, the nodes are visible, where the elements are connected to each other - source: Wiesław Śródka - "Three lessons on the finite element method. Supplementary materials for the subject of strength of materials", PUBLISHING OFFICE OF WROCŁAW UNIVERSITY OF TECHNOLOGY/

In the case of the created software, only heat transport is calculated using the Fourier-Kirchhoff equation for an unsteady state with the use of the convective boundary condition (for a variable heat flux).

The Finite Element Mesh (FEM) - is built of a finite number of spatial objects connected to each other. The smallest part of the mesh is an element. It consists of nodes and walls. When creating an FEM mesh, the difference in node numbering on one element should be as small as possible (due to subsequent calculations).

An example of a finite element in the FEM mesh:



Where the designations N1, N2, N3, N4 are responsible for nodes.

Convection - a heat transfer process associated with the macroscopic movement of matter in fluids (gases and liquids) or plasma. The driving force of convective heat exchange is the temperature gradient.

Mathematical formula: $q = \alpha * (t - t_{\infty})$

where:

Coefficient α - depends on the material and its surface - it determines how "eagerly" heat is transferred to or from a given object;
t - temperature of the tested object;
t∞ - ambient temperature;

Interpolation - Interpolation in FEM is used to calculate, among other things, the temperature and stresses at a given point that is not located on a node using the shape function.

Mathematical formula: $t_p = \{N\}^T * \{t\} = N_1 * t_1 + N_2 * t_2 + \ldots$

Shape function - allows the determination of the sought quantities at points that are not located on the nodes. The formula for the shape function and the number of functions depend on the number of nodes in the element and the space. However, regardless of the formula and quantity, the sum of the shape functions is equal to 1, while in its own node, the value of the shape function is equal to 1.

The following formulas (shape functions are calculated for a two-dimensional element with four sides) were used in the created software:

$$N1 = 0.25(1-\xi)(1-\eta)$$
$$N2 = 0.25(1+\xi)(1-\eta)$$
$$N3 = 0.25(1+\xi)(1+\eta)$$
$$N4 = 0.25(1-\xi)(1+\eta)$$

Gauss-Legendre quadratures - interpolatory quadratures, in which the integrand is approximated by an interpolating polynomial.

Simulation:

The software implementation started with the implementation of **the Fourier-Kirchhoff equation** for the steady-state condition:

$$div(k(t)grad(t))=0,$$

This equation, without the use of differential operators, takes the form:

$$\frac{\partial}{\partial x}\left(k_x(t)\frac{\partial t}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_y(t)\frac{\partial t}{\partial y}\right)+\frac{\partial}{\partial z}\left(k_z(t)\frac{\partial t}{\partial z}\right)=0,$$

where:
$k_x(t)$, $k_y(t)$, $k_z(t)$ - anisotropic heat conduction coefficients dependent on temperature t in three directions;

To correctly introduce the equation into the created program and enable simulation, the first step is to transform the above differential equation into the form of a functional (integral equation):

$$J=\int_V \frac{1}{2}\left(k_x(t)\left(\frac{\partial t}{\partial x}\right)^2+k_y(t)\left(\frac{\partial t}{\partial y}\right)^2+k_z(t)\left(\frac{\partial t}{\partial z}\right)^2\right)dV$$

Assuming that for isotropic materials the values of anisotropic thermal conductivity are equal to each other, the fragment of the functional that will be solved to perform the simulation is as follows:

$$J=\int_V \left(\frac{k(t)}{2}\left(\left(\frac{\partial t}{\partial x}\right)^2+\left(\frac{\partial t}{\partial y}\right)^2+\left(\frac{\partial t}{\partial z}\right)^2\right)\right)dV$$

To fully carry out the calculations, it is necessary to add the convection boundary condition. Since the discretization of the considered problem consists in dividing the area into elements and presenting the temperature inside the element as a function of nodal values according to the relationship:

$$t = \sum_{i=1}^{n} N_i t_i = \{N\}^T \{t\}.$$

In this way (using interpolation), a continuous temperature distribution is obtained. As a result, after solving the functional, the temperature distribution inside the element is obtained, and consequently in the entire grid (not only at nodes). The final equation has the following form:

$$J = \int_V \left(\frac{k}{2} \left(\left(\left\{ \frac{\partial \{N\}}{\partial x} \right\}^T \{t\} \right)^2 + \left(\left\{ \frac{\partial \{N\}}{\partial y} \right\}^T \{t\} \right)^2 + \left(\left\{ \frac{\partial \{N\}}{\partial z} \right\}^T \{t\} \right)^2 \right) \right) dV + \int_S \frac{\alpha}{2} \left(\{N\}^T \{t\} - t_\infty \right)^2 dS$$

To obtain the result, it was necessary to minimise the functional, which involves calculating partial derivatives with respect to the nodal values of temperature {t}. As a result, the following system of equations was obtained:

$$\frac{\partial J}{\partial \{t\}} = \int_V \left(k \left(\left\{ \frac{\partial \{N\}}{\partial x} \right\} \left\{ \frac{\partial \{N\}}{\partial x} \right\}^T + \left\{ \frac{\partial \{N\}}{\partial y} \right\} \left\{ \frac{\partial \{N\}}{\partial y} \right\}^T + \left\{ \frac{\partial \{N\}}{\partial z} \right\} \left\{ \frac{\partial \{N\}}{\partial z} \right\}^T \right) \{t\} \right) dV +$$

$$+ \int_S \alpha \left(\{N\}^T \{t\} - t_\infty \right) \{N\} dS = 0$$

The resulting system of equations in matrix form is:

$$[H]\{t\} + \{P\} = 0$$

However, this is an incomplete equation (used for steady-state conditions) that does not allow for simulating the change in temperature at nodes over time. This is carried out for non-stationary (unsteady) processes, hence the heat transport equation (Fourier-Kirchhoff equation for unsteady state) takes the form:

$$div(k(t)grad(t)) = c\rho \frac{\partial t}{\partial \tau}$$

After necessary transformations, an appropriate scheme for determining the temperature {t1} was selected. Since the explicit scheme has limited applications due to its weak stability for various time steps ($\triangle\boldsymbol{\tau}$), an implicit scheme for determining the temperature {t1} was used, which in matrix form has the following form:

$$\left([H] + \frac{[C]}{\Delta\tau}\right) \{t_1\} - \left(\frac{[C]}{\Delta\tau}\right)\{t_0\} + \{P\} = 0$$

where:

1. **The matrix H** - determines how heat is transported in the material. In the above equation, the matrix HBC was added to it.
   Formula for the matrix H:

$$[H] = \int_V k(t) \left(\left\{ \frac{\partial \{N\}}{\partial x} \right\} \left\{ \frac{\partial \{N\}}{\partial x} \right\}^T + \left\{ \frac{\partial \{N\}}{\partial y} \right\} \left\{ \frac{\partial \{N\}}{\partial y} \right\}^T \right) dV + [H_{BC}]$$

where:
k(t) - thermal conductivity coefficient

2. **The matrix HBC** - a part of the convection boundary condition, which, after being added to the H matrix in the final equation, includes the unknown temperature (α * t). Separating the boundary condition and adding this part to the H matrix was necessary because, according to the definition of solving a system of equations, the unknown (desired) temperature must be present along with the H matrix.
   Formula for the HBC matrix:

$$[H_{BC}] = \int_S \alpha(\{N\}\{N\}^T)\, dS$$

3. **The matrix C** - the matrix of thermal capacity of the material. Physical parameters included in matrix C determine how much energy the material can accumulate.
   Formula for matrix C:

$$[C] = \int_V c\rho \{N\}\{N\}^T dV\,.$$

   where:

   c – specific heat
   ρ – material density

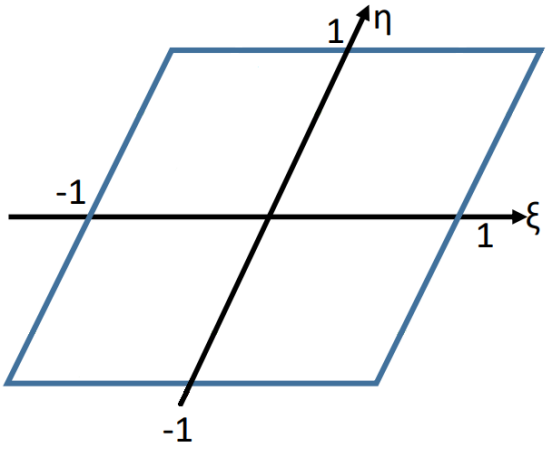
4. **Vector P** - a load vector that indicates what loads are exerted by temperature on all nodes. It implements a boundary condition for the free terms with a known temperature, which is necessary due to the definition of solving a system of equations.
   Formula for vector P:

$$[P] = \int_S \alpha \{N\} t_{ot}\, dS$$

Furthermore, to fully carry out the integration, it is necessary to calculate the Jacobian determinant and multiply individual matrices (along with the P vector) by the appropriate determinant and weights, which depend on the chosen integration scheme. These weights are obtained from the **Gauss-Legendre quadrature table.**

The Jacobian determinant for a 1D system is the ratio of the length of an element in the x-axis to the length of an element in the ksi-axis. In the case of a 2D system, the Jacobian determinant is the ratio of the areas of the global and local systems.

During the computations, a **finite element is used in the local system**, which has the following form:



To transform an element from the grid in the global system to an element in the local system (a normalised element), the Jacobian matrix is used, and in the opposite direction - the inverse Jacobian matrix. These matrices are called the Jacobians of the transformation. Such transformations are used because it is easier to perform calculations in the interval <-1, 1> for an element in the shape of a square. For this purpose, integration points are used, whose values are taken from **the Gauss-Legendre quadrature table:**

N	k	Węzły x_k	Współczynniki A_k
1	0; 1	$\mp 1/\sqrt{3}$	1
2	0; 2	$\mp\sqrt{3/5}$	5/9
	1	0	8/9
3	0; 3	∓0.861136	0.347855
	1;2	∓0.339981	0.652145
4	0;4	∓0.906180	0.236927
	1;3	∓0.538469	0.478629
	2	0	0.568889

Jacobian matrix:

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}$$

Inverse Jacobian matrix:

$$[J]^{-1} = \frac{1}{\det[J]} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix}$$

3. Description of the created software:

The created software works correctly for every shape of an element in the global system, whether it is a square/rectangle (an example of such a grid can be found in the file "Test1_4_4.txt") or any other quadrilateral (examples of such grids are located in the files "Test2_4_4_MixGrid.txt" and "Test3_31_31.txt").

3.1 Implementation of structures and loading data from a file:

At the beginning, the following structures were implemented, allowing to store the necessary data concerning the analysed mesh and material properties:

Node structure:

```
//Struktura węzła:
struct NODE {
    double x = 0.0; //współrzędna x-owa
    double y = 0.0; //współrzędna y-owa
    double nodeTemp = 0.0; //temperatura węzła
    int BC = 0; //flaga wskazująca na obecność warunku brzegowego (0 oznacza brak warunku brzegowego).
};
```

/Inside the implemented structure, data for only one node is stored/

Element structure:

```
struct ELEMENT {
    int ID[4] = { 0, 0, 0, 0 };
};
/*
* Wczytywanie ID węzłów:
*
*    ID[3]               ID[2]
*        _________________
*        |               |
*        |     <--_      |
*        |          \    |
*        |   \      /    |
*        |     -__-      |
*        |               |
*        |_______________|
*
*   ID[0]                 ID[1]
*/
```

/Structure containing an ID for a given element (if different numbers of nodes are needed in the element, then another structure holding IDs should be added)/

Grid structure:

```
//struktura siatki
struct GRID {
    int nE = 0; //liczba elementów.
    int nN = 0; //liczba węzłów.
    NODE* ND = new NODE[nN]; //node Table.
    ELEMENT* EL = new ELEMENT[nE]; //Element table.
} struktura;
```

GlobalData - properties of the test material + simulation data:

```
struct GlobalData {
    double alfa = 0.0; //współczynnik konwekcji
    double tOt = 0.0; //temperatura otoczenia
    double cp = 0.0; //ciepło właściwe
    double gamma = 0.0; //gęstość
    double simulationTime = 0.0; // czas symulacji
    double delt = 0.0; //interwał (co ile krok)
    double pc = 0.0; // przewodniość cieplna
    double initialTemp = 0.0; // temp poczatkowa
    vector<NODE> pom_nN;
    vector<ELEMENT> pom_nE;

} dane;
```

Next, the data from the text file "Test1_4_4.txt" provided by the instructor was implemented. To do this, the "readFromFile()" function was written, which reads the data from the file for any arrangement of data inside it.

```
void readFromFile() {
    double x = 0.0;
    string usun;
    string wyraz;


    ifstream odczyt;
    odczyt.open("Test1_4_4.txt");

    do {
        odczyt >> wyraz;
        if (wyraz == "SimulationTime") {
            odczyt >> dane.simulationTime;
        }
        else if (wyraz == "SimulationStepTime") {
            odczyt >> dane.delt;
        }
        else if (wyraz == "Conductivity") {
            odczyt >> dane.pc;
        }
        else if (wyraz == "Alfa") {
            odczyt >> dane.alfa;
        }
        else if (wyraz == "Tot") {
            odczyt >> dane.tOt;
        }
        else if (wyraz == "InitialTemp") {
            odczyt >> dane.initialTemp;
        }
        else if (wyraz == "Density") {
            odczyt >> dane.gamma;
        }
        else if (wyraz == "SpecificHeat") {
            odczyt >> dane.cp;
        }
        else if (wyraz == "Nodes") {
            odczyt >> wyraz;
            odczyt >> struktura.nN;
            dane.pom_nN = Utworz_nN(struktura.nN);
```

```
            for (int i = 0; i < struktura.nN; i++) { //uzupełnienie temperatury wezłów
                dane.pom_nN[i].nodeTemp = dane.initialTemp;
            }
        }
        else if (wyraz == "Elements") {
            odczyt >> wyraz;
            cout << wyraz << endl;
            odczyt >> struktura.nE;
            dane.pom_nE = Utworz_nE(struktura.nE);
        }
        else if (wyraz == "*Node" || wyraz == "*BC") {
            if (wyraz == "*Node") {
                for (int i = 0; i < struktura.nN; i++) {
                    odczyt >> wyraz;
                    odczyt >> dane.pom_nN[i].x;
                    odczyt >> usun;
                    odczyt >> dane.pom_nN[i].y;
                }
            } else {
                int pom = 0;
                for (;;) {
                    odczyt >> pom;
                    dane.pom_nN[pom-1].BC += 1;
                    odczyt >> usun;
                    if (odczyt.eof()) { break; }
                }
            }
        }
        else if (wyraz == "*Element,") {
            odczyt >> wyraz;
            for (int i = 0; i < struktura.nE; i++) {
                odczyt >> wyraz;
                for (int j = 0; j < 4; j++) {
                    odczyt >> dane.pom_nE[i].ID[j];
                    if (j < 3) odczyt >> usun;
                }
            }
        }
        else {
            continue;
        }
    } while (!odczyt.eof());
}
```

Additionally, the function used to create nodes and elements has been implemented:

```
vector <NODE> Utworz_nN(int i) {
    vector <NODE> pom_nN(i + 2);
    return pom_nN;
}

vector <ELEMENT> Utworz_nE(int i) {
    vector <ELEMENT> pom_nE(i + 2);
    return pom_nE;
}
```

/To avoid memory errors, the size of a given vector was declared larger than required (by 2)/

Element4:

```
struct Element4 {
    friend Bok;
public:
    //Punkty całkowania:
    vector <double> ksi;
    vector <double> eta;

    //Do macierzy Jakobiego (zwykłego i odwróconego):
    vector <vector <double>> pochodna_ksi;
    vector <vector <double>> pochodna_eta;
    vector <vector <double>> pochodnaDlaNPoX;
    vector <vector <double>> pochodnaDlaNPoY;

    vector <vector <double>> pochodna_Po_X_Y;
    vector <vector <double>> odwroconaDlaEta;
    vector <vector <double>> odwroconaDlaKsi;

    vector <double> pochodna_Po_Ksi;
    vector <double> pochodna_Po_Eta;

    //częściowe macierze H
    vector <vector <double>> macierzH;

    //zsumowane macierze H dla danego elementu
    vector <vector <double>> macierzHFinall;

    //Warunki brzegowe na poszczególnych bokach:
    vector <Bok> boki;

    //Macierz warunków brzegowych
    vector<vector<vector<double>>> HBC;

    //Wektor obciążeń
    vector <vector <double>> P;
    vector <vector <double>> pAgregowane;
```

```
    //Macierz C
    vector<vector<double>> macierzC;
    vector <vector <double>> macierzCFinall;

    //Funkcje kształtu po objętości
    vector <vector <double>> N;
    vector <vector <double>> NT;

    //Temperatura na węzłach
    vector <vector <double>> tempNaWezlach;

    //"Finałowa macierz" (agregowana macierz HBC + H)
    vector <vector <double>> agregacja;

    //"Finałowa macierz" (agregowana macierz C)
    vector <vector <double>> agregacjaMacierzyC;

    //wyznacznik macierzy
    vector <double> detJ;

    vector <double> wspolczynnik;
    //ilość punktów całkowania
    int rozmiar = 0;

    Element4(int value, int punktyBok);
    void pchodnKsi();//Obliczanie pochodnej N po ksi
    void pochodnaEta();//Obliczanie pochodnej N po eta
    void jakobian();//Obliczanie macierzy Jakobiego wraz z wyznacznikiem
    void macierzOdwrotna(); //Obliczanie macierzy odwrotnej Jakobiego
    void pochodnaDlaN(); //Obliczanie pochodnej N po x oraz N po Y
    void macierzHBCorazWektorP(); //Zsumowanie macierzyHBC i P z poszczególnych boków
    void macierzHplusC();//Zsumowanie macierzyH oraz macierzyC z punktów całkowania dla poszczególnych elementów
    void Agregacja(); //Agregacja macierzy H oraz macierzy C
    void obliczanieTemperaturyNaWezlach();//Przeprowadzenie symulacji
    //Tworzenie funkcji kształtu dla całki po objętości:
    vector<vector<double>> stworzMacierzN(vector<double> eta, vector<double>ksi, int iloscPc);
```

The following steps were implemented during the course to create the "Element4" structure, which contains fields and methods responsible for individual steps of integrating the H matrix, such as:

- Calculating derivatives $\frac{\partial N}{\partial \xi}$ and $\frac{\partial N}{\partial \eta}$ - "*pchodnKsi()"* and"*pochodnaEta()"*;
- Calculation of Jacobi matrix [J] and determinant (detJ)- "*detJakobian()*";
- Calculating the inverse Jacobi matrix $[J^{-1}]$;
- Calculating derivatives $\frac{\partial N}{\partial x}$ and $\frac{\partial N}{\partial y}$;
- Summation of the matrix H from the points of integration for individual elements - *"macierzHplusC()" ;*

Beside the H matrix, the C matrix is also created in this structure. It contains fields and methods responsible for adding up and storing the values of the HBC matrix and the P vector. Additionally, aggregation and simulation are performed within this structure.

By default, this structure should represent characteristics only for a four-node element (in the case of elements with a different number of nodes, a new structure, e.g. Element3 for a three-node element, should be created). Unfortunately, some fields and methods that should belong to a separate Universal Element structure or should be separate functions have been added to this structure. However, this does not affect the final result of the simulation.

Side - boundary condition of convection:

```
struct Bok { //warunek brzegowy konwekcji
    int iloscPc = 0;
    int ID; //wybor budowanego boku - 0 dolny bok, 1 prawy bok, 2 górny bok, 3 lewy bok.
    vector<int> nrElementu;
    vector <double> ksi;
    vector <double> eta;
    vector <double> wspolczynnik;
    vector<vector<double>> P; //wektor obciążeń
    vector<vector<double>> N; //funkcja kształtu
    vector<vector<double>> NT;//funkcja ksztaltu transponowana
    vector<vector<vector<double>>> HBC;//Macierz HBC
    Bok(int iloscPunktow, int ID);
    void stworzBok();
};
```

The last implemented structure is called "Bok" which contains fields and methods responsible for performing calculations for the surface integral - in the considered software, the HBC matrix and P vector are calculated within the structure.

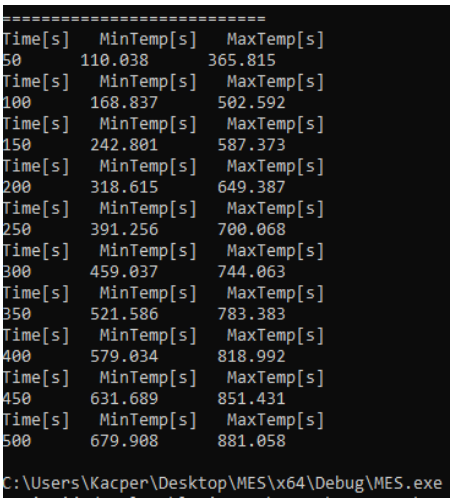
The "createBok()" method performs calculations for a given side of the element if there is a boundary condition on it.

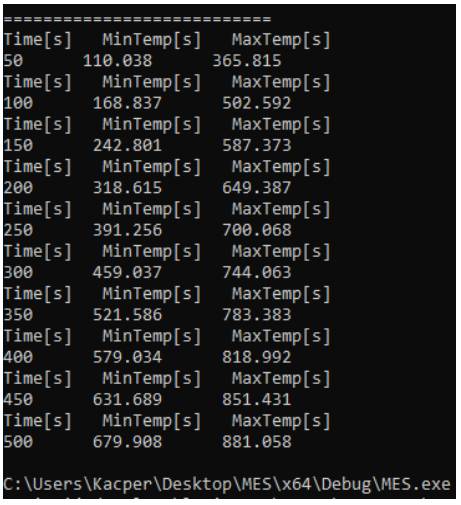
4. Analysis of the obtained results:

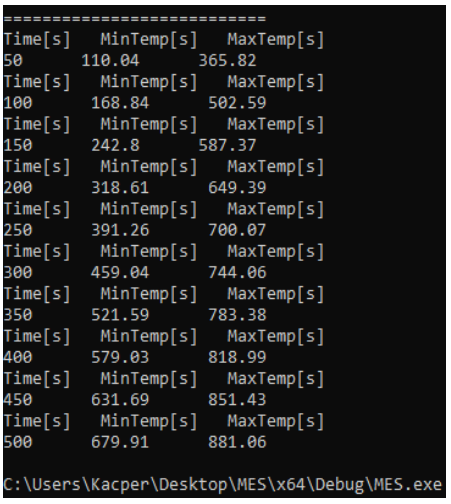
4.1.1 Reference values of minimum and maximum temperatures at nodes for the grid "Test1_4_4.txt".:

Time[s]	MinTemp[s]	MaxTemp[s]
50	110.038	365.815
100	168.837	502.592
150	242.801	587.373
200	318.615	649.387
250	391.256	700.068
300	459.037	744.063
350	521.586	783.383
400	579.034	818.992
450	631.689	851.431
500	679.908	881.058

4.1.2 The obtained minimum and maximum temperature values at nodes for the "Test1_4_4.txt" mesh for the two-, three-, and four-point integration schemes are:



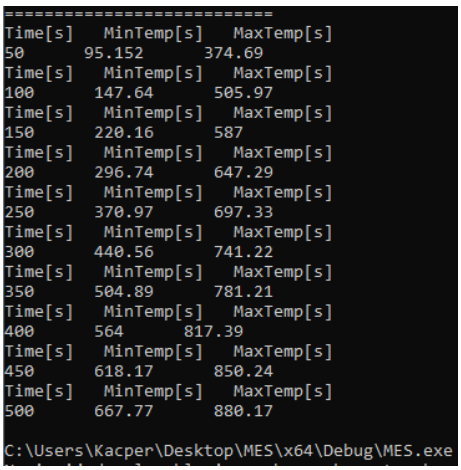




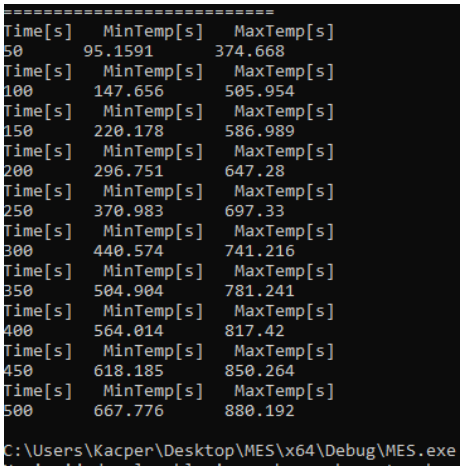
4.2.1 Reference values of minimum and maximum temperature at nodes for the grids "Test2_4_4_MixGrid.txt" and "Test3_31_31.txt":

```
Temperature results

Time = 50 min_T= 95.152, max_T = 374.69

Time = 100 min_T= 147.64, max_T = 505.97

Time = 150 min_T= 220.16, max_T = 587

Time = 200 min_T= 296.74, max_T = 647.29

Time = 250 min_T= 370.97, max_T = 697.33

Time = 300 min_T= 440.56, max_T = 741.22

Time = 350 min_T= 504.89, max_T = 781.21

Time = 400 min_T= 564, max_T = 817.39

Time = 450 min_T= 618.17, max_T = 850.24

Time = 500 min_T= 667.77, max_T = 880.17
```

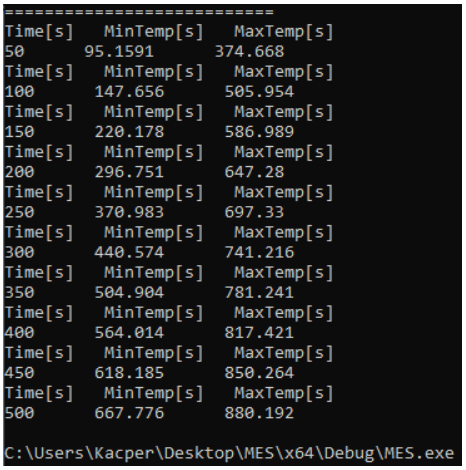
4.2.2 Obtained minimum and maximum temperature values at nodes for the "Test2_4_4_MixGrid.txt" grid using two-, three-, and four-point integration schemes:



```
==========================
Time[s]   MinTemp[s]   MaxTemp[s]
50      95.152      374.69
Time[s]   MinTemp[s]   MaxTemp[s]
100      147.64      505.97
Time[s]   MinTemp[s]   MaxTemp[s]
150      220.16      587
Time[s]   MinTemp[s]   MaxTemp[s]
200      296.74      647.29
Time[s]   MinTemp[s]   MaxTemp[s]
250      370.97      697.33
Time[s]   MinTemp[s]   MaxTemp[s]
300      440.56      741.22
Time[s]   MinTemp[s]   MaxTemp[s]
350      504.89      781.21
Time[s]   MinTemp[s]   MaxTemp[s]
400      564      817.39
Time[s]   MinTemp[s]   MaxTemp[s]
450      618.17      850.24
Time[s]   MinTemp[s]   MaxTemp[s]
500      667.77      880.17

C:\Users\Kacper\Desktop\MES\x64\Debug\MES.exe
```

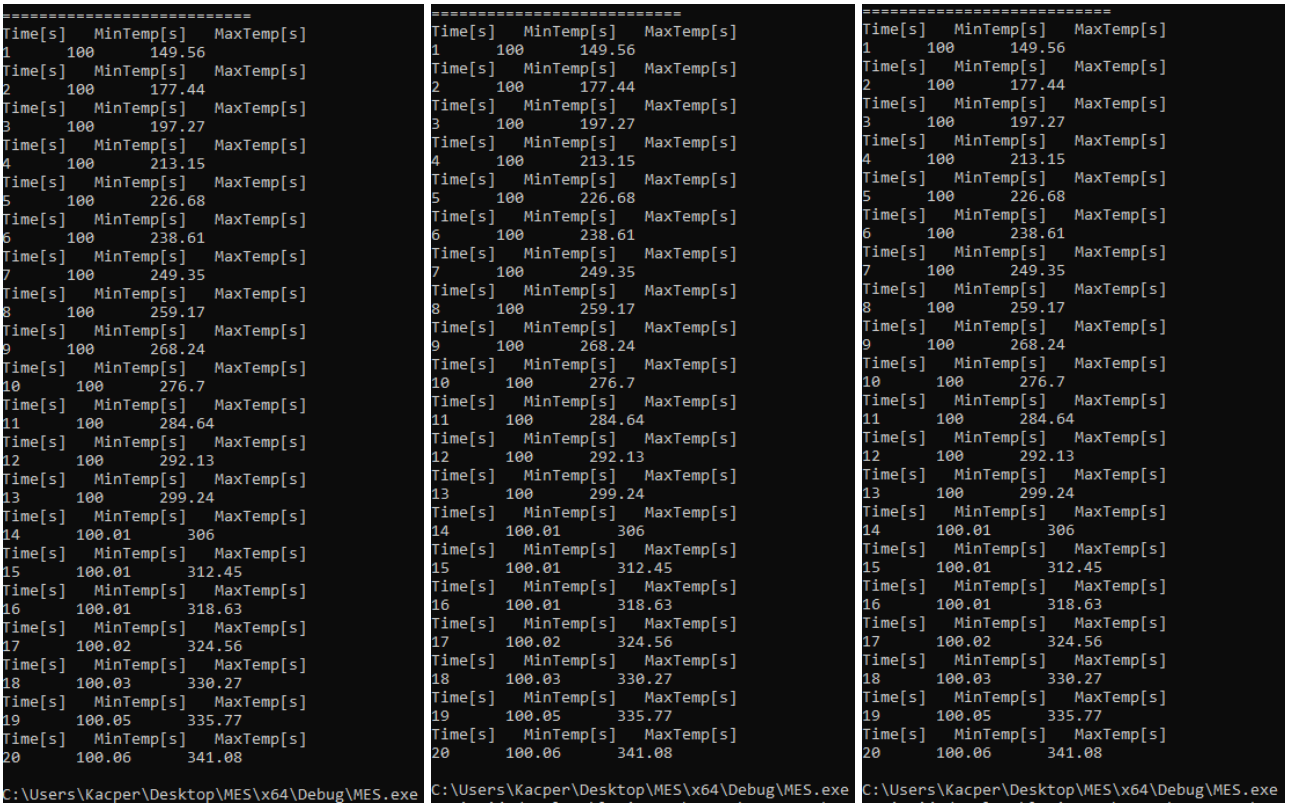


```
==========================
Time[s]   MinTemp[s]   MaxTemp[s]
50      95.1591      374.668
Time[s]   MinTemp[s]   MaxTemp[s]
100      147.656      505.954
Time[s]   MinTemp[s]   MaxTemp[s]
150      220.178      586.989
Time[s]   MinTemp[s]   MaxTemp[s]
200      296.751      647.28
Time[s]   MinTemp[s]   MaxTemp[s]
250      370.983      697.33
Time[s]   MinTemp[s]   MaxTemp[s]
300      440.574      741.216
Time[s]   MinTemp[s]   MaxTemp[s]
350      504.904      781.241
Time[s]   MinTemp[s]   MaxTemp[s]
400      564.014      817.42
Time[s]   MinTemp[s]   MaxTemp[s]
450      618.185      850.264
Time[s]   MinTemp[s]   MaxTemp[s]
500      667.776      880.192

C:\Users\Kacper\Desktop\MES\x64\Debug\MES.exe
```



```
==========================
Time[s]   MinTemp[s]   MaxTemp[s]
50      95.1591      374.668
Time[s]   MinTemp[s]   MaxTemp[s]
100      147.656      505.954
Time[s]   MinTemp[s]   MaxTemp[s]
150      220.178      586.989
Time[s]   MinTemp[s]   MaxTemp[s]
200      296.751      647.28
Time[s]   MinTemp[s]   MaxTemp[s]
250      370.983      697.33
Time[s]   MinTemp[s]   MaxTemp[s]
300      440.574      741.216
Time[s]   MinTemp[s]   MaxTemp[s]
350      504.904      781.241
Time[s]   MinTemp[s]   MaxTemp[s]
400      564.014      817.421
Time[s]   MinTemp[s]   MaxTemp[s]
450      618.185      850.264
Time[s]   MinTemp[s]   MaxTemp[s]
500      667.776      880.192

C:\Users\Kacper\Desktop\MES\x64\Debug\MES.exe
```

4.3 Obtained minimum and maximum temperature values at nodes for the "Test3_31_31.txt" grid for two-, three-, and four-point integration schemes:



```
Time[s]   MinTemp[s]   MaxTemp[s]
1         100          149.56
2         100          177.44
3         100          197.27
4         100          213.15
5         100          226.68
6         100          238.61
7         100          249.35
8         100          259.17
9         100          268.24
10        100          276.7
11        100          284.64
12        100          292.13
13        100          299.24
14        100.01       306
15        100.01       312.45
16        100.01       318.63
17        100.02       324.56
18        100.03       330.27
19        100.05       335.77
20        100.06       341.08

C:\Users\Kacper\Desktop\MES\x64\Debug\MES.exe
```

5. Simulation:

The simulation was performed for a real-life problem, which is an external load-bearing wall in a single-family house. A fragment of the wall with dimensions of 220 mm x 220 mm was used. The layers for which the size would require significantly higher computational power than available were omitted (due to the number of necessary elements). A modified code received from the instructor was used to generate the FEM mesh.

In order to achieve the best simulation representation, modern materials were used. Information on the properties was obtained from the manufacturers' website. This resulted in a three-layered wall built of:

- Blocks of H+H Gold with a width of 175 mm. The declared properties on the manufacturer's website are:
  - The convective heat transfer coefficient: 0,20 $[\frac{W}{m^2*K}]$;
  - Specific heat: 840 $[\frac{J}{kg*K}]$;
  - Density: 385 $[\frac{kg}{m^3}]$;
  - Thermal conductivity: 0,105 $[\frac{W}{m*K}]$;

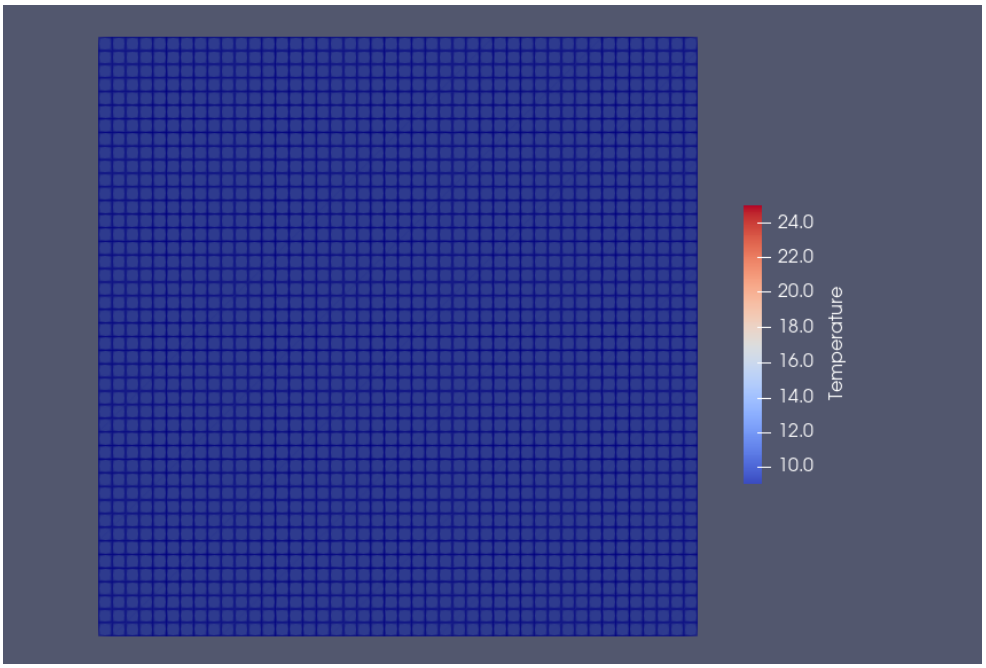
- Rockwool - Frontrock L rock wool lamella board with a width of 300 mm. The declared properties on the manufacturer's website are:
  - The convective heat transfer coefficient: 1[$\frac{W}{m^2*K}$];
  - Specific heat: 750 [$\frac{J}{kg*K}$];
  - Density: 78 [$\frac{kg}{m^3}$];
  - Thermal conductivity: 0,041 [$\frac{W}{m*K}$];

- Planks made of oak wood cut along the fibres. Declared properties are:
  - The convective heat transfer coefficient: 27 [$\frac{W}{m^2*K}$];
  - Specific heat: 2510 [$\frac{J}{kg*K}$];
  - Density: 800 [$\frac{kg}{m^3}$];
  - Thermal conductivity: 0,40 [$\frac{W}{m*K}$];

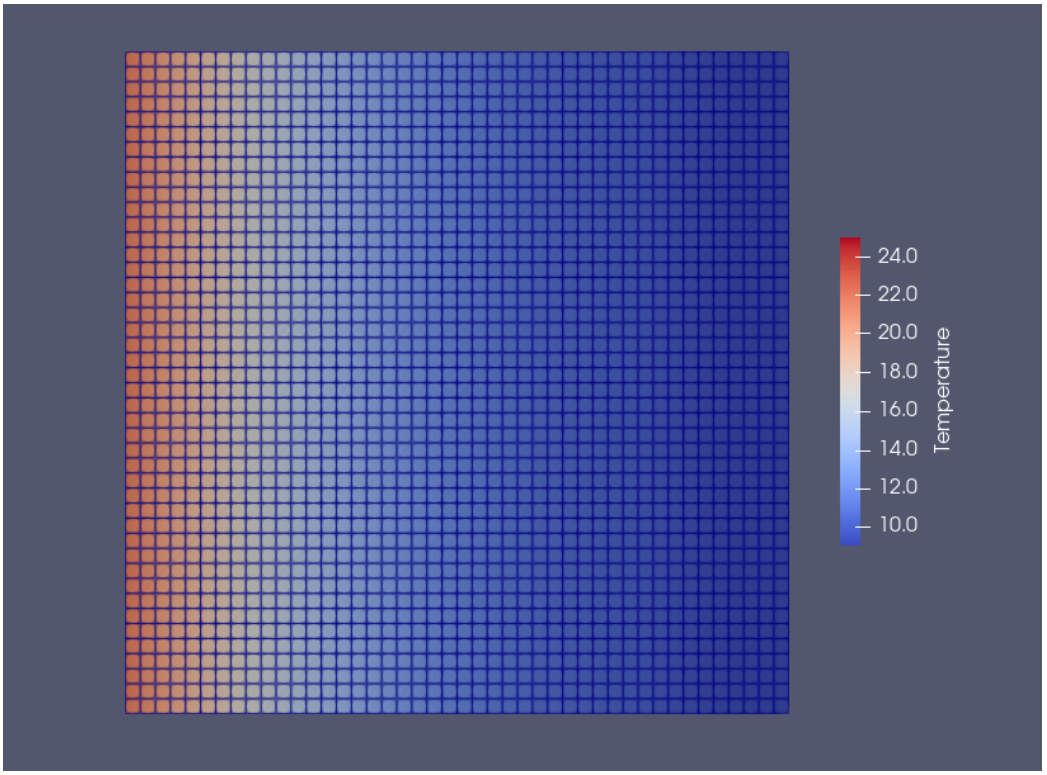
The cost of the above-mentioned materials needed to build a structure with dimensions of 5 x 3 m ranges from 3300 PLN to 4000 PLN depending on the price of wood.

The simulation was carried out for a time step of 12 hours, a simulation time of 30 days, an external temperature of 9 °C, and an internal temperature of 25 °C.

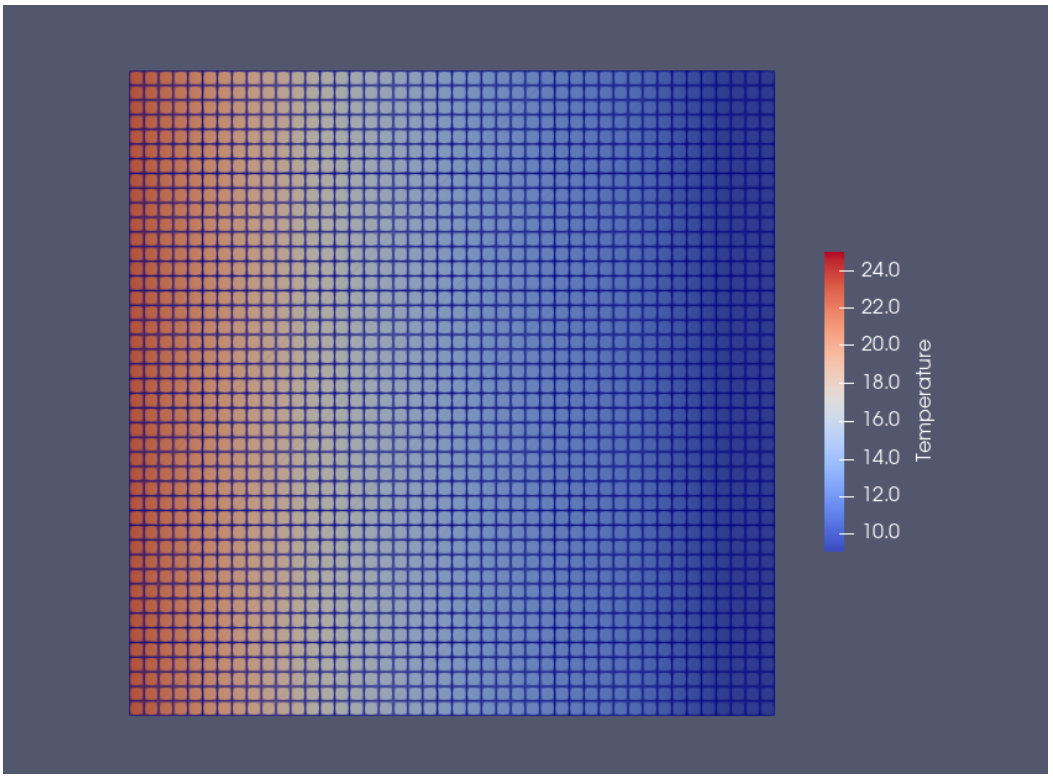
The finite element mesh before starting the simulation:



FEM mesh after 15 days:



FEM mesh after 30 days:



6. Conclusions:

The Finite Element Method provides great possibilities and allows for practical implementation of knowledge gained during previous studies, both in programming, algorithms and subjects related to heat and mass transfer. The developed software can be useful for practical purposes, as exemplified by the created simulation of a load-bearing (external) wall.

Comparing the obtained temperature values with the reference values, it can be inferred that the software operates correctly, despite some deviations that occurred during implementation. Additionally, it can be observed that as the temperature increases, the rate of material heating decreases. This is due to the measurement points - nodes. When their temperature approaches the ambient temperature, it is more difficult to increase it.