Wydział WIMiIP	Imię i nazwisko Zuzanna Będkowska	Rok 3	Grupa 1	Data 18.01.2023
MES	Temat: Symulacja nieustalonych procesów cieplnych przy użyciu oprogramowania MES			Ocena

1. Wstęp teoretyczny:

Implementację algorytmu obliczania temperatury w węzłach siatki rozpoczęto od rozważenia równania Fouriera dla procesu stacjonarnego. Równanie to można zapisać w następujący sposób:

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

gdzie $[H]$ to macierz opisująca transport ciepła między poszczególnymi węzłami siatki, $\{t\}$ to wektor nieznanych temperatur w węzłach, a $\{P\}$ to wektor obciążeń opisujący wpływ temperatury otoczenia na poszczególne węzły siatki elementu. Macierz H dla elementu wyznaczono z następującej zależności:

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

gdzie $k(t)$ to współczynnik przewodności cieplnej, a $[H_{BC}]$ to macierz opisująca transport ciepła wnikającego przez ściany objęte warunkiem brzegowym konwekcji, wyznaczana za pomocą poniższego wzoru:

III. $$[H_{BC}] = \int_S \alpha\{N\}\{N\}^T dS$$

gdzie α to współczynnik wymiany ciepła. Aby zrealizować całkowanie po objętości i powierzchni elementów, zastosowana metodę kwadratur Gaussa-Legendra. Wiedząc, że kwadratury te używane są wyłącznie w przedziale $< -1; 1 >$, każdy z elementów przekształcono do kwadratu o rogach w tym przedziale, stosując macierz przekształcenia i lokalny układ współrzędnych ξ i η. Użyte w obu wzorach $\{N\}$ są wektorami wartości funkcji kształtu w danym punkcie całkowania. Funkcje kształtu zdefiniowano za pomocą następujących wzorów:

IV. $$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$
$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$
$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$
$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$

Macierz przekształcenia elementu ma następującą postać:

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

Wyznacznik tej macierzy zostanie użyty jako dV we wzorze II. W celu wyznaczenia pochodnych $\frac{\partial\{N\}}{\partial x}$ i $\frac{\partial\{N\}}{\partial y}$ dla każdej funkcji kształtu posłużono się układem równań:

VI. $$\begin{bmatrix} \frac{\partial\{N_i\}}{\partial x} \\ \frac{\partial\{N_i\}}{\partial y} \end{bmatrix} = \frac{1}{\det J} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix} \begin{bmatrix} \frac{\partial\{N_i\}}{\partial \xi} \\ \frac{\partial\{N_i\}}{\partial \eta} \end{bmatrix}$$

Utworzone w ten sposób wektory podstawiono do równania II. Otrzymano w ten sposób macierz H w punkcie całkowania, nie uwzględniającą warunku brzegowego Aby otrzymać macierz dla elementu, należy zsumować cząstkowe macierze $[H]$ przemnożone przez iloczyny wag danego punktu całkowania. Znając powyższe zależności i wykorzystując do całkowania kwadratury Gaussa-Legendre'a w 1D, dla elementów z warunkiem brzegowym na przynajmniej jednej ścianie, można wyznaczyć macierz $[H_{BC}]$ podstawiając do równania III wektory funkcji kształtu dla odpowiedniego punktu całkowania tym razem naniesionego na brzeg elementu. Z racji, że kwadratury zwracają poprawne wyniki tylko dla przedziału $< -1; 1 >$, ponownie zastosowano współrzędne lokalne ξ i η, a ściany przekształcono do ścian o długości 2. Jako dS podstawiono połowę długości danego boku elementu. Cząstkowe macierze $[H_{BC}]$ realizujące warunek brzegowy na pojedynczej ścianie, zsumowano do macierzy $\{P\}$. Wektor $\{P\}$ wyznaczono za pomocą poniższego wzoru:

VII. $$\{P\} = \int_S \{N\} \cdot t_{otoczenia} dS$$

Podobnie jak w macierzy $[H_{BC}]$, wektor $\{P\}$obliczany jest wyłącznie dla ścian z warunkiem brzegowym. Używane są te same punkty całkowania naniesione na ścianę, a dSrównież jest połową długości danego boku. Każdą z wyznaczonych wyżej macierzy zagregowano do macierzy globalnej o wymiarze $n \times n$, gdzie n to liczba węzłów siatki.

Aby poszerzyć uzyskane rozwiązanie, o zmianę temperatury przy zmianie kroku czasowego, do równania Fouriera wprowadzono macierz $[C]$ w następujący sposób:

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

Gdzie $[C]$ to macierz opisująca stopień akumulacji energii cieplnej przez węzły, $\Delta\tau$ to krok czasowy a $\{t_0\}$ to wektor temperatur obliczonych w poprzednim kroku czasowym. Macierz $[C]$ wyznaczono z następującej zależności:

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

gdzie c to ciepło właściwe, ρ to gęstość, a $\{N\}$ to wektory wartości funkcji kształtu opisanych w IV w danym punkcie całkowania. Procedura wyznaczania macierzy $[C]$ jest podobna do wyznaczania macierzy $[H]$ - również wykorzystano kwadratury Gaussa-Legendre'a, wykorzystano te same punkty całkowania i ten sam wyznacznik

macierzy przekształcenia. Macierze $[C]$ wyznaczone w punktach całkowania ponownie przemnożono przez iloczyn wag tych punktów i zsumowano do macierzy $[C]$ dla elementu. Tak otrzymane macierze dla elementów zagregowano do macierzy globalnej o wymiarze $n \times n$, gdzie n to liczba węzłów siatki. Globalną macierz $[C]$ podzielono przez $\Delta\tau$ i przygotowano układ równań postaci:

$$[A]\{t\} + \{B\} = 0$$

gdzie:

$$[A] = [H] + \frac{[C]}{\Delta\tau}$$

$$\{B\} = \{P\} - \frac{[C]}{\Delta\tau}\{t_0\}$$

Przygotowany w ten sposób układ równań rozwiązano za pomocą metody eliminacji Gaussa. Aby uzyskać pełną symulację dla określonego przedziału czasowego, obliczenia te należy powtórzyć w każdym kroku czasowym.

2. Implementacja:

W celu implementacji opisanych powyżej zagadnień, wykorzystano następujące struktury:

Global_data - struktura przechowująca dane globalne takie jak czas symulacji (***simulationTime***), krok czasowy (***simulationStepTime***), aktualny czas symulacji (***currentTime***), temperatura otoczenia i początkowa temperatura obiektu (***tot, initialTemperature***) oraz własności materiału, z którego wykonany jest obiekt (***conductivity, alfa, density, specificHeat***). Struktura ta posiada tylko metodę wypisującą (***wypisz()***):

```
struct Global_Data
{
    int simulationTime = 0; //czas symulacji
    int simulationStepTime = 0; //czas jednego kroku symulacji
    int currentTime = 0;
    double conductivity = 0.0; //przewodniosc cieplna
    double alfa = 0.0; //wspolczynnik konwekcyjnej wymiany ciepla
    double tot = 0.0; //temperatura otoczenia
    double initialTemperature = 0.0; //startowa temperatura obiektu
    double density = 0.0; //gestosc obiektu
    double specificHeat = 0.0; //cieplo wlasciwe
    void wypisz();
};
```

Node - struktura opisująca węzeł siatki. Przechowuje informacje o współrzędnych węzła (***x, y***), temperaturze w węźle (***t***), informację o nałożeniu warunku brzegowego na węzeł (***BC***) i metodę wypisującą (***wypisz()***):

```
struct Node
{
    double x = 0.0;
    double y = 0.0; //wspolrzedne
    double t = 0.0; //temperatura
    bool BC = false; //czy na brzegu
    void wypisz();
};
```

Element - struktura reprezentująca element w siatce. Zawiera informacje takie jak numery tworzących ją węzłów (***Node_ID***), własności materiału, z którego zbudowany jest element (***conductivity, density, specificHeat, alfa***), temperaturę otoczenia za najbliższą ścianą siatki (***tot*** - wprowadzona w ten sposób na potrzeby symulacji rzeczywistej, gdzie wykorzystywane są dwie różne temperatury otoczenia). Struktura przechowuje również macierz ***H***, ***HBC*** i ***C***, a także wektor ***P***. Dodatkowo, wyposażono ją w elementy potrzebne do obliczania wyżej wymienionych macierzy takie jak metody wyznaczające macierz Jacobiego (***macierzJacobiego()***), jej wyznacznik (***jacobian()***) i macierz odwrotną (***macierzOdwrotnaJacobiego()***), metodę ***macierzH_macierzC()*** i metetodę ***macierzHBC_wektorP()***:

```
struct Element //tego bedzie duzo
{
    vector <int> Node_ID; //tablica wierzcholkow elementow
    vector <vector <double>> jacobi; //macierz jacobiego, odwracana w trakcie obliczen
    vector <vector <double>> H; //macierz H, pozniej H + HBC
    vector <vector <double>> C; //macierz C
    vector <vector <double>> HBC; //macierz HBC
    vector <vector <double>> P; //Wektor P
    vector <vector <double>> dNdx; //pochodne funkcji kształtu
    vector <vector <double>> dNdy;
    double detJ = 0.0; //wyznacznik macierzy Jacobiego
    double conductivity = 0.0; //własności materiału, wykorzystywane w trakcie obliczeń
    double density = 0.0;
    double specificHeat = 0.0;
    double tot = 0.0;
    double alfa = 0.0;
    Element();
    void wypisz();
    void macierzJacobiego(Element4 & element4,int nr, Grid & grid); //funkcja obliczająca macierz jacobiego
    void jacobian(); //funkcja obliczająca wyznacznik macierzy jacobiego
    void macierzOdwrotnaJacobiego(); //funkcja odwracająca macierz jacobiego
    void macierzH_macierzC(Element4& element4, Grid& grid, Global_Data& data); //odrazu macierz h razy waga, macierzC
    void macierzHBC_wektorP(Element4& element4, Grid& grid, Global_Data& data, Equations & equations);
};
```

Element4 - struktura reprezentująca element uniwersalny. Przechowuje wartości uniwersalne dla elementu czterowęzłowego, takie jak współrzędne punktów całkowania w układzie lokalnym (***ksi***, ***eta***), wagi punktów (***waga***), wartości funkcji kształtu w tych punktach (***CN***) oraz ich pochodne po zmiennych ksi i eta (***macierz_ksi***, ***macierz_eta***). Struktura zawiera również wektor czterech uniwersalnych ścian (***bounds***). W zależności od wybranego schematu całkowania (zmienna ***rzad*** przechowuje informację o wybranej kwadraturze), w konstruktorze generowane są odpowiednie wartości ksi i eta, a następnie na ich podstawie obliczana jest zawartość macierzy ***CN***, ***macierz_ksi*** i ***macierz_eta***. Tworzone są również ściany o odpowiedniej ilości punktów całkowania. Z racji tego, że struktura reprezentuje każdy element przekształcony do postaci uniwersalnej, istnieje tylko jedna instancja tej klasy w programie.

```
struct Element4 //tego tylko 1 instancja, tu sa wartosci tabelaryczne
{
    vector <vector <double>> macierz_ksi; //macierz dNi/ksi
    vector <vector <double>> macierz_eta; //macierz dNi/eta
    vector <vector <double>> CN;
    vector <double> ksi; //vectory wspolrzednych wezlow calkowania
    vector <double> eta;
    vector <double> waga; //wagi wezlow calkowania
    vector <Bound> bounds;
    int rzad = 0; //ilu wezlowa kwadratura
    Element4(int newRzad, int newRzad2); //konstruktor
    void generuj_macierz_ksi(); //funkcje generujace macierze pochodnych
    void generuj_macierz_eta();
    void generuj_macierz_CN();
    void wypisz(); //funkcja wypisujaca wszystkie macierze i vectory
};
```

Bound - struktura reprezentująca ścianę elementu uniwersalnego. Zawiera informację o wybranym schemacie całkowania (***rzad***), współrzędne punktów całkowania w układzie lokalnym (***ksi*** i ***eta***) i wartości funkcji kształtu w tych punktach. Przechowywany jest również numer ściany (***nrS***) - w zależności od wartości tej zmiennej i od wybranej kwadratury, w konstruktorze generowane są odpowiednie współrzędne punktów całkowania. Przyjęto, że ściana 0 to ściana dolna, 1 to prawa, 2 to górna, a 3 to lewa.

```
struct Bound
{
    vector <vector<double>> macierz_N;
    vector <double> ksi;
    vector <double> eta;
    vector <double> waga;
    int rzad = 0;
    int nrS = 0;
    Bound() {};
    Bound(int newRzad, int newNrS);
    void generuj_macierz_N();
};
```

Grid - struktura składująca informacje o siatce. Przechowuje dane takie jak ilość węzłów siatki (***nN***), ilość elementów w siatce (***nE***) oraz tablice wypełnione strukturami węzłów (***Nodes***) i elementów (***Elements***). Posiada również metodę wypisującą (***wypisz()***).

```
struct Grid
{
    int nN = 0; //ilosc wierzcholkow
    int nE = 0; //ilosc elementow
    vector <Node> Nodes; //tablica wezlow
    vector <Element> Elements; //tablica elementow
    void wypisz();
};
```

Equations - jest to struktura przechowująca globalne macierze H (***HG***), C (***CG***) i wektor P (***p***). Struktura zawiera również wektor przechowujący obliczoną w poprzednim kroku temperaturę (***temps0***) i temperaturę obliczaną w kroku bieżącym (***temps1***). Obiekt ten jest odpowiedzialny za agregację macierzy H, C i P pochodzących z poszczególnych elementów (realizują to metody ***agregacjaH***, ***agregacjaC*** i ***agregacjaP***). W metodzie ***solve()*** układ doprowadzany jest do oczekiwanej postaci i rozwiązywany za pomocą prostej implementacji metody Gaussa (***gauss()***).

```
struct Equations
{
    vector<vector <double>> HG;
    vector<vector<double>> CG;
    vector <double> temps0;
    vector <double> temps1;
    vector <vector <double>> p;
    Equations(Global_Data & global_data, Grid & grid);
    void agregacjaH(const Element & element);
    void agregacjaC(const Element& element);
    void agregacjaP(const Element& element);
    void solve();
    void gauss(vector<vector<double>>& wspolczynniki, vector<vector<double>>& wolne, vector<double>& wyniki);
};
```

Obliczenia przeprowadzono w następujący sposób:

```
int main()
{
    Grid grid;
    Global_Data data;
    //read(grid, data);
    generuj_siatke(grid, data);
    Element4 element4(2, 4); //to tyko raz
    Equations equations(data, grid);
    wypisz_macierz(equations.temps0);
    while (data.simulationTime > data.currentTime)
    {
        wyczysc_macierz(equations.HG);
        wyczysc_macierz(equations.CG);
        wyczysc_macierz(equations.p);
        for (auto i : grid.Elements)
        {
            i.macierzH_macierzC(element4, grid, data);
            i.macierzHBC_wektorP(element4, grid, data, equations);
            equations.agregacjaH(i);
            equations.agregacjaC(i);
            equations.agregacjaP(i);
        }
        equations.solve();
        data.currentTime += data.simulationStepTime;
    }
}
```

Siatkę wygenerowano za pomocą funkcji ***generuj_siatke()*** dostosowanej do zagadnienia praktycznego (istnieje możliwość wczytywania danych z dostarczonych plików za pomocą funkcji ***read()***). Na podstawie wygenerowanych danych, utworzono instancje struktur ***Element4*** i ***Equations***. Wypisano również wartość temperatur początkowych w węzłach. Następnie przeprowadzono procedurę obliczeń - w każdym kroku czasowym, dla każdego elementu wyznaczono cząstkowe macierze H, HBC, C i wektor P i agregowano je do macierzy globalnych. Po wygenerowaniu danych dla każdego elementu, następuje przygotowanie układu równań i rozwiązanie w metodzie ***solve()***. Po wydrukowaniu wyniku w metodzie ***solve()***, zwiększany jest krok czasowy, a obliczenia wykonywane są ponownie.

Metody użyte do obliczeń:
Metoda ***macierzH_macierzC()*** służąca do wyznaczania macierzy H i C zgodnie z opisanymi we wstępie teoretycznym wzorami:

```
void Element::macierzH_macierzC(Element4& element4, Grid& grid, Global_Data & data)
{
    for (int j = 0; j < element4.rzad * element4.rzad; ++j)
    {
        wyczysc_macierz(jacobi);
        macierzJacobiego(element4, j, grid);
        jacobian();
        macierzOdwrotnaJacobiego();
        for (int i = 0; i < 4; ++i)
        {
            dNdx[0][i] = element4.macierz_ksi[j][i] * jacobi[0][0] + element4.macierz_eta[j][i] * jacobi[0][1];
            dNdy[0][i] = element4.macierz_ksi[j][i] * jacobi[1][0] + element4.macierz_eta[j][i] * jacobi[1][1];
        }
        vector <vector <double>> dNdxT = T(dNdx);
        vector <vector <double>> dNdyT = T(dNdy);
        vector <vector <double>> mnozeniex = dNdxT * dNdx;
        vector <vector <double>> mnozeniey = dNdyT * dNdy;
        vector <vector <double>> sumaMacierzy = mnozeniex + mnozeniey;
        double iloczyn = conductivity * detJ;
        double waga = element4.waga[j / element4.rzad] * element4.waga[j % element4.rzad];
        double mnoznik = waga * iloczyn;
        vector <vector <double>> Hj = mnoznik * sumaMacierzy;
        H = H + Hj;


        vector <vector <double>> N = { element4.CN[j] };
        vector <vector <double>> NT = T(N);
        vector <vector <double>> CJ = NT * N;
        mnoznik = waga * detJ;
        CJ = mnoznik * CJ;
        C = C + CJ;
    }
    double stala = (density * specificHeat) / (double)data.simulationStepTime;
    C = stala  * C;
}
```

Fragment metody ***macierzHBC_wektorP()*** dla dolnej ściany, wyznaczająca zgodnie z opisanymi we wstępie wzorami, macierz HBC i wektor P dla dolnej ściany elementu:

```
void Element::macierzHBC_wektorP(Element4& element4, Grid& grid, Global_Data& data, Equations & equations)
{
    int bound1 = 0, bound2 = 0;
    if (grid.Nodes[Node_ID[0] - 1].BC && grid.Nodes[Node_ID[1] - 1].BC) //sciana 1
    {
        bound1 = 0;
        bound2 = 1;
        double detj = sqrt(pow((grid.Nodes[Node_ID[bound1] - 1].x - grid.Nodes[Node_ID[bound2] - 1].x), 2)
            + pow((grid.Nodes[Node_ID[bound1] - 1].y - grid.Nodes[Node_ID[bound2] - 1].y), 2))/2.0;
        for (int i = 0; i < element4.bounds[bound1].rzad; ++i)
        {
            vector <double> N = element4.bounds[bound1].macierz_N[i];
            vector <vector <double>> NT = T(N);
            vector <vector <double>> iloczynMacierzy = NT * N;
            iloczynMacierzy = element4.bounds[bound1].waga[i] * iloczynMacierzy;
            iloczynMacierzy = detj * iloczynMacierzy;
            iloczynMacierzy = alfa * iloczynMacierzy;
            HBC = HBC + iloczynMacierzy;
            vector <double> Ppom = element4.bounds[bound1].macierz_N[i];
            vector <vector <double>> PT = T(Ppom);
            //wypisz_macierz(PT);
            PT = alfa * PT;
            PT = detj * PT;
            PT = element4.bounds[bound1].waga[i] * PT;
            PT = tot * PT;
            P = P + PT;
        }
    }
```

Metoda ***solve()*** przygotowująca układ równań do rozwiązania:

```
void Equations::solve()
{
    vector <double> pom(temps0.size(), 0.0);
    vector <vector <double>> wspolczynniki = HG + CG;
    vector <vector <double>> PT = T(p);
    vector <vector <double>> iloczyn = (temps0 * CG);
    vector <vector <double>> wolne = PT + iloczyn;
    gauss(wspolczynniki, wolne, temps1);
    wypisz_macierz(temps1);
    temps0 = temps1;
    temps1 = pom;
}
```

Meotda ***gauss()*** rozwiązująca przygotowane równanie za pomocą metody eliminacji Gaussa i zwracająca wynik w postaci tablicy rozwiązań:

```
void Equations::gauss(vector<vector<double>>& wspolczynniki, vector<vector<double>>& wolne, vector<double>& wyniki)
{
    int n = wspolczynniki.size();
    double mnoznik = 0.0;
    for (int i = 0; i < n - 1; ++i) //petla po wierszach
    {
        for (int j = i + 1; j < n; ++j) //petla po elementach wiersza != 0
        {
            mnoznik = wspolczynniki[j][i] / wspolczynniki[i][i];
            for (int k = i; k < n; ++k)
            {
                wspolczynniki[j][k] -= mnoznik * wspolczynniki[i][k];
            }
            wolne[0][j] -= wolne[0][i] * mnoznik;
        }
    }

    //czesc 2: postepowanie odwrotne
    for (int i = n - 1; i >= 0; --i)
    {
        double skladnik = 0;
        for (int k = i; k < n; k++)
        {
            skladnik += wyniki[k] * wspolczynniki[i][k];
        }
        wyniki[i] = (wolne[0][i] - skladnik) / wspolczynniki[i][i];
    }
}
```

3. Wyniki dla zamieszczonych testów:
   a. dla siatki z pliku ***Test1_4_4***:

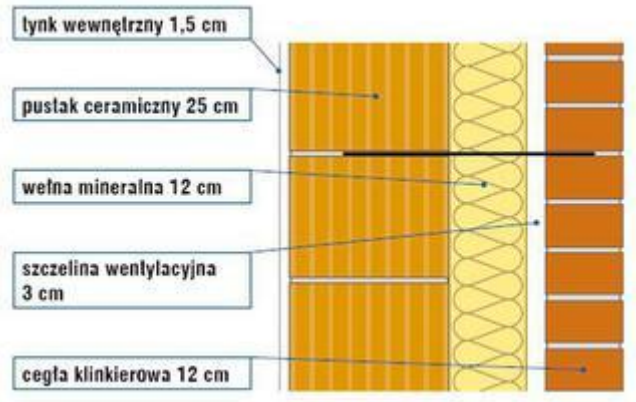
100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
365.815	249.015	249.015	365.815	249.015	110.038	110.038	249.015	249.015	110.038	110.038	249.015	365.815	249.015	249.015	365.815
502.592	353.1	353.1	502.592	353.1	168.837	168.837	353.1	353.1	168.837	168.837	353.1	502.592	353.1	353.1	502.592
587.373	434.597	434.597	587.373	434.597	242.801	242.801	434.597	434.597	242.801	242.801	434.597	587.373	434.597	434.597	587.373
649.387	503.472	503.472	649.387	503.472	318.615	318.615	503.472	503.472	318.615	318.615	503.472	649.387	503.472	503.472	649.387
700.068	564.171	564.171	700.068	564.171	391.256	391.256	564.171	564.171	391.256	391.256	564.171	700.068	564.171	564.171	700.068
744.063	618.775	618.775	744.063	618.775	459.037	459.037	618.775	618.775	459.037	459.037	618.775	744.063	618.775	618.775	744.063
783.383	668.364	668.364	783.383	668.364	521.586	521.586	668.364	668.364	521.586	521.586	668.364	783.383	668.364	668.364	783.383
818.992	713.593	713.593	818.992	713.593	579.034	579.034	713.593	713.593	579.034	579.034	713.593	818.992	713.593	713.593	818.992
851.431	754.921	754.921	851.431	754.921	631.689	631.689	754.921	754.921	631.689	631.689	754.921	851.431	754.921	754.921	851.431
881.058	792.717	792.717	881.058	792.717	679.908	679.908	792.717	792.717	679.908	679.908	792.717	881.058	792.717	792.717	881.058

   b. dla siatki z pliku ***Test2_4_4_MixGrid***:

100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
349.185	231.523	270.038	374.686	231.523	95.1518	122.21	270.038	270.038	122.21	95.1519	231.523	374.686	270.038	231.523	349.185
488.526	329.309	381.088	505.968	329.309	147.644	186.875	381.088	381.088	186.875	147.644	329.309	505.968	381.088	329.309	488.526
577.694	409.886	463.404	586.998	409.886	220.164	262.503	463.404	463.404	262.503	220.164	409.886	586.998	463.404	409.886	577.694
642.959	479.956	530.936	647.286	479.956	296.736	338.038	530.936	530.936	338.038	296.736	479.956	647.286	530.936	479.956	642.959
695.73	542.526	589.691	697.334	542.526	370.968	409.678	589.691	589.691	409.678	370.968	542.526	697.334	589.691	542.526	695.73
741.041	599.111	642.3	741.219	599.111	440.56	476.261	642.3	642.3	476.261	440.56	599.111	741.219	642.3	599.111	741.041
781.21	650.587	690.016	780.668	650.587	504.891	537.614	690.016	690.016	537.614	504.891	650.587	780.668	690.016	650.587	781.21
817.392	697.547	733.528	816.511	697.547	564.002	593.93	733.528	733.528	593.93	564.002	697.547	816.511	733.528	697.547	817.392
850.237	740.445	773.292	849.219	740.445	618.174	645.531	773.292	773.292	645.531	618.174	740.445	849.219	773.292	740.445	850.237
880.168	779.658	809.659	879.118	779.658	667.766	692.772	809.659	809.659	692.772	667.766	779.658	879.118	809.659	779.658	880.168

   c. dla siatki z pliku ***Test3_31_31*** - ze względu na wielkość wydruku, wynik zamieszczono w postaci nagrania z symulacji w programie paraview przeprowadzonej na podstawie obliczonych danych.
4. Symulacja problemu rzeczywistego:

Aby przetestować program, przeprowadzono symulację rozkładu temperatury w ścianie zewnętrznej budynku. Zastosowano uproszczony model ściany przedstawiony na poniższej grafice:



Aby ułatwić modelowanie, pominięto warstwę tynku wewnętrznego (chcąc zastosować przynajmniej 3 elementy na każdą warstwę, przy uwzględnieniu tynku należałoby zastosować elementy o grubości 0,5 cm, co dałoby siatkę o szerokości ponad 100 elementów, której koszt obliczeniowy przerasta użyty do symulacji komputer). Aby nie wprowadzać kolejnego warunku brzegowego konwekcji w ścianie, pominięto szczelinę wentylacyjną. W efekcie otrzymano ścianę o następujących wymiarach:

- Warstwa pustaków ceramicznych o grubości 24 cm
- Warstwa wełny mineralnej o grubości 12 cm
- Warstwa cegły klinkierowej o grubości 12 cm

Do zbudowania modelu siatki użyto elementów kwadratowych o wymiarach 4cm × 4cm. Wybrano fragment ściany o wysokości 48 cm, otrzymując siatkę kwadratową o wymiarach 0,48m × 0,48m, czyli 12 × 12 elementów, 13 × 13 węzłów.

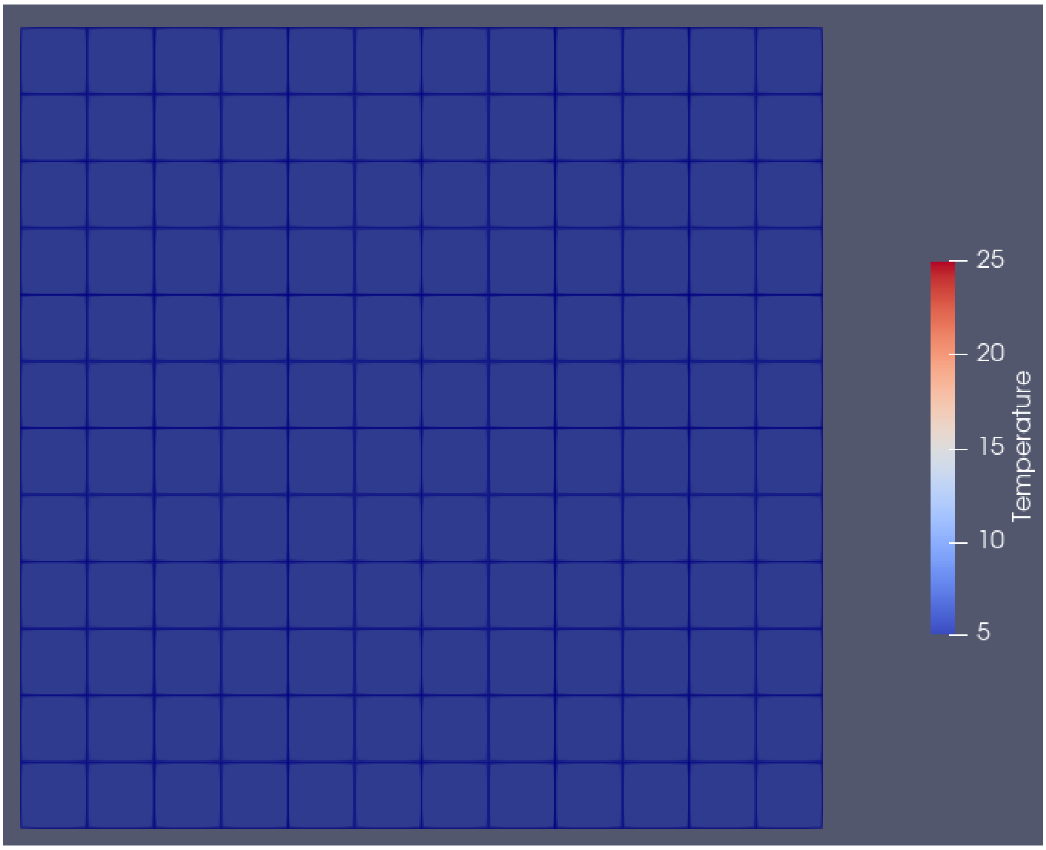
Dodatkowo przyjęto następujące wartości własności materiałowych:

Materiał	Gęstość $[\frac{kg}{m^3}]$	Ciepło właściwe $[\frac{J}{kg*K}]$	Przewodność cieplna $[\frac{W}{m*K}]$
Pustak ceramiczny	1100	840	0,440
Wełna mineralna	60	750	0,05
Cegła klinkierowa	1900	880	1,05

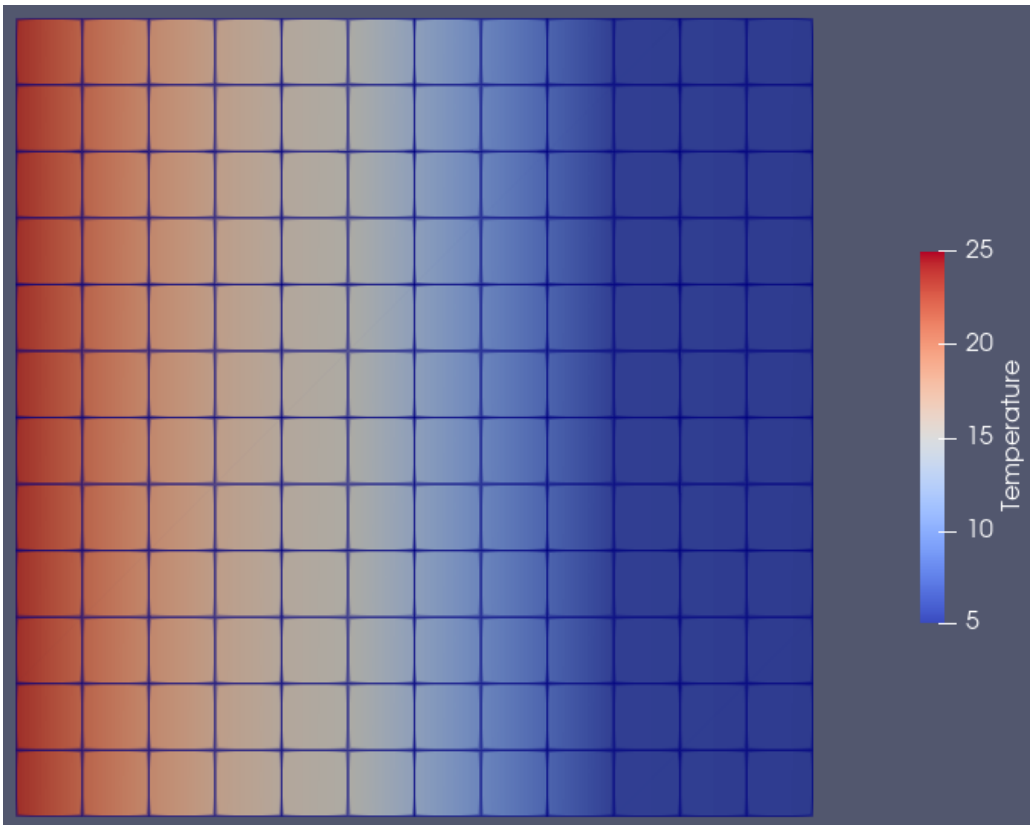
Ze względu na brak danych, za współczynnik α przyjęto 25 $\frac{W}{m^2*K}$. Przyjęto, że temperatura na zewnątrz budynku to 5℃, a wewnątrz to 25℃. Przyjęto czas symulacji 10 dni (864000 sekund), a krok czasowy 1 godzina (3600 sekund). Otrzymano następujące wyniki:
Wyniki dla pierwszego kroku czasowego:

18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006
18.8767	10.065	6.85158	5.68462	5.27434	5.16727	5.08664	5.04314	5.01812	5.00088	5.00037	5.00015	5.00006

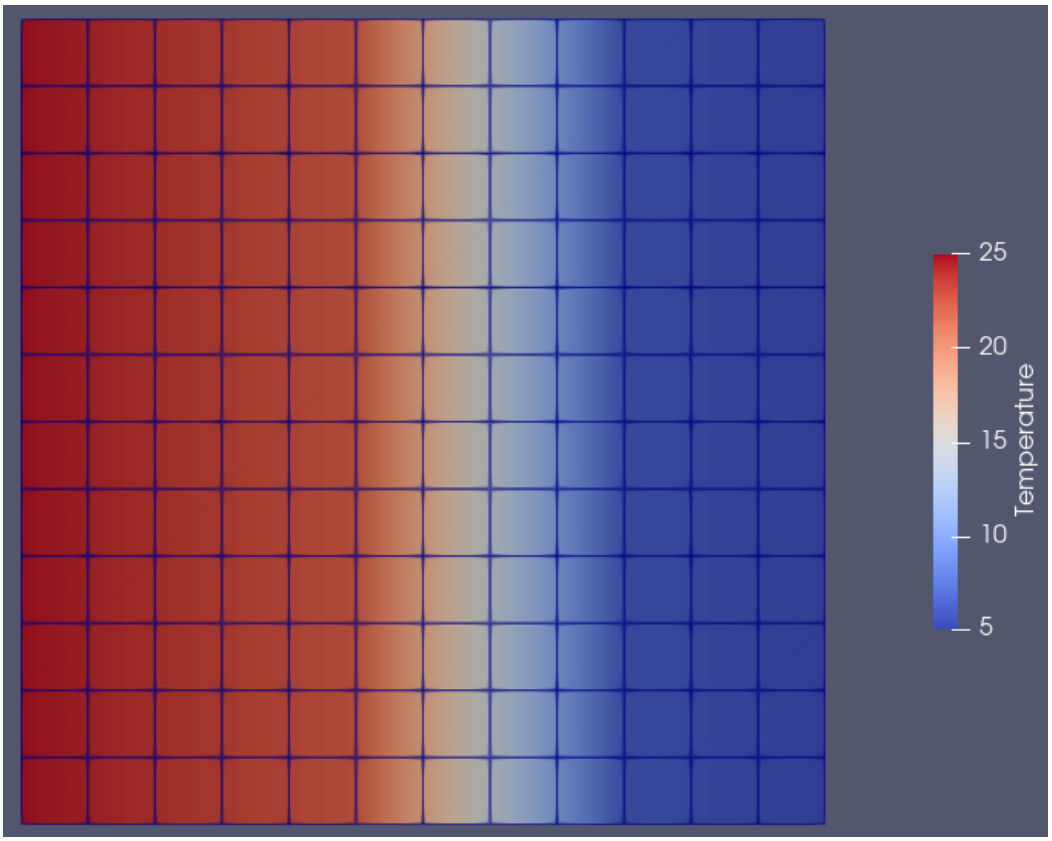
Wizualizacja przed rozpoczęciem symulacji:



Wizualizacja po 12 godzinach:



Wizualizacja po zakończeniu symulacji:



5. Wnioski:

Można uznać, że w trakcie zajęć uzyskano program działający poprawnie i wykonujący obliczenia w zadowalająco dokładny sposób - wyniki dla przypadków testowych obliczone przez program zgadzają się z wynikami dostarczonymi przez prowadzącego zajęcia. Również symulacja problemu rzeczywistego dała zadowalające efekty. Patrząc na rozkład temperatur w ścianie po zakończeniu symulacji, można wyodrębnić 3 warstwy, z których zbudowany jest obiekt, co jest zgodne z oczekiwaniami. Otrzymane warstwy charakteryzują się innymi właściwościami materiałowymi, a co za tym idzie inaczej przewodzą ciepło, co jest widoczne w trakcie symulacji. Warto również podkreślić, że warstwy pokrywają się z załączonym schematem ściany.

Dzięki zastosowaniu metody elementów skończonych, w łatwy sposób rozwiązano problem wyznaczenia temperatury w każdym punkcie danego obiektu i symulacji procesu cieplnego w którym udział bierze badany obiekt. Metoda elementów skończonych umożliwia podział ogromnego zadania na znacznie łatwiejsze do policzenia (i zwracające dokładniejsze wyniki) części, co niestety może wiązać się ze zwiększającym się czasem przeprowadzania obliczeń. Aby właściwie korzystać z MES, należy pamiętać o rozważnym wyborze gęstości siatki, o czym przekonano się w zadaniu praktycznym, gdy przy siatce o szerokości 100 elementów obliczenia stały się niemożliwe do wykonania, a przy siatce niemal ośmiokrotnie mniejszej otrzymano zadowalające wyniki. W omawianym problemie, istotny okazał się również dobór schematu całkowania - chociaż większa ilość kwadratur wydawałaby się dokładniejsza, otrzymano niemalże identyczne wyniki dla mniejszej ilości punktów całkowania, co pozwoliło znacznie zmniejszyć czas wykonania programu. Aby uzyskać program działający jeszcze sprawniej, należałoby przeprowadzić refaktoryzację kodu i usunąć zbędne i powtarzające się operacje. Mając jednak na uwagę złożoność obliczeniową metody eliminacji Gaussa i złożone obliczenia dla każdego elementu, należy się liczyć z tak długim czasem wykonania dla wielkich siatek.