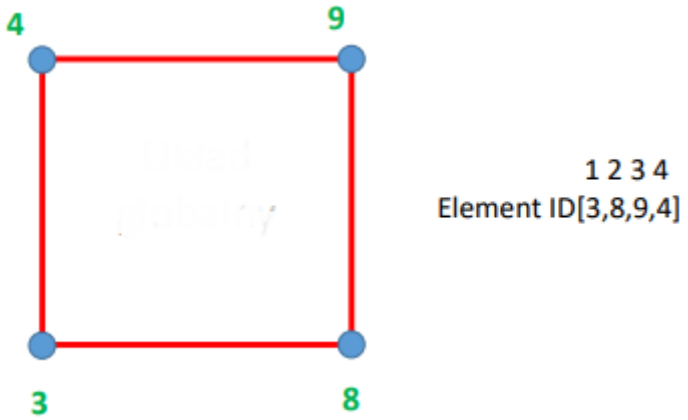
Łukasz Oprych ITE Gr. 5	Temat: Symulacja nieustalonego procesu cieplnego oprogramowaniem MES	Data 11.01.2024r.

Wstęp teoretyczny:

Metoda Elementów Skończonych to metoda numeryczna do rozwiązywania równań różniczkowych wykorzystywana w zastosowaniach inżynierskich do przeprowadzania symulacji, np. przepływu ciepła, naprężeń. Ideą MES jest zamiana dowolnej ciągłej wartości na model dyskretny. W celu przeprowadzenia symulacji na badanym obiekcie stosuje się siatkę MES. Siatka ta dzieli obiekt na dużą ilość bardzo małych elementów skończonych stworzonych z węzłów, które łączone są w ściany.



Przykładowy element siatki MES

W kwestii rozwiązywanego problemu w oprogramowaniu, poza opracowaniem całkowania numerycznego metodą Gaussa, obliczany jest transport ciepła za pomocą równania Fouriera-Kirchoffa w stanie nieustalonym z wykorzystaniem konwekcyjnego warunku brzegowego, który jest zadany wzorem:

$$q = \alpha(t - t_{amb})$$

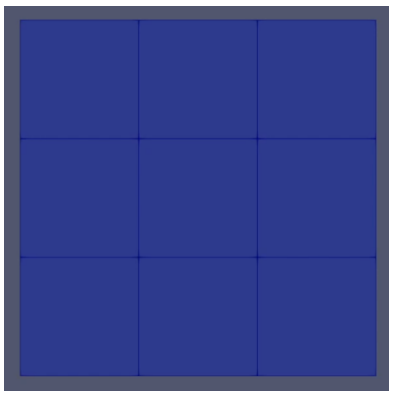
q – strumień ciepła, α – współczynnika przejmowania ciepła, t – temperatura powierzchni materiału, t_{amb} – temperatura otoczenia.

Konwekcyjny warunek brzegowy określa prędkość przepływu płynu na granicy obszaru, w którym rozważamy równanie różniczkowe.

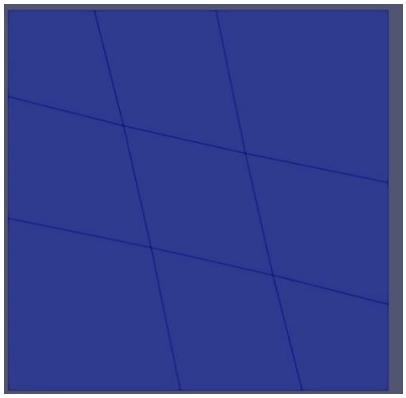
Opis modelu warunku brzegowego:

W projekcie testowano dwuwymiarowe siatki MES, które miały różną ilość elementów czterowęzłowych oraz różne kształty.

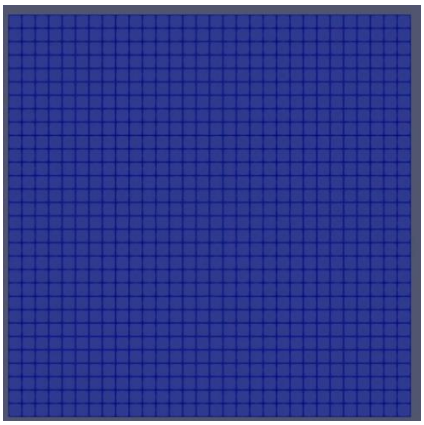
Siatka 4x4 w kształcie kwadratu 9 elementowa, z 16 węzłami.



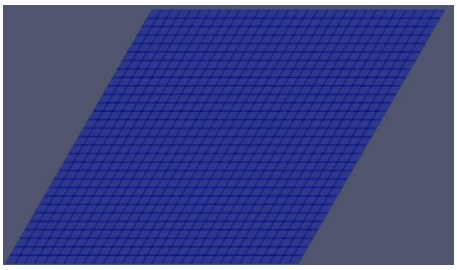
Siatka 4x4 w kształcie kwadratu z elementami o różnych kształtach, 9 elementowa z 16 węzłami



Siatka 31x31 w kształcie kwadratu, 900 elementowa z 961 węzłami



Siatka 31x31 w kształcie trapezu 900 elementowa z 961 węzłami



Na węzłach znajdujących się na brzegach każdej z siatek mamy do czynienia z warunkiem brzegowym, jest to miejsce, gdzie energia wchodzi w układ, w tym przypadku cieplna. Dla elementów, których krawędzie mają warunek brzegowy (na obu węzłach krawędzi mamy warunek brzegowy) w celu analizy wpływu konwekcji oraz temperatury otoczenia liczymy wektory P oraz Macierze HBC.

Wektor P – to wektor obciążeń (strumieni cieplnych) opisujący wpływ temperatury otoczenia na węzły elementu siatki.

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

Macierz HBC – to uzupełnienie [H] o część warunku brzegowego konwekcji. [HBC] opisuje transport ciepła, które wnika przez ściany objęte warunkiem brzegowym konwekcji.

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

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

Zgrubny opis MES:

Początkowo implementowanie rozwiązania było rozważane dla równania Fouriera dla procesu ustalonego. Można przedstawić je za pomocą układu równań w postaci macierzowej:

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

W którym $[H]$ jest macierzą, która opisuje wymianę energii cieplnej pomiędzy poszczególnymi węzłami siatki w materiale, $\{t\}$ to wektor wartości temperatur na węzłach, $\{P\}$ wektor obciążeń reprezentujący strumienie ciepła.

Wzór na macierz H:

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

k(t) – współczynnik przewodności cieplnej, $\left\{ \frac{\partial \{N\}}{\partial x} \right\}$ jeden z wektorów pochodnych funkcji kształtu, $[H_{BC}]$ to jak wcześniej wspomniano to macierz opisująca transport ciepła wnikającego przez ściany objęte warunkiem brzegowym konwekcji

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

α – współczynnik wymiany energii cieplnej materiału, *{N}* – wektor wartości funkcji kształtu

W celu uzyskania między innymi składowych macierzy H, potrzebnym było wyliczenie wektorów funkcji kształtu dla danych punktów całkowania *{N}* oraz wykonanie całkowania po powierzchni i objętości, stosując metodę kwadratur Gaussa-Legendre'a. Są to kwadratury interpolacyjne, w których całkę aproksymujemy wielomianem interpolacyjnym w przedziale $< -1; 1 >$. W tym celu elementy przekształcamy do kwadratu z węzłami w owym przedziale, stosując macierz przekształcenia Jacobiego oraz układ lokalny współrzędnych ξ oraz η.

Wzory na funkcję kształtu w kolejności od dolnej krawędzi elementu:

$$N_1 = 0{,}25\,(1 - \xi)(1 - \eta)$$

$$N_2 = 0{,}25\,(1 + \xi)(1 - \eta)$$

$$N_3 = 0{,}25\,(1 + \xi)(1 + \eta)$$

$$N_4 = 0{,}25\,(1 - \xi)(1 + \eta)$$

Macierz przekształcenia Jacobiego dla danego elementu:

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

Po obliczeniu wyznacznika tej macierzy możemy wyznaczyć pochodne do wektorów $\left\{\frac{\partial\{N\}}{\partial x}\right\}, \left\{\frac{\partial\{N\}}{\partial y}\right\}$ do wzoru na *[H]* za pomocą wzoru:

$$\begin{bmatrix} \frac{\partial\{N_i\}}{\partial x} \\ \frac{\partial\{N_i\}}{\partial y} \end{bmatrix} = \frac{1}{detJ} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix} \begin{bmatrix} \frac{\partial\{N_i\}}{\partial \xi} \\ \frac{\partial\{N_i\}}{\partial \eta} \end{bmatrix}$$

Następnie po otrzymaniu danych wektorów można obliczyć macierz H jeszcze bez uwzględnienia części z warunkiem brzegowym. Macierz dla danego elementu uzyskujemy sumując cząstkowe macierze i mnożąc je przez iloczyn wag pod konkretny punkt całkowania.

Przykład dla 2-pkt schematu całkowania:

$$H = H_{pc1} * w_1 * w_1 + H_{pc2} * w_2 * w_1 + H_{pc3} * w_1 * w_2 + H_{pc4} * w_2 * w_2$$

Konkretne wartości punktów całkowania oraz wagi zależą od schematu całkowania zgodnie z tabelą kwadratur Gaussa-Legendre'a:

Number of points, n	Points, x_i	Weights, w_i
1	0	2
2	$\pm\sqrt{\frac{1}{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18+\sqrt{30}}{36}$
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18-\sqrt{30}}{36}$

Następnie w celu wyznaczenia macierzy $[H_{BC}]$ trzeba było dokonać całkowania kwadraturą Gaussa-Legendre'a w 1D w elementach, których krawędzie ścian miały warunek brzegowy. Do równania na macierz wstawiamy wektory funkcji kształtu dla punktów całkowania umieszczonych na brzegach elementu. Następnie, w celu policzenia dS w układzie lokalnym obliczono jakobian przekształcenia ze wzoru:

$$detJ = \frac{\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}}{2}$$

Następnie cząstkowe $[H_{BC}]$ zsumowano do macierzy dla danych elementów.

Kolejnym krokiem było wyznaczenie wektora obciążeń $\{P\}$ ze wzoru:

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

α – współczynnik wymiany energii cieplnej materiału, $\{N\}$ – wektor wartości funkcji kształtu,

t_{ot} – temperatura otoczenia

Całkowanie dla wektora $\{P\}$ zrealizowano na tej samej zasadzie co dla Macierzy $[H_{BC}]$

W celu wykonania prawidłowych obliczeń, wszystkie macierze zagregowano do macierzy wymiaru ilość węzłów x ilość węzłów siatki.

Równanie dla procesu ustalonego w przypadku tworzonego oprogramowania nie spełnia wszystkich wymagań, ponieważ realizujemy je tylko w procesie stacjonarnym i przez owe braki nie można wykonać symulacji zmian temperatury na węzłach w czasie.

Realizujemy to za pomocą równania dla procesu niestacjonarnego:

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

Δ τ – krok czasowy

$\{t_0\}$ – temperatura początkowo w kroku 1, w kolejnych temperatura dla poprzedniego $\Delta\tau$

$\{t_1\}$ – temperatura obliczona

[C] – macierz opisująca ilość możliwej do zgromadzenia energii cieplnej przez węzeł siatki.

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

c – ciepło właściwe, ρ - gęstość materiału

W celu wyznaczenia macierzy [C] wykorzystujemy takie same jak do macierzy [H] kwadratury Gaussa-Legendre'a, punkty całkowania, wyznacznik macierzy przekształcenia. Następnie macierze C dla danych punktów całkowania mnożymy przez iloczyn wag dla danych punktów całkowania, następnie dla danego elementu zsumowano. Otrzymane macierze C zagregowano w ten sam sposób jak macierz H, z czego powstała globalna macierz C. Zgodnie z wcześniej podanym równaniem macierz C podzielono przez krok czasowy. W celu wykonania symulacji czasowej, wymagane jest obliczenie {t1} dla każdego kroku czasowego, w tym celu należy rozwiązać układ równań w postaci Ax+B = 0, który rozwiązujemy za pomocą metody eliminacji Gaussa.

Składowe układu równań:

$A = [H] + \frac{C}{\Delta\tau}$, $B = -(\frac{C}{\Delta\tau}\{t_0\} + \{P\})$

$x = \{t_1\}$

Zgrubna charakterystyka kodu:

GlobalData to struktura, przechowująca dane niezbędne do wykonania symulacji transportu ciepła. Mówi ona właściwościach materiału oraz parametrach symulacji.

```
struct GlobalData {
    int simulationTime; //czas symulacji
    int simulationStepTime; //krok czasowy
    int conductivity; // przewodnosc
    int alfa; //wspolczynnik konwekcyjnej wymiany ciepla
    int tot; //temperatura otoczenia
    int initialTemp; //temperatura poczatkowa
    int density; //gestosc
    int specificHeat; //cieplo wlasciwe
    int nodesNumber; //ilosc wezlow
    int elementsNumber; // ilosc elementow
};
```

Struktura DnDxDnDy odpowiada za przechowywanie danych dla danego elementu, takie jak: pochodne funkcji kształtu, współrzędne węzłów, wyznacznik macierzy Jakobiego dla całkowania 1D i 2D oraz wskazanie na to czy krawędź elementu jest brzegową.

```
struct DnDxDnDy {
    double** dnDx; //pochodne funkcji ksztaltu
    double** dnDy;
    double* x; //wspolrzedne wezla
    double* y;
    double* detJ; //wyznacznik jakobianu
    int* boundaryEdges; //warunek brzegowy
    double* detJ1D; //wyznacznik jakobianu do calkowania 1D
};
```

Struktura shapeFactors odpowiada za przechowywanie danych o elemencie uniwersalnym, takie jak: pochodne funkcji kształtu, wartości funkcji kształtu, funkcje kształtu dla węzłów elementu objętych warunkiem brzegowym, ilość węzłów, ilość punktów całkowania, punkty całkowania dla krawędzi objętych warunkiem brzegowym, oraz wagi dla całkowania 1D i 2D.

```
struct shapeFactors {
    double** eta_array; //pochodna funkcji ksztaltu po eta
    double** ksi_array;// pochodna funkcji ksztaltu po ksi
    double** shape_func;  //funkcje ksztaltu
    double*** shape_func_boundary; //funkcje ksztaltu do obliczen z uwzgledenieniem BC
    int nodes = 4;
    int points; //pc
    int edge_points; //pc warunek brzegowy
    vector <double> w_array; //wagi
    vector <double> w_array1D; //wagi do calkowania 1D
};
```

Funkcja calculateDEtaDKsi odpowiada za zwracanie obliczonych danych dla struktury shapeFactors dla danego schematu całkowania.

Fragmenty kodu:

```
struct shapeFactors calculateDEtaDKsi (int nodes, int n) {
    int points; // pc
    double** eta_array; //dn/deta
    double** ksi_array; //dn/dksi
    vector <double> w_array; //wagi
    vector <double> w_array1D;

    double** shape_func;
    double*** shape_func_boundary = new double** [4]; //funkcje ksztaltu do obliczen z uwzgledenieniem BC

    vector <double> eta; // wspolrzedne pc 2D
    vector <double> ksi;
    vector <double> eta_boundary[4]; // wspolrzedne pc 1D
    vector <double> ksi_boundary[4];

    double a; //wspolczynniki
    double b;
    double w1; //wagi
    double w2;

    int edgePoints; //wspolrzedne punktow z warunkiem brzegowym

    switch (n) {
```

Przykład dla schematu trójpunktowego całkowania:

```
case 2:
    points = 9;
    a = sqrt( x: 3./5);
    edgePoints = 3;
    eta.insert ( position: eta.end(), {-a, -a, -a, 0, 0, 0, a, a, a});
    ksi.insert  ( position: ksi.end(), {-a, 0, a, -a, 0, a, -a, 0, a});

    eta_boundary[0].insert( position: eta_boundary[0].end(), {-1, -1, -1});  //wspolrzedne dla dolnej krawedzi
    ksi_boundary[0].insert( position: ksi_boundary[0].end(), {-a, 0, a});
    eta_boundary[1].insert( position: eta_boundary[1].end(), {-a, 0, a });
    ksi_boundary[1].insert( position: ksi_boundary[1].end(), {1, 1, 1});  //wspolrzedne dla prawej krawedzi
    eta_boundary[2].insert( position: eta_boundary[2].end(), {1, 1, 1});
    ksi_boundary[2].insert( position: ksi_boundary[2].end(), {a, 0, -a}); //gorna
    eta_boundary[3].insert( position: eta_boundary[3].end(), {-a, 0, a});
    ksi_boundary[3].insert( position: ksi_boundary[3].end(), {-1, -1, -1}); //lewa

    w1 = 5./9;
    w2 = 8./9;

    w_array.insert( position: w_array.end(), {w1 * w1, w2 * w1, w1 * w1, w1 * w1, w1 * w2, w2 * w2, w1 * w2, w1 * w1, w2 * w1, w1 * w1}); //wagi zielone punkty

    w_array1D.insert( position: w_array1D.end(), {w1, w2, w1}); //wagi na krawedzi

    break;
```

Obliczanie funkcji kształtu i ich pochodnych:

```
for (int j = 0; j < points; ++j) {
    shape_func [0][j] = 0.25 * (1 - eta[j])*( 1 - ksi[j] ); //N1
    eta_array[0][j] = -0.25 *  (1 - ksi[j]); // dN1/deta
    ksi_array[0][j] = -0.25 * (1 - eta[j]); // dN1/dksi
    shape_func [1][j] = 0.25 * (1 - eta[j])*( 1 + ksi[j] ); // N2
    eta_array[1][j] = -0.25 *  (1 + ksi[j]); //dN2/deta
    ksi_array[1][j] = 0.25 * (1 - eta[j]); //dN2/dksi
    shape_func [2][j] = 0.25 * (1 + eta[j])*( 1 + ksi[j] ); // N3
    eta_array[2][j] = 0.25 *  (1 + ksi[j]); //dN3/deta
    ksi_array[2][j] = 0.25 * (1 + eta[j]); //dN3/dksi
    shape_func[3][j] = 0.25 * (1 + eta[j])*( 1 - ksi[j] ); // N4
    eta_array[3][j] = 0.25 *  (1 - ksi[j]); //dN4/deta
    ksi_array[3][j] = -0.25 * (1 + eta[j]);//dN4/dksi
}

for (int i = 0; i < 4; ++i) {
    shape_func_boundary[i] = new double* [edgePoints];
    for (int j = 0; j < edgePoints; ++j) {
        shape_func_boundary[i][j] = new double [4];
        shape_func_boundary [i][j][0] = 0.25 * (1 - eta_boundary[i][j])*( 1 - ksi_boundary[i][j] );//N1 BC
        shape_func_boundary [i][j][1] = 0.25 * (1 - eta_boundary[i][j])*( 1 + ksi_boundary[i][j] );//N2
        shape_func_boundary [i][j][2] = 0.25 * (1 + eta_boundary[i][j])*( 1 + ksi_boundary[i][j] );//N3
        shape_func_boundary [i][j][3] = 0.25 * (1 + eta_boundary[i][j])*( 1 - ksi_boundary[i][j] );//N4
    }
}
```

Funkcja wykonująca obliczenia dla parametrów struktury DnDxDnDy takie jak: wyznaczanie warunku brzegowego, wyznaczanie punktów całkowania, liczenie Jakobianów 1D i 2D oraz ich wyznaczników, pochodne funkcji kształtu potrzebne do całkowania macierzy H. Parametr factors odpowiada za przechowywanie informacji dla elementu uniwersalnego.

Fragment kodu:

```
struct DnDxDnDy* calculateDnDxDnDy(double** points, int** elements, struct shapeFactors factors, struct GlobalData gd, int* BC){
    struct DnDxDnDy* results = new struct DnDxDnDy [gd.elementsNumber];
    for (int i = 0; i < gd.elementsNumber; ++i) {
        results[i].dnDx = new double* [factors.points];
        results[i].dnDy = new double* [factors.points];
        results[i].x = new double [factors.points]; //x dla punktu calkowania
        results[i].y = new double [factors.points];
        results[i].detJ = new double[factors.points];
        results[i].detJ1D = new double [4];
        results[i].boundaryEdges = new int [4]; //prawdziwe tylko dla czworokątnych elementów
        double* x = new double [factors.nodes];
        double* y = new double [factors.nodes];
        for (int j = 0; j < factors.nodes; ++j) { //przepisanie wspolrzednych wezlow w elemencie
            x[j] = points[elements[i][j]][0];
            y[j] = points[elements[i][j]][1];// j-ty punkt w i-tym elemencie
        }

        //wyznaczanie bc na danej krawedzi
        if( BC[elements[i][0]] == 1 && BC[elements[i][1]] == 1 ){
            results[i].boundaryEdges[0] = 1;
        }
```

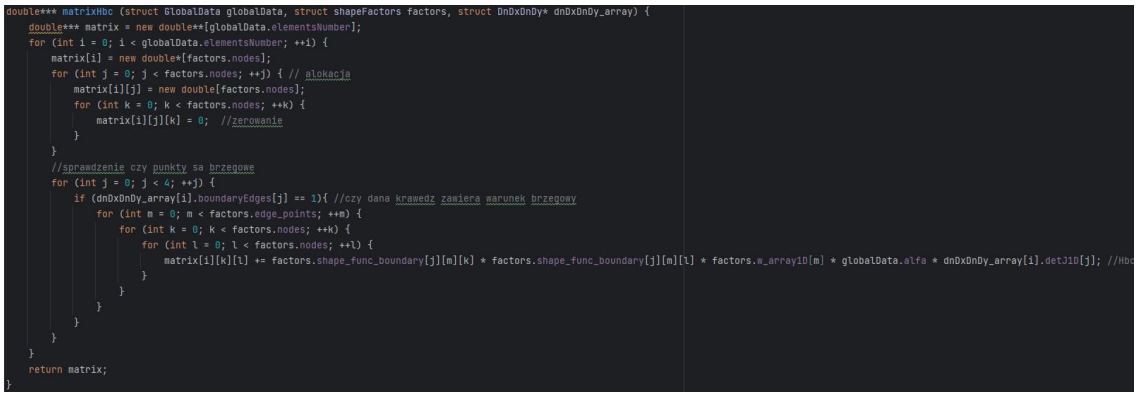
Funkcja matrixH() służy do wyznaczania macierzy H dla elementów 4 węzłowych, zgodnie z wzorami zawartymi w opisie MES, składowe wykorzystujemy z wcześniej wykorzystanych funkcji, struktur oraz wczytanych z pliku danych:

```
double*** matrixH (struct GlobalData globalData, struct shapeFactors factors,struct DnDxDnDy* dnDxDnDy_array){
    double*** matrix = new double**[globalData.elementsNumber];
    for (int i = 0; i < globalData.elementsNumber; ++i) {
        matrix[i] = new double*[factors.nodes];
        for (int j = 0; j < factors.nodes; ++j) {
            matrix[i][j] = new double[factors.nodes]; //alokacja
            for (int k = 0; k < factors.nodes; ++k) {
                matrix[i][j][k] = 0; //zerowanie
            }
        }
        for (int j = 0; j < factors.points; ++j) {
            for (int k = 0; k < factors.nodes; ++k) {
                for (int l = 0; l < factors.nodes; ++l) {
                    matrix[i][k][l] += (dnDxDnDy_array[i].dnDx[j][k] * dnDxDnDy_array[i].dnDx[j][l] + dnDxDnDy_array[i].dnDy[j][k] * dnDxDnDy_array[i].dnDy[j][l]) * dnDxDnDy_array[i].detJ[j] * factors.w_array[j] * globalData.conductivity
                }
            }
        }
    }
    return matrix;
}
```

Funkcja C() służy do wyznaczania macierzy C dla elementów 4 węzłowych, zgodnie z wzorami:

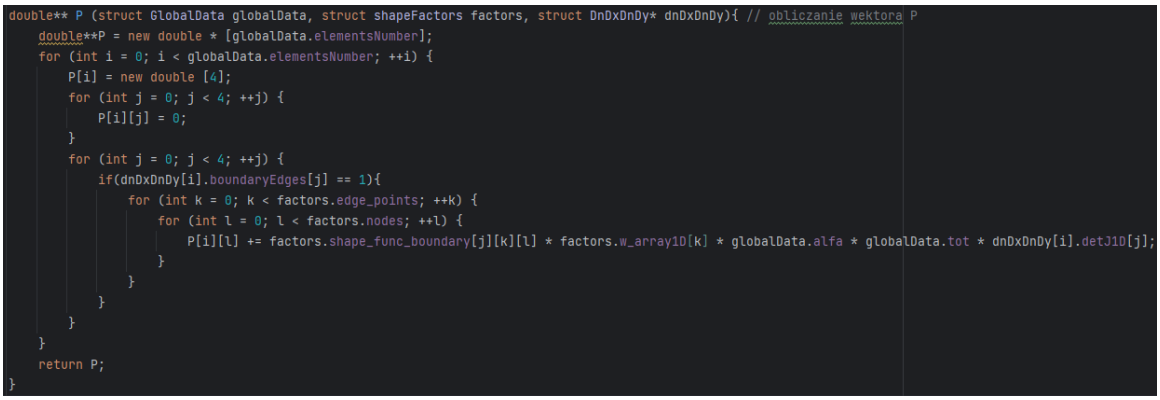
```
double*** C (struct GlobalData globalData, struct shapeFactors factors, struct DnDxDnDy* dnDxDnDy_array){ //macierz C
    double*** matrix = new double**[globalData.elementsNumber];
    for (int i = 0; i < globalData.elementsNumber; ++i) {
        matrix[i] = new double*[factors.nodes];
        for (int j = 0; j < factors.nodes; ++j) {
            matrix[i][j] = new double[factors.nodes];
            for (int k = 0; k < factors.nodes; ++k) {
                matrix[i][j][k] = 0;
            }
        }
        for (int j = 0; j < factors.points; ++j) {
            for (int k = 0; k < factors.nodes; ++k) {
                for (int l = 0; l < factors.nodes; ++l) {
                    matrix[i][k][l] += factors.shape_func[k][j] * factors.shape_func[l][j] * globalData.density * globalData.specificHeat * dnDxDnDy_array[i].detJ[j] * factors.w_array[j];
                }
            }
        }
    }
    return matrix;
}
```

Funkcja matrixHBC() służąca do wyznaczania macierzy HBC zgodnie z wzorami:



```
double*** matrixHbc (struct GlobalData globalData, struct shapeFactors factors, struct DnDxDnDy* dnDxDnDy_array) {
    double*** matrix = new double**[globalData.elementsNumber];
    for (int i = 0; i < globalData.elementsNumber; ++i) {
        matrix[i] = new double*[factors.nodes];
        for (int j = 0; j < factors.nodes; ++j) { // alokacja
            matrix[i][j] = new double[factors.nodes];
            for (int k = 0; k < factors.nodes; ++k) {
                matrix[i][j][k] = 0;  //zerowanie
            }
        }
        //sprawdzenie czy punkty sa brzegowe
        for (int j = 0; j < 4; ++j) {
            if (dnDxDnDy_array[i].boundaryEdges[j] == 1){ //czy dana krawedz zawiera warunek brzegowy
                for (int m = 0; m < factors.edge_points; ++m) {
                    for (int k = 0; k < factors.nodes; ++k) {
                        for (int l = 0; l < factors.nodes; ++l) {
                            matrix[i][k][l] += factors.shape_func_boundary[j][m][k] * factors.shape_func_boundary[j][m][l] * factors.w_array1D[m] * globalData.alfa * dnDxDnDy_array[i].detJ1D[j]; //Hbc
                        }
                    }
                }
            }
        }
    }
    return matrix;
}
```

Funkcja P() służąca do wyznaczania wektora P zgodnie z wzorami:



```
double** P (struct GlobalData globalData, struct shapeFactors factors, struct DnDxDnDy* dnDxDnDy){ // obliczanie wektora P
    double**P = new double * [globalData.elementsNumber];
    for (int i = 0; i < globalData.elementsNumber; ++i) {
        P[i] = new double [4];
        for (int j = 0; j < 4; ++j) {
            P[i][j] = 0;
        }
        for (int j = 0; j < 4; ++j) {
            if(dnDxDnDy[i].boundaryEdges[j] == 1){
                for (int k = 0; k < factors.edge_points; ++k) {
                    for (int l = 0; l < factors.nodes; ++l) {
                        P[i][l] += factors.shape_func_boundary[j][k][l] * factors.w_array1D[k] * globalData.alfa * globalData.tot * dnDxDnDy[i].detJ1D[j];
                    }
                }
            }
        }
    }
    return P;
}
```

Funkcja gauss() odpowiada za metodę eliminacji Gaussa wykorzystując wykonane obliczenia z metod sumMatrix() odpowiadającej za pierwszą część równania układu niestacjonarnego zawierającą zaagregowane macierz FullH i macierz C, krok czasowy, oraz obliczoną temperaturę oraz PandC() odpowiadającej za drugą część owego równania zawierającą temperaturę z poprzedniego kroku, zaagregowaną macierz C podzieloną przez krok czasowy, zaagregowany wektor P.

```
double* gauss (double** Hinitial, double* Pinitial, int N) {
    double** H = new double* [N];
    double* P = new double[N];
    for (int i = 0; i < N; ++i) {
        H[i] = new double [N];
        for (int j = 0; j <  N; ++j) {
            H[i][j] = Hinitial[i][j]; //przypisanie wyniku z HandC sumMatrix
        }
        P[i] = Pinitial[i]; //PandC
    }
    for (int i = 0; i < N; ++i) {
        double c = H[i][i]; //wspolczynnik w algorytmie gaussa
        for (int j = i; j < N; ++j) {
            H[i][j] = H[i][j] / c; //uzyskanie 1 na diagonali
        }
        P[i]=P[i] / c; //dzielenie c dla prawej strony rownania

        for (int j = i+1; j < N; ++j) {
            c = H[j][i];
            for (int k = i; k < N; ++k) {
                H[j][k] -= c * H[i][k];
            }
            P[j] -= P[i] * c;
        }
    }
    for (int i = N-1; i >= 0 ; --i) {
        for (int j = i-1; j >= 0; --j){
            double c = H[j][i];
            for (int k = i; k >= j; --k) {
                H[j][k] -= c * H[i][k];
            }
            P[j] -= P[i] * c;
        }
    }
    return P;
}
```

Wykonanie obliczeń układu równań w celu wyznaczenia $\{t_1\}$ w danym kroku czasowym w funkcji symulacjaCzasowa() przy użyciu metody eliminacji Gaussa.

```
double** symulacjaCzasowa (double** H, double** C, double* P, struct GlobalData globalData){
    int t = 0;
    int N = globalData.simulationTime / globalData.simulationStepTime + 1;

    double** temp = new double* [N];
    for (int i = 0; i < N; ++i) {
        temp[i] = new double [globalData.nodesNumber];
        for (int j = 0; j < globalData.nodesNumber; ++j) {
            temp[i][j] = 0;
        }
    }
    double** HandC = sumMatrix(H, C, globalData);  // pierwsza czesc rownania
    for (int i = 0; i < globalData.nodesNumber; ++i) {
        temp[0][i] = globalData.initialTemp; //t0
    }

    for (int i = 0; i < N-1; ++i) {
        double* vec = PandC(C, P,  temp: temp[i], globalData); //druga czesc rownania

        temp[i+1] =  gauss( Hinitial: HandC,  Pinitial: vec,  N: globalData.nodesNumber); //t1
    }
    return temp;
}
```

Następnie w funkcji main() wczytano siatkę z warunkami brzegowymi, odpowiednimi parametrami do symulacji, oraz wywołano wcześniej napisane funkcje dla konkretnych schematów całkowania.

Aby uzyskać dokładne wyniki symulacji z danego kodu, warto uruchamiać go na systemie operacyjnym Linux, gdzie dodatkowo obliczenia zostają wykonywane szybciej.

Wyniki oprogramowania:

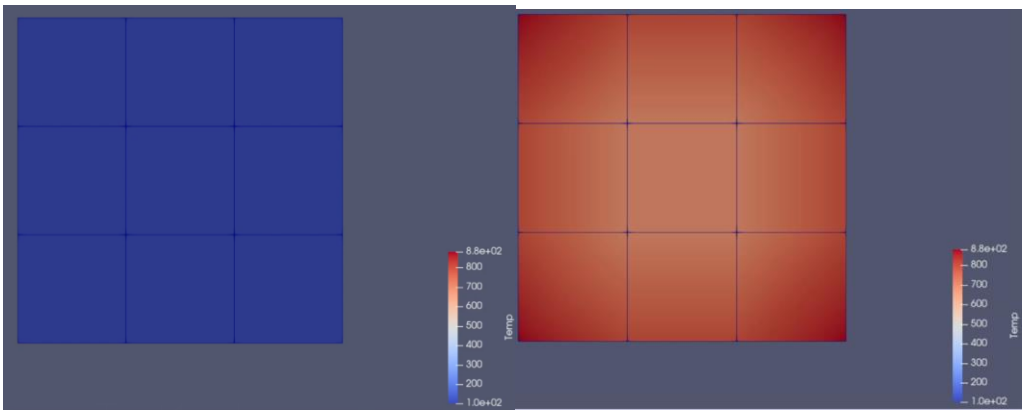
Wyniki symulacji dla podanych testów przedstawione w sposób: Krok czasowy, najniższa uzyskana temperatura spośród węzłów, najwyższa temperatura uzyskana spośród węzłów.

Wyniki przedstawiono również w formie wizualnej w programie ParaView gdzie wczytano poszczególne siatki wraz z uzyskanymi wynikami.

Test1_4_4:

```
Symulacja:
T = 0
temp min= 100 temp max= 100
T = 50
temp min= 110.038 temp max= 365.815
T = 100
temp min= 168.837 temp max= 502.592
T = 150
temp min= 242.801 temp max= 587.373
T = 200
temp min= 318.615 temp max= 649.387
T = 250
temp min= 391.256 temp max= 700.068
T = 300
temp min= 459.037 temp max= 744.063
T = 350
temp min= 521.586 temp max= 783.383
T = 400
temp min= 579.034 temp max= 818.992
T = 450
temp min= 631.689 temp max= 851.431
T = 500
temp min= 679.908 temp max= 881.058
```

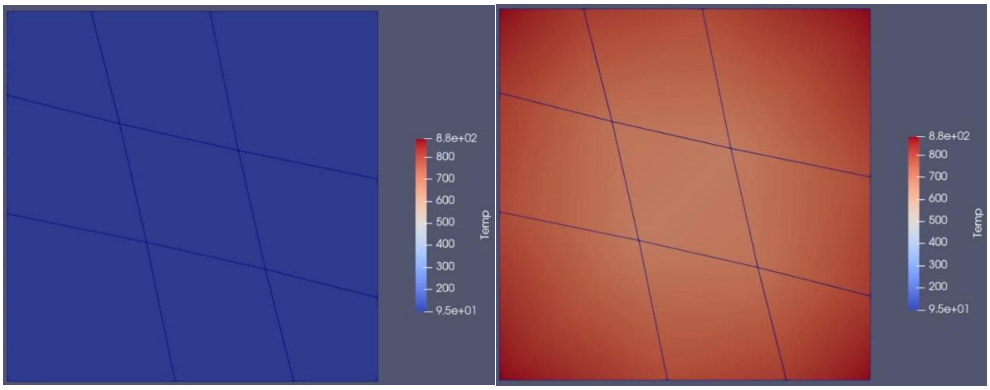
ParaView (początek i koniec symulacji):



Test2_4_4_MixGrid:

```
Symulacja:
T = 0
temp min= 100 temp max= 100
T = 50
temp min= 95.1591 temp max= 374.668
T = 100
temp min= 147.656 temp max= 505.954
T = 150
temp min= 220.178 temp max= 586.989
T = 200
temp min= 296.751 temp max= 647.28
T = 250
temp min= 370.983 temp max= 697.33
T = 300
temp min= 440.574 temp max= 741.216
T = 350
temp min= 504.904 temp max= 781.241
T = 400
temp min= 564.014 temp max= 817.421
T = 450
temp min= 618.185 temp max= 850.264
T = 500
temp min= 667.776 temp max= 880.192
```

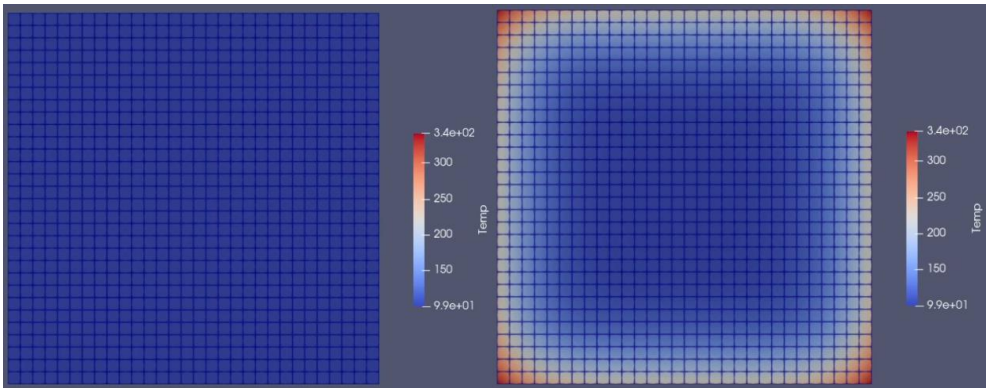
ParaView (początek i koniec symulacji):



Test3_31_31_kwadrat:

```
Symulacja:
T = 0
temp min= 100 temp max= 100
T = 1
temp min= 100 temp max= 149.557
T = 2
temp min= 100 temp max= 177.445
T = 3
temp min= 100 temp max= 197.267
T = 4
temp min= 100 temp max= 213.153
T = 5
temp min= 100 temp max= 226.683
T = 6
temp min= 100 temp max= 238.607
T = 7
temp min= 100 temp max= 249.347
T = 8
temp min= 100 temp max= 259.165
T = 9
temp min= 100 temp max= 268.241
T = 10
temp min= 100 temp max= 276.701
T = 11
temp min= 100.001 temp max= 284.641
T = 12
temp min= 100.002 temp max= 292.134
T = 13
temp min= 100.003 temp max= 299.237
T = 14
temp min= 100.005 temp max= 305.997
T = 15
temp min= 100.009 temp max= 312.451
T = 16
temp min= 100.014 temp max= 318.631
T = 17
temp min= 100.021 temp max= 324.564
T = 18
temp min= 100.032 temp max= 330.271
T = 19
temp min= 100.046 temp max= 335.772
T = 20
temp min= 100.064 temp max= 341.085
```

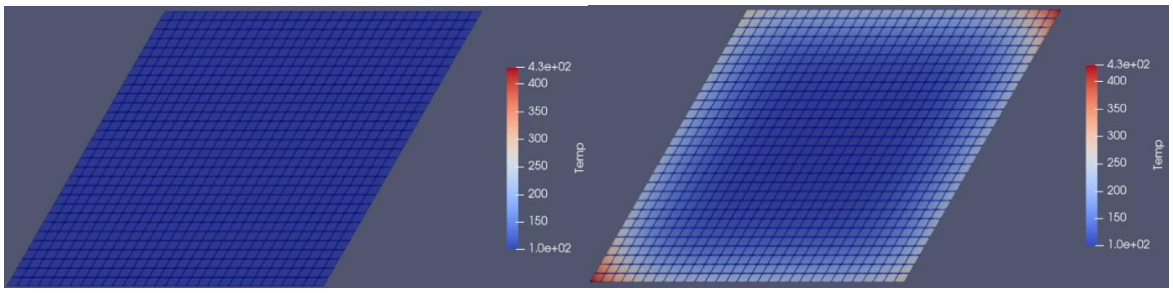
ParaView (początek i koniec symulacji):



Test4_31_31_trapez:

```
Symulacja:
T = 0
temp min= 100 temp max= 100
T = 1
temp min= 100 temp max= 166.936
T = 2
temp min= 100 temp max= 207.233
T = 3
temp min= 100 temp max= 236.287
T = 4
temp min= 100 temp max= 259.465
T = 5
temp min= 100 temp max= 279.031
T = 6
temp min= 100 temp max= 296.121
T = 7
temp min= 100.001 temp max= 311.385
T = 8
temp min= 100.001 temp max= 325.235
T = 9
temp min= 100.003 temp max= 337.951
T = 10
temp min= 100.005 temp max= 349.731
T = 11
temp min= 100.01 temp max= 360.723
T = 12
temp min= 100.018 temp max= 371.04
T = 13
temp min= 100.03 temp max= 380.771
T = 14
temp min= 100.047 temp max= 389.987
T = 15
temp min= 100.072 temp max= 398.747
T = 16
temp min= 100.105 temp max= 407.099
T = 17
temp min= 100.149 temp max= 415.083
T = 18
temp min= 100.205 temp max= 422.734
T = 19
temp min= 100.276 temp max= 430.081
T = 20
temp min= 100.364 temp max= 437.15
```

ParaView (początek i koniec symulacji):



Wnioski:

Dzięki zastosowaniu metody elementów skończonych, udało się skutecznie rozwiązać problem wyznaczenia temperatury w każdym węźle i symulacji procesu cieplnego badanych obiektów. Metoda elementów skończonych umożliwia nam podział dużego zadania na mniejsze, znacznie łatwiejsze do policzenia oraz zwracające dokładniejsze wyniki zadania, co niestety wpływa negatywnie na czas przeprowadzania obliczeń. Wyniki temperatur obliczone za pomocą oprogramowania, są zgodne z wynikami podanymi przez prowadzącego, przez co można stwierdzić, że symulacja wykonuje się prawidłowo W celu poprawienia działania programu, trzeba usunąć zbędne i powtarzalne operacje. Ze względu na złożoność obliczeniową metody eliminacji Gaussa i złożone obliczenia dla każdego elementu, należy się liczyć z długim czasem wykonania dla siatek z testów 3 i 4.