SPRAWOZDANIE

METODA ELEMENTÓW SKOŃCZONYCH

PATRYCJA JABŁOŃSKA, GR. LAB. 2, INFORMATYKA TECHNICZNA

Spis treści

Wstęp teoretyczny

Metoda elementów skończonych jest techniką przybliżania równań różniczkowych cząstkowych, które zazwyczaj służą jako matematyczne modele stanów systemów fizycznych. Te modele są reprezentowane przez funkcje zależne od czasu i przestrzeni.

MES jest powszechnie stosowana do rozwiązywania problemów inżynieryjnych i matematycznych. Kluczowym założeniem jest możliwość przekształcenia dowolnej wartości ciągłej (na przykład temperatury) na model dyskretny. Ten model opiera się na ograniczonej liczbie punktów, zwanych węzłami, które tworzą skończoną liczbę elementów skończonych.

Podczas naszych zajęć skupialiśmy się na problemie transferu ciepła. Model fizycznego ciała był reprezentowany za pomocą węzłów i elementów.

Opis modelu z warunkami brzegowymi

Do testowania używamy 4 siatek MES:

1. Test 1 siatka kwadratowa 4x4 węzły – Test1_4_4
2. Test 2 siatka kwadratowa 4x4 węzły - Test2_4_4_MixGrid
3. Test 3 siatka kwadratowa 31x31 – Test3_31_31_kwadrat

4. Test 4 siatka kwadratowa 31x31 – Test4_31_31_trapez

Każdy plik zawiera wartości początkowe składające się z:

- Czas Symulacji
- Krok Symulacji
- Przewodność cieplna
- Alfa
- Temperatura otoczenia
- Temperatura Początkowa
- Gęstość
- Ciepło Właściwe

Podane są także:

- Liczba Węzłów
- Liczba Elementów

Następnie:

- Współrzędne Węzłów
- Połączenia Elementów
- Warunki Brzegowe

Opis MES

Do opisu przepływu ciepła w programie wykorzystuje się równanie Fouriera. Głównym celem było przekształcenie modelu ciągłego na model dyskretny, który opiera się na ograniczonej liczbie węzłów, które są reprezentowane za pomocą skończonej liczby elementów skończonych.

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

które może być zapisane też w postaci:

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

Rozwiązanie danego równania polega na poszukiwaniu minimum funkcjonału, dla którego to równanie jest równaniem Eulera. Zgodnie z kalkulacją wariacyjną, przy założeniu, że materiał jest izotropowy, funkcjonał przyjmuje następującą formę:

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

Aby funkcjonał spełniał warunki brzegowe, rozszerzamy go o całkę i ostateczny wzór ma postać:

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

Minimalizacja funkcjonału polega na obliczeniu pochodnych cząstkowych względem wartości temperatur w węzłach.

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

Macierz H to macierz, która charakteryzuje jeden element skończony. Informuje, w jaki sposób następuje transfer ciepła w jednym elemencie skończonym.

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

Macierz Hbc opisuje zachowanie materiału na brzegu (uwzględnia warunek brzegowy), w tym przypadku obliczana jest wartość całki po powierzchni. Warunek brzegowy dzielimy na dwie części. W macierzy Hbc uwzględniamy temperaturę, która się zmienia w trakcie nagrzewania, dlatego później sumujemy ją z macierzą H podczas agregacji.

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

Macierz C jest miarą ilości ciepła, które może być przechowywane przez pojedynczy element skończony. Jest ona funkcją pojemności cieplnej i gęstości.

$$[C] = \int_V \rho c_p (\{N\}\{N\}^T) \, dV$$

Wektor P jest związany z drugim warunkiem brzegowym i pokazuje, jak temperatura otoczenia wpływa na ściany materiału. Temperatura otoczenia jest stała. Wektor P jest obliczany numerycznie. Oblicza się całkę po powierzchni ściany elementu, na której występuje warunek brzegowy. Kryterium spełnienia warunku brzegowego jest takie samo jak dla macierzy Hbc.

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

Charakterystyka kodu

Następujące kroki implementacji rozwiązania, które obsługuje 2, 3 i 4 punktowe schematy całkowania:

1. Wczytanie z pliku wartości początkowych, węzłów oraz elementów.

```
if (word == "SimulationTime") iss >> sim_time;
else if (word == "SimulationStepTime") iss >> sim_step_time;
else if (word == "Conductivity") iss >> con;
else if (word == "Alfa") iss >> alfa;
else if (word == "Tot") iss >> tot;
else if (word == "InitialTemp") iss >> in_temp;
else if (word == "Density") iss >> den;
else if (word == "SpecificHeat") iss >> spec_heat;
else if (word == "Nodes") {
    string pom;
    iss >> pom;
    if (pom == "number") iss >> nn;
}
else if (word == "Elements") {
    string pom;
    iss >> pom;
    if (pom == "number") iss >> en;
```

```
while (getline(file, line)) {
    if (line == "*Node") {
        nodes = new Node[globalData.nn + 1];
        for (int i = 1; i <= globalData.nn; i++) {
            getline(file, line);
            istringstream iss(line);
            int n;
            double x, y;
            char c;
            iss >> n >> c >> x >> c >> y;
            nodes[i].x = x;
            nodes[i].y = y;
            nodes[i].BC = 0;
        }
    }
    else if (line.find("*Element") != string::npos) {
        elements = new Element[globalData.en];
        for (int i = 0; i < globalData.en; i++) {
            getline(file, line);
            istringstream iss(line);
            int n;
            int a, b, c, d;
            char ch;
            iss >> n >> ch >> a >> ch >> b >> ch >> c >> ch >> d;
            elements[i].nodes[0] = a;
            elements[i].nodes[1] = b;
            elements[i].nodes[2] = c;
            elements[i].nodes[3] = d;
        }
    }
```

Wynik programu dla przykładowego pliku:

```
SimulationTime: 500
SimulationStepTime: 50
Conductivity: 25
Alfa: 300
Tot: 1200
InitialTemp: 100
Density: 7800
SpecificHeat: 700
Nodes number: 16
Elements number: 9
Nodes:
1. = [ 0.1, 0.005 ] BC = 1
2. = [ 0.0546918, 0.005 ] BC = 1
3. = [ 0.0226541, 0.005 ] BC = 1
4. = [ 0, 0.005 ] BC = 1
5. = [ 0.1, -0.0403082 ] BC = 1
6. = [ 0.0623899, -0.0326101 ] BC = 0
7. = [ 0.0303522, -0.0253522 ] BC = 0
8. = [ 0, -0.0176541 ] BC = 1
9. = [ 0.1, -0.0723459 ] BC = 1
10. = [ 0.06964478, -0.06464478 ] BC = 0
11. = [ 0.0376101, -0.0573899 ] BC = 0
12. = [ 0, -0.0496918 ] BC = 1
13. = [ 0.1, -0.095 ] BC = 1
14. = [ 0.0773459, -0.095 ] BC = 1
15. = [ 0.0453082, -0.095 ] BC = 1
16. = [ 0, -0.095 ] BC = 1
Elements:
1. = [ 1, 2, 6, 5 ]
2. = [ 2, 3, 7, 6 ]
3. = [ 3, 4, 8, 7 ]
4. = [ 5, 6, 10, 9 ]
5. = [ 6, 7, 11, 10 ]
6. = [ 7, 8, 12, 11 ]
7. = [ 9, 10, 14, 13 ]
8. = [ 10, 11, 15, 14 ]
9. = [ 11, 12, 16, 15 ]
```

2. Obliczenie wartości dN_dEta oraz dN_dKsi.

```
if (index < points_number) {
    dN_dEta[index][0] = -(1.0 / 4.0) * (1 - gauss.get_point(i));
    dN_dEta[index][1] = (1.0 / 4.0) * (1 - gauss.get_point(i));
    dN_dEta[index][2] = (1.0 / 4.0) * (1 + gauss.get_point(i));
    dN_dEta[index][3] = -(1.0 / 4.0) * (1 + gauss.get_point(i));
}
if (index < points_number) {
    dN_dKsi[index][0] = -(1.0 / 4.0) * (1 - gauss.get_point(j));
    dN_dKsi[index][1] = -(1.0 / 4.0) * (1 + gauss.get_point(j));
    dN_dKsi[index][2] = (1.0 / 4.0) * (1 + gauss.get_point(j));
    dN_dKsi[index][3] = (1.0 / 4.0) * (1 - gauss.get_point(j));
}
```

Wynik programu dla przykładowego pliku:

```
* * * dN_dKsi * * *
        N1              N2              N3              N4
1 |     -0.443649 |     -0.0563508 |    0.0563508 |     0.443649
2 |     -0.25 |         -0.25 |         0.25 |  0.25
3 |     -0.0563508 |    -0.443649 |     0.443649 |     0.0563508
4 |     -0.443649 |     -0.0563508 |    0.0563508 |     0.443649
5 |     -0.25 |         -0.25 |         0.25 |  0.25
6 |     -0.0563508 |    -0.443649 |     0.443649 |     0.0563508
7 |     -0.443649 |     -0.0563508 |    0.0563508 |     0.443649
8 |     -0.25 |         -0.25 |         0.25 |  0.25
9 |     -0.0563508 |    -0.443649 |     0.443649 |     0.0563508

* * * dN_dEta * * *
        N1              N2              N3              N4
1 |     -0.443649 |     0.443649 |      0.0563508 |     -0.0563508
2 |     -0.443649 |     0.443649 |      0.0563508 |     -0.0563508
3 |     -0.443649 |     0.443649 |      0.0563508 |     -0.0563508
4 |     -0.25 |         0.25 |  0.25 |  -0.25
5 |     -0.25 |         0.25 |  0.25 |  -0.25
6 |     -0.25 |         0.25 |  0.25 |  -0.25
7 |     -0.0563508 |    0.0563508 |     0.443649 |     -0.443649
8 |     -0.0563508 |    0.0563508 |     0.443649 |     -0.443649
9 |     -0.0563508 |    0.0563508 |     0.443649 |     -0.443649
```

3. Obliczanie jakobianu.

```
double* dN_dKsi = universalElement.dN_dKsi[index];
double* dN_dEta = universalElement.dN_dEta[index];

double x0 = nodes[elements[indexi].nodes[0]].get_x();
double x1 = nodes[elements[indexi].nodes[1]].get_x();
double x2 = nodes[elements[indexi].nodes[2]].get_x();
double x3 = nodes[elements[indexi].nodes[3]].get_x();

double y0 = nodes[elements[indexi].nodes[0]].get_y();
double y1 = nodes[elements[indexi].nodes[1]].get_y();
double y2 = nodes[elements[indexi].nodes[2]].get_y();
double y3 = nodes[elements[indexi].nodes[3]].get_y();

jakobian[0][0] = dN_dKsi[0] * y0 + dN_dKsi[1] * y1 + dN_dKsi[2] * y2 + dN_dKsi[3] * y3;
jakobian[0][1] = dN_dEta[0] * y0 + dN_dEta[1] * y1 + dN_dEta[2] * y2 + dN_dEta[3] * y3;
jakobian[1][0] = dN_dKsi[0] * x0 + dN_dKsi[1] * x1 + dN_dKsi[2] * x2 + dN_dKsi[3] * x3;
jakobian[1][1] = dN_dEta[0] * x0 + dN_dEta[1] * x1 + dN_dEta[2] * x2 + dN_dEta[3] * x3;

det = (jakobian[1][1] * jakobian[0][0]) - (jakobian[1][0] * jakobian[0][1]);

reverse_jakobian[0][0] = jakobian[0][0] / det;
reverse_jakobian[0][1] = -jakobian[0][1] / det;
reverse_jakobian[1][0] = -jakobian[1][0] / det;
reverse_jakobian[1][1] = jakobian[1][1] / det;
```

Jakobian w przestrzeni 2D to proporcja pól, nie odcinków. Jest to macierz 2x2, ponieważ obliczamy go w dwuwymiarowej przestrzeni. Na przekątnej mamy współrzędne odpowiadające za rozszerzanie i kurczenie w poziomie i pionie, a druga przekątna to skosy elementów. Liczba jakobianów dla macierzy H i C zależy od punktów całkowania, ponieważ w każdym punkcie całkowania obliczamy jakobian.

Wynik programu dla przykładowego pliku:

```
***Dla punku calkowania:***:1
my_jakobian:

-0.0166667              0
0               -0.0166667

det:0.000277778
```

4. Obliczanie macierzy H

Obliczamy dN_dx oraz dN_dy potrzebne do wyliczenia macierzy H.

```
dN_dx[j][k] = jakob.get_reverse_jakobian(0, 0) * dN_dEta[k]
    + jakob.get_reverse_jakobian(0, 1) * dN_dKsi[k];
dN_dy[j][k] = jakob.get_reverse_jakobian(1, 0) * dN_dEta[k]
    + jakob.get_reverse_jakobian(1, 1) * dN_dKsi[k];
```

W macierzy H obliczamy zmienną temperaturę. Proces ten obejmuje całkowanie po powierzchni (pomnożone przez wyznacznik jakobianu 2x2). Punkty całkowania w macierzy H są wykorzystywane w pochodnych funkcji kształtu względem x i y.

```
for (int punkty_c = 0; punkty_c < points_number; punkty_c++) {

    double resultX[4][4];
    for (int a = 0; a < 4; ++a) {
        for (int j = 0; j < 4; ++j) {
            resultX[a][j] = dN_dx[punkty_c][a] * dN_dx[punkty_c][j];
        }
    }
    double resultY[4][4];
    for (int a = 0; a < 4; ++a) {
        for (int j = 0; j < 4; ++j) {
            resultY[a][j] = dN_dy[punkty_c][a] * dN_dy[punkty_c][j];
        }
    }
    double resultSum[4][4];
    for (int a = 0; a < 4; a++) {
        for (int j = 0; j < 4; j++) {
            resultSum[a][j] = global.con * (resultX[a][j] + resultY[a][j]) * pom_det[punkty_c];
        }
    }
    for (int a = 0; a < 4; ++a) {
        for (int j = 0; j < 4; ++j) {
            elements[i].set_H(a, j, (resultSum[a][j] * (gauss.get_weight(punkty_c / gauss.get_number_pc()))
                * gauss.get_weight(punkty_c % gauss.get_number_pc())));
        }
    }
```

Wynik programu dla przykładowego pliku:

```
****Glowna Macierz H:****
23.3769 -5.65921        -13.2722        -4.44553
-5.65921        12.6566 -2.49555        -4.50181
-13.2722        -2.49555        20.9584 -5.19066
-4.44553        -4.50181        -5.19066        14.138
```

5. Obliczanie macierzy C

W macierzy C wartości są przemnażane przez parametry gęstości i ciepła właściwego. Aby uzyskać te wartości, stosujemy funkcje kształtu pomnożone przez odpowiednie wagi. Przejście między układem lokalnym a globalnym realizujemy poprzez obliczenia wyznacznika macierzy Jacobiego.

```
for (int k = 0; k < 4; k++) {
    for (int l = 0; l < 4; l++) {
        elements[i].set_C(k, l, (global.den * global.spec_heat * (element.shapes_function_values[punkty_c][k]) * (element.shapes_function_values[punkty_
            * pom_det[punkty_c] * (gauss.get_weight(punkty_c / gauss.get_number_pc()) * gauss.get_weight(punkty_c % gauss.get_number_pc()))));
    }
}
```

Wynik programu dla przykładowego pliku:

```
**** Macierz C lokalnie:****
674.074 337.037 168.519 337.037
337.037 674.074 337.037 168.519
168.519 337.037 674.074 337.037
337.037 168.519 337.037 674.074
```

6. Obliczanie macierzy Hbc i wektora P

W kontekście macierzy Hbc i wektora P, ilość jakobianów odpowiada liczbie powierzchni brzegowych. Jakobian reprezentuje stosunek długości odcinka w układzie lokalnym do długości odcinka w układzie globalnym, przy czym krok w układzie lokalnym wynosi 2. Dlatego obliczoną długość odcinka dzielimy przez 2.

```
double calculate_detJ(Node node1, Node node2) {
    return sqrt(pow(node1.get_x() - node2.get_x(), 2) + pow(node1.get_y() - node2.get_y(), 2));
}
```

Macierz Hbc w połączeniu z wektorem P tworzą warunek brzegowy, który opisuje zachowanie na krawędziach. Punkty całkowania w macierzy Hbc są używane do obliczania macierzy kształtów N. Najpierw formułujemy funkcje kształtu, następnie obliczamy jakobian. Następnie mnożymy przez detJ oraz współczynnik przekazywania ciepła. Musimy również pamiętać o pomnożeniu przez odpowiednie wagi. Wektor P jest obliczany poprzez mnożenie temperatury otoczenia, współczynnika alfa, wartości funkcji w punktach całkowania i ich wag.

```
for (int j = 0; j < 4; j++) {
    for (int i = 0; i < gauss.get_number_pc(); i++) {
        surface[j].shapes_function_values[i][0] = (1.0 / 4.0) * (1.0 - surface[j].ksi[i]) * (1.0 - surface[j].eta[i]);
        surface[j].shapes_function_values[i][1] = (1.0 / 4.0) * (1.0 + surface[j].ksi[i]) * (1.0 - surface[j].eta[i]);
        surface[j].shapes_function_values[i][2] = (1.0 / 4.0) * (1.0 + surface[j].ksi[i]) * (1.0 + surface[j].eta[i]);
        surface[j].shapes_function_values[i][3] = (1.0 / 4.0) * (1.0 - surface[j].ksi[i]) * (1.0 + surface[j].eta[i]);
    }
}
```

```
for (int n = 0; n < 4; n++) {
    int node1 = elements[i].get_nodes(n);
    int node2 = elements[i].get_nodes((n + 1) % 4);
    if (nodes[node1].get_bc() && nodes[node2].get_bc()) {
        double det_J = jakob.calculate_detJ(nodes[node1], nodes[node2]) / 2.0;
        for (int j = 0; j < element.gauss.get_number_pc(); j++) {
            for (int k = 0; k < 4; k++) {
                for (int l = 0; l < 4; l++) {
                    elements[i].set_H_bc(k, l, (global.alfa * element.surface[n].weights[j] * (element.surface[n].shapes_function_values[j][k])
                        * (element.surface[n].shapes_function_values[j][l]) * det_J));
                }
                elements[i].set_P(k, (global.alfa * det_J * global.tot * element.surface[n].weights[j] * (element.surface[n].shapes_function_values[j][k])));
            }
        }
```

Wynik programu dla przykładowego pliku:

```
**** Macierz H_bc:****
6.66667 1.66667 0       1.66667
1.66667 3.33333 0       0
0       0       0       0
1.66667 0       0       3.33333
```

7. Agregacja macierzy H

Agregacja to proces przechodzenia z układu lokalnego do globalnego. Polega na zsumowaniu lokalnych macierzy H i Hbc w celu utworzenia globalnej macierzy H.

```
for (int n = 0; n < 4; n++) {
    for (int m = 0; m < 4; m++) {
        elements[i].set_H_sum(n, m, (elements[i].get_H(n, m) + elements[i].get_H_bc(n, m)));
    }
}
```

```
for (int i = 0; i < global.en; i++)
{
    for (int n = 0; n < 4; n++) {
        for (int m = 0; m < 4; m++) {
            H_global[elements[i].get_nodes(n) - 1][elements[i].get_nodes(m) - 1] += elements[i].get_H_sum(n, m);
        }
    }
}
```

Wynik dla przykładowego pliku:

```
**** Macierz H_global:****
26.824 -1.13431 0 0 -1.13431 -10.963 0 0 0 0 0 0 0 0 0 0
-1.13431 43.3438 -3.58878 0 -6.10152 -7.64516 -13.2722 0 0 0 0 0 0 0 0 0
0 -3.58878 40.7351 -4.35854 0 -4.50181 -7.71486 -12.3674 0 0 0 0 0 0 0 0
0 0 -4.35854 20.7445 0 0 -5.23115 -4.35854 0 0 0 0 0 0 0 0
-1.13431 -6.10152 0 0 43.3438 -7.64516 0 0 -3.58878 -13.2722 0 0 0 0 0 0
-10.963 -7.64516 -4.50181 0 -7.64516 71.0151 -10.2767 0 -4.50181 -10.2767 -15.2049 0 0 0 0 0
0 -13.2722 -7.71486 -5.23115 0 -10.2767 71.0237 -7.71486 0 -3.26505 -10.2767 -13.2722 0 0 0 0
0 0 -12.3674 -4.35854 0 0 -7.71486 40.7351 0 0 -4.50181 -3.58878 0 0 0 0
0 0 0 0 -3.58878 -4.50181 0 0 40.7351 -7.71486 0 0 -4.35854 -12.3674 0 0
0 0 0 0 -13.2722 -10.2767 -3.26505 0 -7.71486 71.0237 -10.2767 0 -5.23115 -7.71486 -13.2722 0
0 0 0 0 0 -15.2049 -10.2767 -4.50181 0 -10.2767 71.0151 -7.64516 0 -4.50181 -7.64516 -10.963
0 0 0 0 0 0 -13.2722 -3.58878 0 0 -7.64516 43.3438 0 0 -6.10152 -1.13431
0 0 0 0 0 0 0 0 0 -4.35854 -5.23115 0 0 20.7445 -4.35854 0 0
0 0 0 0 0 0 0 0 0 -12.3674 -7.71486 -4.50181 0 -4.35854 40.7351 -3.58878 0
0 0 0 0 0 0 0 0 0 0 -13.2722 -7.64516 -6.10152 0 -3.58878 43.3438 -1.13431
0 0 0 0 0 0 0 0 0 0 0 -10.963 -1.13431 0 0 -1.13431 26.824
```

8. Utworzenie wektora P globalnego.

```
for (int i = 0; i < global.en; i++) {
    for (int j = 0; j < 4; j++) {
        P_global[elements[i].get_nodes(j) - 1] += elements[i].get_P(j);
    }
}
```

Wynik dla przykładowego pliku:

```
**** Macierz P_global:****
16310.9, 13922.3, 9844.53, 8155.47, 13922.3, 0, 0, 9844.53, 9844.53, 0, 0, 13922.3, 8155.47, 9844.53, 13922.3, 16310.9
```

9. Agregacja macierzy C

```
for (int n = 0; n < 4; n++) {
    for (int m = 0; m < 4; m++) {
        C_global[elements[i].get_nodes(n) - 1][elements[i].get_nodes(m) - 1] += elements[i].get_C(n, m);
    }
}
```

10. Utworzenie macierzy [HC_koniec]: H + C/dT

```
for (int a = 0; a < pom; a++) {
    for (int b = 0; b < pom; b++) {
        C_koniec[a][b] = C_global[a][b] / global.sim_step_time;
    }
}

for (int a = 0; a < pom; a++) {
    for (int b = 0; b < pom; b++) {
        HC_koniec[a][b] = C_koniec[a][b] + H_global[a][b];
    }
}
```

Wynik programu dla przykładowego pliku:

```
**** Macierz [H] + [C]_dT:****
49.6157 9.73255 0 0 9.73255 -5.79402 0 0 0 0 0 0 0 0 0
9.73255 77.7647 2.93113 0 -0.932583 8.95154 -10.0546 0 0 0 0 0 0 0 0
0 2.93113 61.4125 -0.45159 0 -1.28423 2.80359 -10.2816 0 0 0 0 0 0 0
0 0 -0.45159 28.0294 0 0 -3.14542 -0.451591 0 0 0 0 0 0 0
9.73255 -0.932583 0 0 77.7647 8.95155 0 0 2.93113 -10.0546 0 0 0 0 0
-5.79402 8.95154 -1.28423 0 8.95155 128.202 1.9811 0 -1.28423 1.98109 -12.2512 0 0 0 0 0
0 -10.0546 2.80359 -3.14542 0 1.9811 116.23 2.8036 0 -0.311373 1.9811 -10.0546 0 0 0 0
0 0 -10.2816 -0.451591 0 0 2.8036 61.4125 0 0 -1.28423 2.93113 0 0 0 0
0 0 0 0 2.93113 -1.28423 0 0 61.4125 2.8036 0 0 -0.451589 -10.2816 0 0
0 0 0 0 -10.0546 1.98109 -0.311373 0 2.8036 116.23 1.9811 0 -3.14542 2.8036 -10.0546 0
0 0 0 0 0 -12.2512 1.9811 -1.28423 0 1.9811 128.202 8.95154 0 -1.28423 8.95154 -5.79402
0 0 0 0 0 0 -10.0546 2.93113 0 0 8.95154 77.7647 0 0 -0.932583 9.73255
0 0 0 0 0 0 0 0 0 -0.451589 -3.14542 0 0 28.0294 -0.451589 0 0
0 0 0 0 0 0 0 0 0 -10.2816 2.8036 -1.28423 0 -0.451589 61.4125 2.93113 0
0 0 0 0 0 0 0 0 0 0 -10.0546 8.95154 -0.932583 0 2.93113 77.7647 9.73255
0 0 0 0 0 0 0 0 0 0 0 -5.79402 9.73255 0 0 9.73255 49.6157
```

11. Utworzenie wektora {P_koniec}: P+(C/dT)*t0, gdzie w kolejnych iteracjach pętli nowo obliczone temperatury zostają przypisane jako wartości początkowe.

```
for (int a = 0; a < pom; a++) {
    for (int b = 0; b < pom; b++) {
        if (ilosc == 1)
            C_koniec[a][b] *= global.in_temp;
        else
            C_koniec[a][b] *= t1[b];

        C_dt[a] += C_koniec[a][b];
    }
}

for (int a = 0; a < pom; a++)
{
    P_koniec[a] = P_global[a] + C_dt[a];
}
```

12. Zaimplementowanie funkcji z eliminacją Gaussa do obliczenia nowych temperatur w węzłach

```
bool eliminategauss(int n, double** AB, double* X)
{
    double m, s;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (fabs(AB[i][i]) < eps)
            {
                return false;
            }
            m = -AB[j][i] / AB[i][i];
            AB[j][i] = 0;
            for (int k = i + 1; k <= n; k++)
            {
                AB[j][k] += m * AB[i][k];
            }
        }
    }
    for (int i = n - 1; i >= 0; i--)
    {
        s = AB[i][n];

        for (int j = n - 1; j >= i + 1; j--)
        {
            s -= AB[i][j] * X[j];
        }
        if (fabs(AB[i][i]) < eps)
        {
            return false;
        }

        X[i] = s / AB[i][i];
    }

    return true;
}
```

13. Rozwiązanie układu równań - obliczamy wyniki w postaci wektora temperatur {t}. Proces ten obejmuje zastosowanie eliminacji Gaussa do obliczenia nowych temperatur w węzłach, a następnie przypisanie nowo obliczonych temperatur jako wartości początkowe w kolejnych iteracjach pętli. W rezultacie uzyskujemy aktualny stan temperatury w układzie. Następnie wypisujemy na ekranie maksymalną oraz minimalną temperaturę w danym kroku czasowym, co pozwala monitorować zmiany temperatury w trakcie symulacji.

```cpp
for (int x = 0; x < pom; x++) {
    for (int y = 0; y < pom; y++) {
        AB[x][y] = HC_koniec[x][y];
    }
}
for (int x = 0; x < pom; x++) {
    AB[x][pom] = P_koniec[x];
}

X = new double[pom];
eliminategauss(pom, AB, X);

for (int i = 0; i < pom; i++)
{
    t1[i] = 0.0;
}
for (int i = 0; i < pom; i++)
{
    t1[i] = X[i];
}

min = X[0];
max = X[0];
for (int i = 1; i < pom; i++)
{
    if (X[i] > max)
    {
        max = X[i];
    }
    else if (X[i] < min)
    {
        min = X[i];
    }
}
cout << endl;
cout << "Time = " << global.sim_step_time * ilosc << ", Min_t = " << min << ", Max_t = " << max << endl;
```

Porównanie wyników oprogramowania z testami zamieszczonymi na UPEL

- Test1_4_4

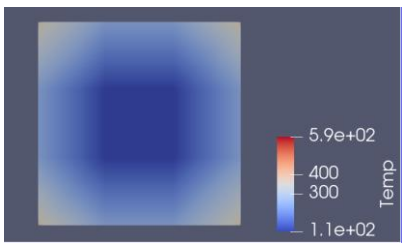
Wynik mojego programu:

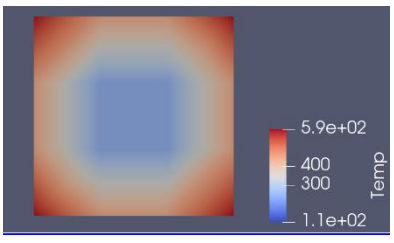
```
Time = 50, Min_t = 110.038, Max_t = 365.815

Time = 100, Min_t = 168.837, Max_t = 502.592

Time = 150, Min_t = 242.801, Max_t = 587.373

Time = 200, Min_t = 318.615, Max_t = 649.387

Time = 250, Min_t = 391.256, Max_t = 700.068

Time = 300, Min_t = 459.037, Max_t = 744.063

Time = 350, Min_t = 521.586, Max_t = 783.383

Time = 400, Min_t = 579.034, Max_t = 818.992

Time = 450, Min_t = 631.689, Max_t = 851.431

Time = 500, Min_t = 679.908, Max_t = 881.058
```

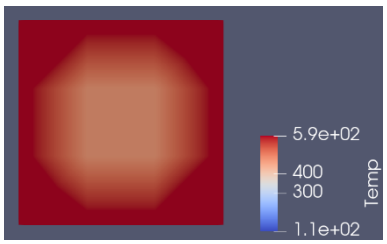
Wynik z UPEL:

```
4x4:
110.03797659406167 365.8154705784631
168.83701715655656 502.5917120896439
242.80085524391868 587.372666691486
318.61459376004086 649.3874834542602
391.2557916738893 700.0684204214381
459.03690325635404 744.0633443187048
521.5862742337766 783.382849723737
579.0344449687701 818.9921876836681
631.6892368621455 851.4310425916341
679.9075931513394 881.057634906017
```

Symulacja rozkładu temperatury w czasie za pomocą programu Paraview:







- Test2_4_4_MixGrid

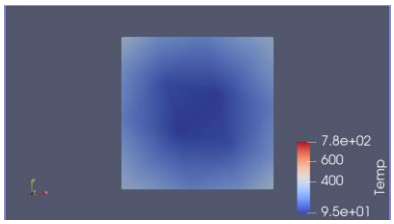
Wynik mojego programu:

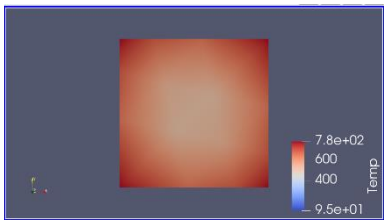
```
Time = 50, Min_t = 95.1518, Max_t = 374.686
Time = 100, Min_t = 147.644, Max_t = 505.968
Time = 150, Min_t = 220.164, Max_t = 586.998
Time = 200, Min_t = 296.736, Max_t = 647.286
Time = 250, Min_t = 370.968, Max_t = 697.334
Time = 300, Min_t = 440.56, Max_t = 741.219
Time = 350, Min_t = 504.891, Max_t = 781.21
Time = 400, Min_t = 564.002, Max_t = 817.392
Time = 450, Min_t = 618.174, Max_t = 850.237
Time = 500, Min_t = 667.766, Max_t = 880.168
```

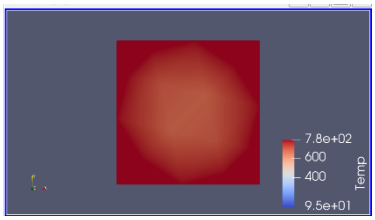
Wynik z UPEL:

```
4x4 mix:
95.15184673458245 374.6863325385064
147.64441665454345 505.96811082245307
220.1644549730314 586.9978503916302
296.7364399006366 647.28558387732
370.968275802604 697.3339863103786
440.5601440058566 741.2191121514377
504.8911996551285 781.209569726045
564.0015111915015 817.3915065469778
618.1738556427995 850.2373194670416
667.7655470268747 880.1676054000437
```

Symulacja rozkładu temperatury w czasie za pomocą programu Paraview:







- Test3_31_31_kwadrat

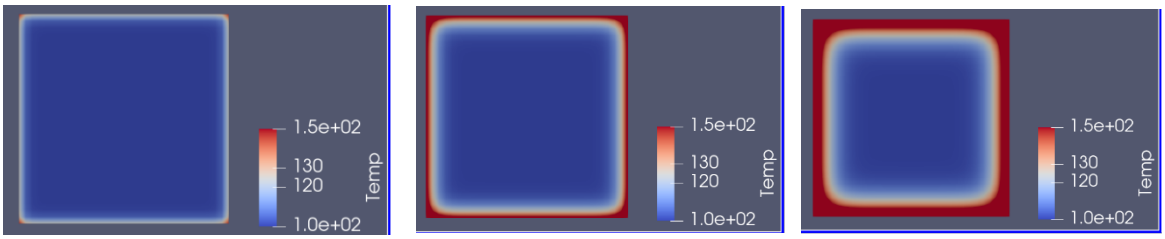
Wynik mojego programu:

```
Time = 1, Min_t = 100, Max_t = 149.557
Time = 2, Min_t = 100, Max_t = 177.445
Time = 3, Min_t = 100, Max_t = 197.267
Time = 4, Min_t = 100, Max_t = 213.153
Time = 5, Min_t = 100, Max_t = 226.683
Time = 6, Min_t = 100, Max_t = 238.607
Time = 7, Min_t = 100, Max_t = 249.347
Time = 8, Min_t = 100, Max_t = 259.165
Time = 9, Min_t = 100, Max_t = 268.241
Time = 10, Min_t = 100, Max_t = 276.701
```

Wynik z UPEL:

```
31x31:
99.99969812978378 149.5566275788947
100.00053467957446 177.44482649738018
100.00084733335379 197.2672291500534
100.00116712763896 213.15348263983788
100.00150209858216 226.6837398631218
100.001852708951 238.60869878203812
100.00222410506852 249.34880985057373
100.00263047992797 259.1676797521773
100.00310216686808 268.24376548847937
100.00369558647527 276.70463950306436
100.00450560745507 284.64527660833346
100.00567932588369 292.1386492100023
100.00742988613344 299.242260871447
100.01004886564658 306.00237684844643
100.01391592562979 312.4568735346492
100.01950481085419 318.637221302136
100.02738525124852 324.56990275925733
```

Symulacja rozkładu temperatury w czasie za pomocą programu Paraview:



- Test4_31_31_trapez

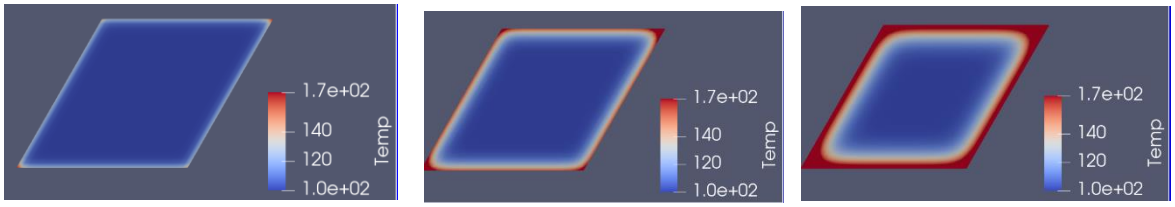
Wynik mojego programu:

```
Time = 1, Min_t = 100, Max_t = 166.936
Time = 2, Min_t = 100, Max_t = 207.233
Time = 3, Min_t = 100, Max_t = 236.287
Time = 4, Min_t = 100, Max_t = 259.465
Time = 5, Min_t = 100, Max_t = 279.031
Time = 6, Min_t = 100, Max_t = 296.121
Time = 7, Min_t = 100.001, Max_t = 311.385
Time = 8, Min_t = 100.001, Max_t = 325.235
Time = 9, Min_t = 100.003, Max_t = 337.951
Time = 10, Min_t = 100.005, Max_t = 349.731
Time = 11, Min_t = 100.01, Max_t = 360.723
Time = 12, Min_t = 100.018, Max_t = 371.04
Time = 13, Min_t = 100.03, Max_t = 380.771
Time = 14, Min_t = 100.047, Max_t = 389.987
Time = 15, Min_t = 100.072, Max_t = 398.747
```

Wynik z UPEL:

```
31x31 trapez:
99.99911415177323 166.9362149651147
99.99873673857435 207.23241215252318
99.99913622824398 236.28484836489392
99.99886349097673 259.4615454293437
99.99856368769406 279.02621207925847
99.99833307076041 296.11440465672047
99.99828777060439 311.37745804357013
99.9986389011151 325.22693428231423
99.999733155009 337.94169376607823
100.00209185746898 349.72071897716853
100.0064419788215 360.71173731817754
100.01373418237196 371.02790901578555
100.02514547372809 380.7581368613588
100.04206658948135 389.9737401779163
```

Symulacja rozkładu temperatury w czasie za pomocą programu Paraview:



Wnioski

Metoda Elementów Skończonych (MES) umożliwia przeprowadzanie symulacji dla elementów o złożonej geometrii - przy zmiennych wymiarach lub nieliniowych warunkach brzegowych. Zrozumienie tej metody wymaga wiedzy z różnych dziedzin: fizyki, mechaniki konstrukcji itp., a także zaawansowanej wiedzy matematycznej i informatycznej.

Na podstawie wykonanych symulacji można zaobserwować, jak zmienia się rozkład temperatury w węzłach siatki na obiekcie w przestrzeni 2D. Uzyskane wyniki pozwoliły nam zidentyfikować główne cechy tego rozkładu, obserwując różnice temperatur pomiędzy

obszarami centralnymi a brzegowymi. Warunki brzegowe odegrały kluczową rolę w kształtowaniu temperatury, szczególnie w obszarach blisko granic modelu.

MES jest kluczowym narzędziem wykorzystywanym przez inżynierów do rozwiązywania problemów obliczeniowych i przeprowadzania badań opartych na przykład na:

• przepływie ciepła lub płynów,

• naprężeniach,

• symulacjach deformacji.

Należy pamiętać, że MES to metoda przybliżona. Wyniki zawsze należy poddać weryfikacji. Aby zwiększyć dokładność uzyskanych pomiarów, zaleca się użycie gęstszej siatki lub zmniejszenie kroku czasowego. Jednak im bardziej szczegółowe są warunki początkowe, tym większa jest złożoność obliczeń i tym samym - zwiększa się czas obliczeń.