Imię i nazwisko Jakub Nowak	Wydział WIMiIP	Kierunek Informatyka Techniczna
Grupa 4	Sprawozdanie z laboratoriów	

Spis treści

1. Wstęp teoretyczny

W ramach omawianego projektu opracowano oprogramowanie **metody elementów skończonych,** służące do symulacji procesu nieustalonego transportu ciepła z warunkiem brzegowym konwekcji dla siatek 2D. Do poprawnego opisu interesującego nas zjawiska konieczne było poznanie jego podstaw teoretycznych:

a. Zjawiska cieplne zachodzące w **stanie nieustalonym** opisuje równanie Fouriera w postaci:

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

b. Rozwiązanie tego zagadnienia sprowadza się do zadania polegającego na poszukiwaniu minimum takiego funkcjonału, dla którego równanie to będzie równaniem Eulera. Według rachunku wariacyjnego i po narzuceniu warunków brzegowych (poprzez dodanie do funkcjonału odpowiednich całek) będzie miał on postać:

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

c. Podczas tworzenia naszego oprogramowania przyjęliśmy następujące założenia:

1) W projekcie uwzględniamy jedynie dwa warunki brzegowe, a mianowicie:
   - na powierzchni jest zadana temperatura *t*;
   - na powierzchni zadany jest strumień ciepła *q* według prawa konwekcji:

$$k(t)\left(\frac{\partial t}{\partial x}a_x+\frac{\partial t}{\partial y}a_y+\frac{\partial t}{\partial z}a_z\right)=\alpha_{konw}(t-t_\infty),$$

2) Nie uwzględniamy prędkości generowania ciepła *Q*
3) Zakładamy, że rozpatrywany przez nas materiał wykazuje izotropię w zakresie przewodnictwa cieplnego - jego właściwości są takie same

niezależnie od kierunku, w którym są mierzone (współczynniki przewodzenia ciepła: $k_x = k_y = k_z = k(t)$)

4) Istnieje kilka możliwości rozwiązania układu równań dla niestacjonarnego procesu w zależności od tego w jakiej chwili czasu będziemy rozpatrywać wektor {t}. Przyjmujemy, że {t}={t_1} ({t}={t_0} powoduje słabą stabilność rozwiązań dla różnych $\Delta\tau$), dlatego otrzymujemy niejawny schemat wyznaczania temperatury.

d. Po dokonaniu dyskretyzacji problemu (zastąpieniu temperatury w postaci dyskretnej na temperaturę w postaci ciągłej korzystając z interpolacji) oraz wzięciu pod uwagę wszystkich powyższych założeń układ równań, który musimy rozwiązać (w postaci macierzowej) wygląda następująco:

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

gdzie:

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

$$[C]=\int_V c\rho\{N\}\{N\}^T dV\,. \qquad \{P\}=-\int_S \alpha\{N\}t_\infty dS$$

W przy naszego programu obliczamy temperaturę dla obiektu 2D z tego względu ostatni człon macierzy H (odnoszący się do zmiennej *z*) odpuszczamy. Znalezienie temperatury t_1 sprowadza się do rozwiązania powyższego układu równań. W tym celu będziemy korzystać z metody Gaussa rozwiązywania układów równań:

Rozwiązywanie układu równań metodą Gaussa - Crouta

Metoda eliminacji Gaussa opiera się o tworzenie macierzy współczynników układu, która jest następnie przekształcana do postaci macierzy trójkątnej górnej i dolnej jednocześnie. Algorytm eliminacji Gaussa posiada dosyć istotną wadę - bazuje on na dzieleniu przez elementy przekątnej głównej macierzy współczynników. Może to doprowadzić do dzielenia przez zero, dlatego w programie korzystam z modyfikacji tej metody, która powoduje, że dzielnik będzie posiadał największą na moduł wartość i nie dojdzie do sytuacji, gdy będzie on posiadał wartość zero.

Do rozwiązywania powyższych całe będziemy korzystać z numerycznej metody:

Całkowanie metodą Gaussa

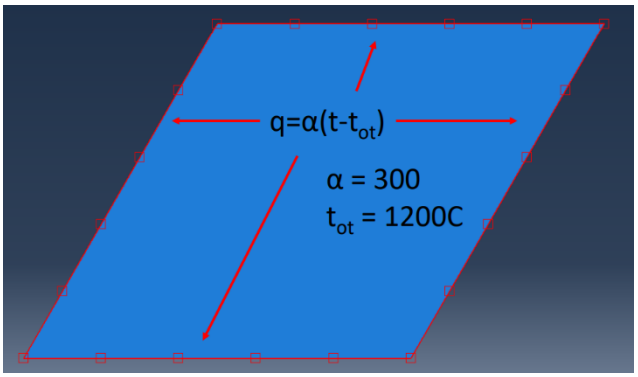
Metoda całkowania Gaussa, znana również jako kwadratura Gaussa, jest techniką numerycznego przybliżania całek, szczególnie przydatną w przypadku całek o skomplikowanych kształtach funkcji. Idea tej metody opiera się na wyborze odpowiednich punktów zwanych **węzłami kwadratury** oraz **wag** dla tych punktów. Odbywa się w przedziale <-1; 1>, a schematy całkowania zostały opracowane oraz stabelaryzowane. Wzór, którym będziemy się posługiwać w celu obliczenia interesujących nas macierzy obejmuje całkowanie w przestrzeni 2D i wygląda następująco:

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta)d\xi\,d\eta=\sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(\xi_i,\eta_j)$$

2. Opis modelu z warunkami brzegowymi

Jak wspominałem wcześniej w tworzeniu oprogramowania uwzględniany jest **warunek brzegowy pierwszego rodzaju**, który mówi o tym, że znany jest rozkład temperatury na powierzchni ciała (znamy temperaturę początkową w całej objętości ciała).

Pod uwagę brany jest także **warunek brzegowy trzeciego rodzaju**, ponieważ znana jest temperatura otoczenia oraz wartość współczynnika przejmowania ciepła. Prócz wspomnianych wyżej wartości posiadamy takie informacje o modelu jak: całkowity czas symulacji, krok czasowy, współczynnik przewodzenia ciepła, ciepło właściwe materiału i jego gęstość oraz ilość elementów i węzłów.

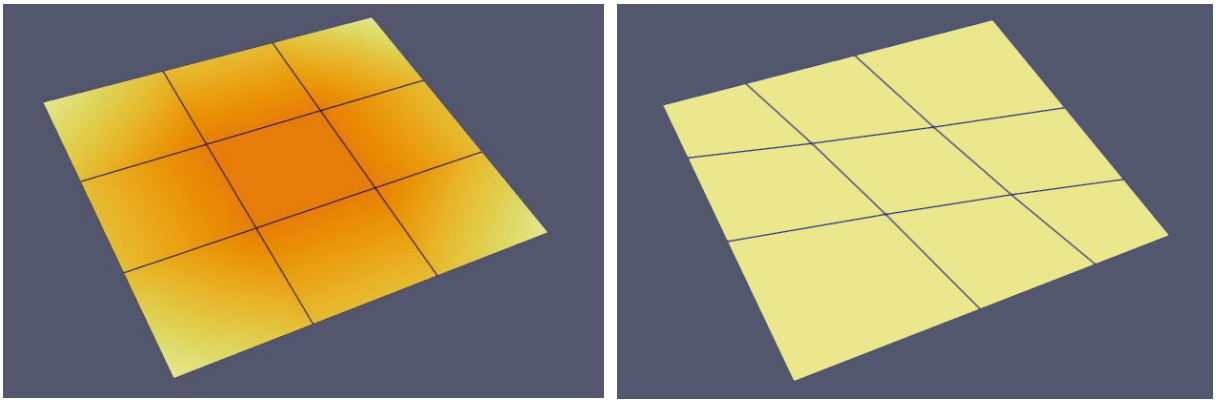



```
SimulationTime 500
SimulationStepTime 50
Conductivity 25
Alfa 300
Tot 1200
InitialTemp 100
Density 7800
SpecificHeat 700
Nodes_number 16
Elements_number 9
```

W ramach projektu rozpatrujemy 4 różne siatki 2D:

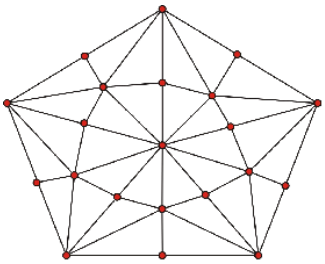
- Kwadratowa siatka 4 x 4 węzły (9 elementów)
- Kwadratowa siatka 4 x 4 węzły o nieregularnych kształtach elementów skończonych
- Kwadratowa siatka 31 x 31 węzłów (900 elementów)
- Siatka 31 x 31 węzłów w kształcie trapezu

Wszystkie z nich posiadają 4-węzłowe elementy skończone.



Powyżej przykład siatek 4 x 4

3. Opis MES



Metoda elementów skończonych (MES) to numeryczna technika analizy stosowana do rozwiązania różnych problemów inżynierskich i naukowych, takich jak analiza naprężeń, przewodnictwa cieplnego, dynamiki struktur, elektromagnetyzmu itp. Metoda ta pozwala na modelowanie złożonych struktur przy użyciu skończonych elementów, co umożliwia przybliżoną analizę zachowań fizycznych. MES jest metodą przybliżoną i jej stosowanie wymaga odpowiedniej wiedzy teoretycznej.
Metoda ta składa się z następujących etapów:

1. **Przygotowanie modelu -** w pierwszym etapie należy przygotować model numeryczny obiektu, który ma być zbadany. Model ten opisuje geometrię i właściwości materiałowe obiektu, a także warunki brzegowe i obciążenia, które będą na nim działać.
2. **Podział obiektu na elementy skończone -** następnie obiekt jest podzielony na małe elementy skończone, które można opisać matematycznie.
3. **Wyznaczenie równań -** po podziale obiektu na elementy skończone należy wyznaczyć równania, które opisują zachowanie się obiektu w określonych , należy wziąć pod uwagę m.in. własności materiału oraz warunki brzegowe.
4. **Rozwiązanie równań –** numerycznie rozwiązywane równania pozwalają na określenie rozkładu naprężeń, odkształceń, temperatur i innych charakterystyk obiektu w różnych punktach.
5. **Analiza wyników -** ostateczny etap analizy MES polega na analizie wyników i wyciągnięciu z nich wniosków.

Zalety:

- Podstawową zaletą MES jest możliwość uzyskiwania rozwiązań dla obszarów o skomplikowanych kształtach, dla których nie jest możliwe przeprowadzenie ścisłych obliczeń analitycznych.
- Możliwość symulacji różnych warunków - analiza MES pozwala na symulowanie różnych warunków, co jest trudne lub niemożliwe do wykonania w badaniach doświadczalnych.
- Możliwość analizy niestandardowych geometrii. Analiza MES umożliwia analizę niestandardowych geometrii, które są trudne do zbadania w inny sposób.

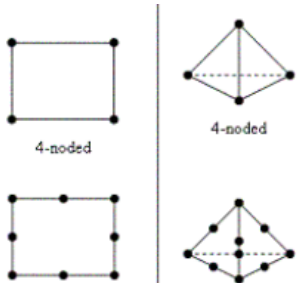
Wady:

- Złożoność i koszty obliczeniowe - metoda jest złożonym i czasochłonnym procesem obliczeniowym, który wymaga znacznych zasobów obliczeniowych. W przypadku skomplikowanych modeli, proces ten może trwać wiele godzin lub dni, co z kolei może prowadzić do wysokich kosztów obliczeniowych.
- Ryzyko błędów numerycznych. W przypadku niepoprawnego ustawienia parametrów analizy MES lub nieodpowiedniego modelu matematycznego, istnieje ryzyko wystąpienia błędów numerycznych, które mogą prowadzić do niewłaściwych wyników.

Bardziej szczegółowo MES opiera się na:

Podziale (dyskretyzacji) układu na pewną ilość elementów skończonych

Element skończony – prosta figura geometryczna (płaska lub przestrzenna), dla której określone zostały wyróżnione punkty zwane węzłami. Węzły znajdują się w wierzchołkach elementu skończonego, mogą być również na bokach i wewnętrzu.



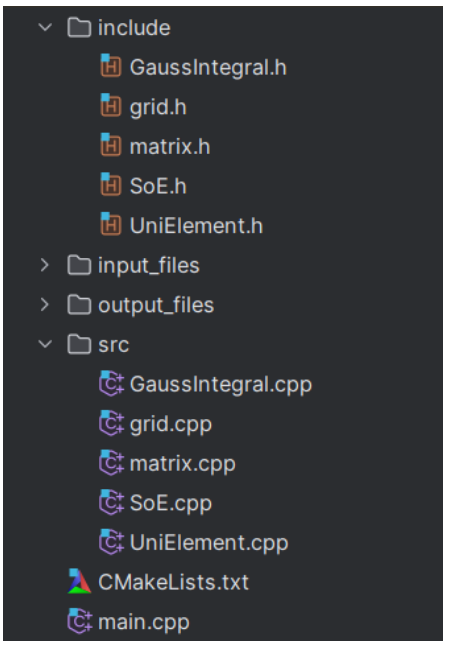
Zastąpieniu układu równań różniczkowych układem równań algebraicznych (zmienne ciągłe wyraża się za pomocą wartości węzłowych oraz funkcji kształtu).

Funkcja kształtu mają szczególne własności: suma wartości funkcji kształtu w danym punkcie wynosi 1, funkcja kształtu może przybierać wartości z przedziału [0:1], funkcje kształtu są ciągłe na granicach pomiędzy elementami.

$$N_1(x)=(x_2-x)/(x_2-x_1)$$
$$N_2(x)=(x-x_1)/(x_2-x_1)$$

4. Charakterystyka kodu

Program został napisany w języku C++, jego struktura wygląda następująco:



- **main** – główny plik programu
- **grid** – plik posiadający klasy przechowujące dane globalne (temperatura początkowa, współczynniki itp.) elementy oraz węzły siatki jak i klasę, która je agreguje w jedną całość, znajdują tu się także funkcje zapisujące i odczytujące dane z plików.
- **matrix** – jest to klasa pozwalając prowadzić obliczenia na macierzach, klasa ta jest autorstwa dr inż. Łukasza Sztangreta i została użyta na podstawie licencji CC BY-SA 3.0
- **SoE** – klasa umożliwia agregacje macierzy do jednego głównego układu równań oraz rozwiązanie tego układu.
- **UniElement** – klasa „uniwersalna element" pozwala obliczyć macierze H, C oraz wektor P dla każdego elementu w układzie.
- **GaussIntegral** – klasa pozwala na całkowanie metodą kwadratur Gaussa.

Napisane oprogramowanie składa się z kilku kluczowych etapów:

1. Wczytywanie danych z pliku i wstawianie ich w odpowiednie struktury (GlobalData, Node, Element, Grid).
2. Obliczenie macierzy (H, Hbc, C, P) potrzebnych do stworzenia układu równań (dla każdego elementu siatki osobno). W tym celu konieczne jest obliczanie jakobianów oraz skorzystanie z klasy *GaussIntegral* przechowującej wagi oraz współrzędne punktów potrzebne do całkowania.
3. Zagregowanie obliczonych macierzy do jednej macierzy głównej oraz rozwiązanie układu równań dla każdego kroku czasowego.
4. Zapis danych do pliku w odpowiedni sposób – tak żeby mógł być on zinterpretowany przez postprocesor.

Cały ten schemat postępowania przedstawiony jest poniżej w funkcji głównej programu:

```
int main() {
    //------------------------ DANE ----------------------------
    //plik do odczytu
    ifstream inputFile(path.c_str());
    if (!inputFile.good())
        cerr << "Blad otwarcia pliku" << endl;

    //siatka
    Grid grid;
    GlobalData globalData;

    //1. wczytywanie z pliku (wszystkiego)
    readFromFile(inputFile, globalData, grid);
    inputFile.close();

    //element uniwersalny dla okreslonej ilosci pkt calkownia
    UniElement uElement(gaussPoints);

    //---------------------- OBLICZENIA -------------------------
    try{
        //2. obliczanie macierzy H, C, Hbc i wektora P
        for (int i = 0; i < globalData.Elements_number; ++i){
            uElement.calculateH_C(grid.elements[i], globalData);
            uElement.calculateHbc_P(grid.elements[i], globalData);
            grid.elements[i].calculateGlobalH(globalData);
        }
        SoE systemOfEq(globalData);

        //3. agregacja i obliczanie ukladu rownan
        for (int i = 0; i < globalData.Elements_number; ++i)
            systemOfEq.aggregation(grid.elements[i]);
        try{
            //4. obliczenie wyniku i zapis do pliku!
            systemOfEq.calcuateResult(globalData, grid);
            //test
            systemOfEq.test(t1_max, t1_min);
        }
        catch(string e) {
            cerr<<"Uklad rownan --> "<<e<<endl;
        }

    }catch (string e){
        cerr<<"Obliczenia: "<<e<<endl;
    }
    return 0;
}
```

Obliczanie macierzy H i C

Jednym z najważniejszych i najbardziej skomplikowanych etapów jest obliczenie macierzy H oraz C, macierze te zawierają w sobie współczynniki układu równań.

Macierz H posiada informacje opisuje współczynniki wymiany ciepła przez przewodzenie między punktami w układzie. Opisuje więc wpływ przewodzenia ciepła na temperatury w węzłach układu, ale opis ten jest tylko dla wnętrza obiektu (nie uwzględnia warunków brzegowych).

Macierz C mówi natomiast o pojemności cieplnej badanego materiału, czyli zdolności do przechowywania energii cieplnej. Jest to jedyny element (jeśli chodzi o rozpatrywany przez

nas przypadek) zależny od czasu – nie branie pod uwagę tej macierzy sprawi, że rozwiązywać będziemy proces ustalony.

Opis wyznaczania macierzy H:

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

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

Do obliczenia macierzy H konieczne są pochodne funkcji kształtu po x oraz y obliczane z wykorzystaniem poniższego wzoru:

Obliczanie wartości **funkcji kształtu względem ksi oraz względem eta** (ostatniego członu równania) odbywa się w konstruktorze klasy ***UniElement*** i wykorzystywane są do tego współrzędne punktów całkowania kwadratury Gaussa oraz funkcje reprezentujące pochodne. W wyniku tych działań powstają macierze, których liczba wierszy jest zależna od ilości punktów całkowania. Przykład pochodnej:

```
//pochodne funkcji kształtu --> n to eta_dN, e to ksi_dN
static double d1de(double n){return (-1./4.) * (1. - n);}
```

Następnie dane te wykorzystywane są w celu wyznaczenia jakobianu i odwrotności jego współczynnika, jakobian liczymy korzystając ze wzory na interpolacyjnego oraz współrzędnych elementu skończonego (wzory obok).

$$x = \sum_{i=1}^{np}(N_i x_i) = N_1x_1 + N_2x_2 + N_3x_3 + N_4x_4$$

$$\frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi}x_1 + \frac{\partial N_2}{\partial \xi}x_2 + \frac{\partial N_3}{\partial \xi}x_3 + \frac{\partial N_4}{\partial \xi}x_4$$

Po obliczeniu pochodnych korzystamy ze wzoru na H - transponujemy jedną z powstałych macierzy, przemnażamy przez siebie i dodajemy do siebie człony *x* i *y*. Na koniec mnożymy wszystko przez wyznacznik jakobianu (odpowiadający *dV*) oraz współczynnik przewodzenia ciepła (*k(t)*). W ten sposób otrzymujemy macierze H dla każdego punktu całkowania, aby uzyskać właściwą macierz H dla elementu musimy wykorzystać wzór całkowania metodą Gaussa przemnażając macierze przez odpowiednie wagi oraz sumując wszystko.

Obliczanie obu macierzy (H oraz C) jest dokonywane w jednej funkcji (*calculateH_C()*) ze względu na podobieństwo ich liczenia. Obie macierze korzystają z tych samych wartości wyznaczników jakobianów – wyznaczanie macierzy C jest jednak nieco prostsze, gdyż nie wymaga wyliczania pochodnych. Korzystamy natomiast z wartości funkcji kształtu względem ksi i eta – macierze te również obliczane są w konstruktorze w podobny sposób jak było to dla pochodnych. Przykładowa funkcja kształtu 2D

```
//funkcje kształtu --> n to eta, e to ksi
static double N1(double e, double n){return (1./4.) * (1. - e) * (1 - n);}
```

Opisywana funkcja wygląda następująco:

```
void UniElement::calculateH_C(Element& e, const GlobalData& gd) {
    //zmienne lokalne - wektor macierzy H, dx i dy
    matrix dX(1, dN);
    matrix dY(1, dN);
    matrix resultX, resultY, H_pc, H;    //H
    matrix C_pc, C;                      //C
    vector <Jacobian> jacobians(node, Jacobian(2, 2));
    //------------------------- OBLICZANIE JAKOBIANU ---------------------
    for (int i = 0; i < node; ++i) {          //node = wiersze
        for (int j = 0; j < dN; ++j) {       //4 bo jakobian 2x2 - kolumny
            jacobians[i].J(0, 0) += (ksi_dN(i, j) * e.nodes[j].x);
```

```
            jacobians[i].J(0, 1) += (ksi_dN(i, j) * e.nodes[j].y);
            jacobians[i].J(1, 0) += (eta_dN(i, j) * e.nodes[j].x);
            jacobians[i].J(1, 1) += (eta_dN(i, j) * e.nodes[j].y);
        }
        jacobians[i].detJ = det(jacobians[i].J);
        jacobians[i].Jinv = inv(jacobians[i].J);
    }

    //----------- OBLICZANIE dN/dx i dN/dy H oraz C -----------------
    int N = -1;      //numer wiersza
    int M = -1;      //numer kolumny
    for (int i = 0; i < node; ++i) {
        if(N == (Npc - 1)) N = 0;
        else N++;
        if(i%Npc == 0) M++;

        for (int j = 0; j < dN; ++j) {
            dX(0, j) = jacobians[i].Jinv(0, 0) * ksi_dN(i, j) +
            jacobians[i].Jinv(0, 1) * eta_dN(i, j);
            dY(0, j) = jacobians[i].Jinv(1, 0) * ksi_dN(i, j) +
            jacobians[i].Jinv(1, 1) * eta_dN(i, j);
        }
        resultX = trans(get_row(dX,0)) * get_row(dX,0);
        resultY = trans(get_row(dY,0)) * get_row(dY,0);

        H_pc = gd.Conductivity * jacobians[i].detJ * (resultX + resultY);
        H = H + (integral.getA(N) * integral.getA(M)) * H_pc;

        //--------------- C ----------------
        //wszystko co powyzej dotyczy macierzy H!!
        C_pc = gd.Density * gd.SpecificHeat * jacobians[i].detJ *
        (trans(get_row(ksiEta_N,i)) * get_row(ksiEta_N, i));
        C = C + (integral.getA(N) * integral.getA(M)) * C_pc;
    }
    e.H = H; e.C = C;
}
```

Obliczanie macierzy Hbc i wektora P (warunku brzegowego)

Podobnie sprawa ma się w kontekście obliczania warunku brzegowego, zarówno **macierz Hbc** jak i **wektor P** obliczane są w jednej funkcji (*calculateHbc_P()*). Jest to spowodowane tym, że również wykorzystywany jest wspólny wyznacznik jakobianu oraz wartość współczynnika konwekcyjnej wymiany ciepła. Podział warunku brzegowego na dwie części jest podyktowany natomiast tym, że jedna z jego części odnosi się do wartości temperatury będącej niewiadomą układu równań, druga zaś część zawiera temperaturę otoczenia, która u nas jest stała przez cały czas trwania symulacji. W związku z tym dla ułatwienia implementacji warunek brzegowy jest podzielony – macierz Hbc zostanie dodana do macierzy H natomiast wektor P stanowić będzie wyrazy wolny układu. Rozpatrywany warunek brzegowy przedstawiony jest następującym wzorem:

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

Do obliczenia omawianych macierzy wykorzystane zostały poniższe wzory:

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

W związku charakterystyką waruneku brzegowego możemy go rozpatrywać jedynie na krawędzi obiektu, a więc w węzłach mających bezpośredni styk z otoczeniem. Aby to zrealizować z pliku z danymi początkowymi wczytywane zostały flagi mówiące, czy węzeł leży na krawędzi czy nie. Wyznaczanie Hbc oraz P odbywa się więc jedynie dla brzegowych węzłów.

Najpierw obliczany jest wyznacznik jakobianu według wzory ***L/2***, gdzie *L* to długość rozpatrywanej krawędzi. Następnie wykorzystywane są wartości funkcji kształtu względem ksi i eta dla każdej rozpatrywanej krawędzi, które zostały wyliczone i zapisane do macierzy w konstruktorze. Potem (podobnie jak miało to miejsce podczas obliczania macierzy H) poszczególne macierze Hbc oraz P dla punktow całkowania są sumowane i przemnażane przez wagi.

W przypadku warunku brzegowego otrzymujemy tyle macierzy Hbc oraz tyle wektorów P ile jest w elemencie krawędzi zewnętrznych, żeby otrzymać dla elementu jedną wynikową macierz musimy jeszcze wszystko zsumować (oczywiście P i Hbc osobno). Aby warunek był poprawny musimy jeszcze przemnożyć go przez konwekcyjny współczynnik wymiany ciepła oraz wyznacznik jakobianu, a wektor P mnożymy także przez temperature otoczenia.

```
void UniElement::calculateHbc_P(Element& e, const GlobalData& gd){

    matrix tmp, H_pc, Hbc;
    matrix P(dN, 1), P_pc;
    int first, second;

    for (int i = 0; i < dN; ++i) {
        //ustawianie odpowiednich punktów
        first = i;
        if(i == (dN-1))
            second = 0;
        else
            second = i+1;

        //Tylko dla brzegowych krawędzi  //wzór na detJ: L / 2
        if((e.nodes[first].BC == 1) && (e.nodes[second].BC == 1)){
         double detJ = sqrt(pow((e.nodes[first].x - e.nodes[second].x), 2)
                   + pow((e.nodes[first].y - e.nodes[second].y), 2)) / 2.;

            for (int j = 0; j < Npc; ++j) {
              //Hbc
              tmp = trans(get_row(surface[i],j)) * get_row(surface[i],j);
              H_pc = H_pc + integral.getA(j) * tmp;
              //P
              P_pc = P_pc + integral.getA(j)* trans(get_row(surface[i],j));
            }

            double mult = gd.Alfa * detJ;
            Hbc =  mult * H_pc + Hbc;
            P = gd.Tot *  mult * P_pc + P;
            //zerowanie
            H_pc = H_pc - H_pc ;
            P_pc = P_pc - P_pc;
        }
    }
    e.P = P;
    e.Hbc = Hbc;
}
```

Agregacja i obliczanie układu równań:

Ostatnim znaczącym elementem oprogramowania jest etap sumowania uzyskanych wyników do dwóch globalnych macierzy ze współczynnikami układu równań (*globalH* i *globalC*) oraz do jednego wektora z wyrazami wolnymi (*globalP*) . Robimy to w klasie ***Soe*** (*System of Equations*) w dosyć prosty sposób – należy wykorzystać wczytane wcześniej z pliku ID każdego węzła. W związku z tym, że niektóre węzły są wspólne dla kilku elementów to tworząc globalne macierze należy dodawać do siebie wartości wpisywane do danej komórki z wartościami, które już się tam znajdowały (początkowo macierze wypełnione są zerami). Kod przedstawiający agregacje wygląda następująco:

```
void SoE::aggregation(const Element &e) {

    int* n = get_size(e.H); //n[0] - liczba wierszy, n[1] - kolumn
    for (int i = 0; i < n[0]; ++i) {
        for (int j = 0; j < n[1]; ++j) {
          globalH(e.nodes[i].node_id-1, e.nodes[j].node_id-1) += e.H(i, j);
          globalC(e.nodes[i].node_id-1, e.nodes[j].node_id-1) += e.C(i, j);
        }
        globalP(e.nodes[i].node_id-1, 0) += e.P(i, 0);
    }
}
```

Po agregacji można przystąpić do rozwiązywania układu równań przedstawionego wzorem:

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

Najpierw obliczane są elementy stałe (niezmienne w trakcie symulacji), czyli wartości współczynników oraz wyrazów wolnych. Macierz C dzielona jest przez wartość kroku czasowego (z danych globalnych), następnie wartości te dodawane są macierzy H i powstaje finalna macierz współczynników układu. Następnie ta sama macierz mnożona przez wektor temperatur początkowych (również odczytany z danych globalnych i zainicjalizowany w konstruktorze) daje w wyniku wektor, który dodawany jest do wektora P – tak powstają wyrazy wolne.

Układ równań obliczany jest metodą Gaussa-Crouta dla każdej chwili symulacji (ilość iteracji to ogólny czas symulacji dzielony przez wartość kroku czasowego). Podczas symulacji zmienia się wartość temperatury początkowej $\{t_0\}$ – jest to wynik układu równań z poprzedniej chwili czasowej zapisywany do wektora X. W każdej iteracji konieczne jest ponowne obliczenie wyrazów wolnych. W poniższej funkcji część kodu (dla jego przejrzystości) odpowiedzialna za zapisu do pliku została usunięta.

```
// ------------------------------------------------------------
// | UKLAD ROWNAN: ([H] * [C]/dT)*{t1} - ([C]/dT)*{t0} + {P} |
// ------------------------------------------------------------
void SoE::calcuateResult(const GlobalData& gd, const Grid& grid) {
    int symStep = (int)gd.SimulationStepTime;
    globalC = globalC/symStep;      // macierz [C]/dT   //T=tał(czas)
    globalH = globalH + globalC;    // ([H] * [C]/dT) * {t1} - układ rownan

    for (int i = symStep; i <= gd.SimulationTime; i+=symStep) {
        X = globalC * X + globalP; // ([C]/dT)*{t0} + {P} - wyrazy wolne
        this->Gauss_Crout();
        // zapis do pliku …
        cout<<endl<<"Wyniki dla t: "<<i<<"s"<<endl;
        this->print();
    }
}
```

5. Porównanie wyników oprogramowania z testami

W celu przetestowania wyników oprogramowania stworzyłem prostą funkcje porównującą rezultat z wynikami z pliku zawierającego wartości maksymalne i minimalne dla każdej siatki. Testowane jest 10 pierwszych otrzymanych wartości, funkcja przerywa działanie program jeśli wyniki nie są zgodne w granicach określonej dokładności (*epsilon*). Testowanie przeprowadziłem dla wyników obliczonych z użyciem **2 punktów całkowania**.

```
void SoE::test(double *tMax, double *tMin){
    int* n = get_size(X);
    double epsilon = 1e-04;
    for (int i = 0; i < 10; ++i) {  // testowanie 10 pierwszych wartosci
        if(abs(resultMin[i] - tMin[i]) > epsilon ||
           abs(resultMax[i] - tMax[i]) > epsilon ){
            cerr<<"Blad - wyniki nie sa zgodne"<<endl;
            exit(0);
        }else{
            cout<<setprecision(10)<<"Min: "<<resultMin[i]<<" | "<<tMin[i]
                <<"\tMax: "<<tMax[i]<<" | "<<resultMax[i]<<endl;
        }
    }
    cout<<endl<<"Test OK"<<endl;
}
```

Test dla siatki 4x4:

```
C:\Users\48602\Desktop\SemestrV\MES\Projekt_MES\cmake-build-debug-mingw\Projekt_MES.exe
----------------TESTOWANIE Z DOKLADNOSCIA: 0.0001----------------
     PROGRAM       TEST             PROGRAM       TEST
Min: 110.0379724 | 110.0379766  Max: 365.8154706 | 365.8154726
Min: 168.8370098 | 168.8370172  Max: 502.5917121 | 502.5917143
Min: 242.8008463 | 242.8008552  Max: 587.3726667 | 587.3726667
Min: 318.6145887 | 318.6145938  Max: 649.3874835 | 649.3874822
Min: 391.2557918 | 391.2557917  Max: 700.0684204 | 700.0684183
Min: 459.0369089 | 459.0369033  Max: 744.0633443 | 744.0633415
Min: 521.5862854 | 521.5862742  Max: 783.3828497 | 783.3828462
Min: 579.0344614 | 579.034445   Max: 818.9921877 | 818.9921836
Min: 631.6892582 | 631.6892369  Max: 851.4310426 | 851.4310378
Min: 679.9076191 | 679.9075932  Max: 881.0576349 | 881.0576294

Test OK
```

Test dla siatki 4x4 mix:

```
C:\Users\48602\Desktop\SemestrV\MES\Projekt_MES\cmake-build-debug-mingw\Projekt_MES.exe
----------------TESTOWANIE Z DOKLADNOSCIA: 1e-05----------------
     PROGRAM       TEST             PROGRAM       TEST
Min: 95.151849 | 95.15184673    Max: 374.6863325 | 374.6863332
Min: 147.6444191 | 147.6444167  Max: 505.9681108 | 505.9681113
Min: 220.1644558 | 220.164455   Max: 586.9978504 | 586.9978493
Min: 296.7364384 | 296.7364399  Max: 647.2855839 | 647.2855822
Min: 370.9682724 | 370.9682758  Max: 697.3339863 | 697.3339845
Min: 440.5601436 | 440.560144   Max: 741.2191122 | 741.2191101
Min: 504.8912021 | 504.8911997  Max: 781.2095697 | 781.2095686
Min: 564.0015163 | 564.0015112  Max: 817.3915065 | 817.3915046
Min: 618.1738633 | 618.1738556  Max: 850.2373195 | 850.2373168
Min: 667.7655569 | 667.765547   Max: 880.1676054 | 880.1676019

Test OK
```

Test dla siatki 31x31:

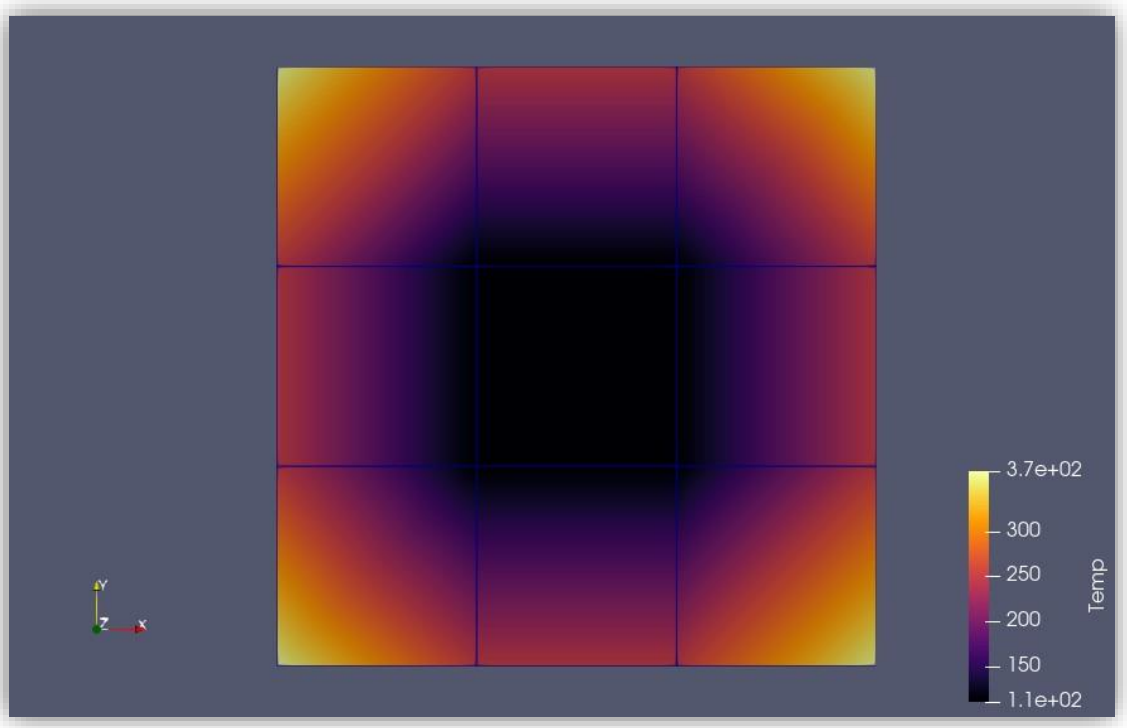
```
C:\Users\48602\Desktop\SemestrV\MES\Projekt_MES\cmake-build-debug-mingw\Projekt_MES.exe
-----------------TESTOWANIE Z DOKLADNOSCIA: 0.01-----------------
     PROGRAM       TEST              PROGRAM       TEST
Min: 100 | 99.99969813  Max: 149.5566276 | 149.5569518
Min: 100 | 100.0005347  Max: 177.4448265 | 177.444928
Min: 100.0000001 | 100.0008473  Max: 197.2672292 | 197.2669629
Min: 100.0000003 | 100.0011671  Max: 213.1534826 | 213.1527873
Min: 100.0000016 | 100.0015021  Max: 226.6837399 | 226.6825834
Min: 100.0000065 | 100.0018527  Max: 238.6086988 | 238.6070648
Min: 100.0000215 | 100.0022241  Max: 249.3488099 | 249.3466919
Min: 100.0000622 | 100.0026305  Max: 259.1676798 | 259.1650792
Min: 100.0001598 | 100.0031022  Max: 268.2437655 | 268.240689
Min: 100.0003713 | 100.0036956  Max: 276.7046395 | 276.7010979

Test OK
```

Test dla siatki 31x31 trapez:

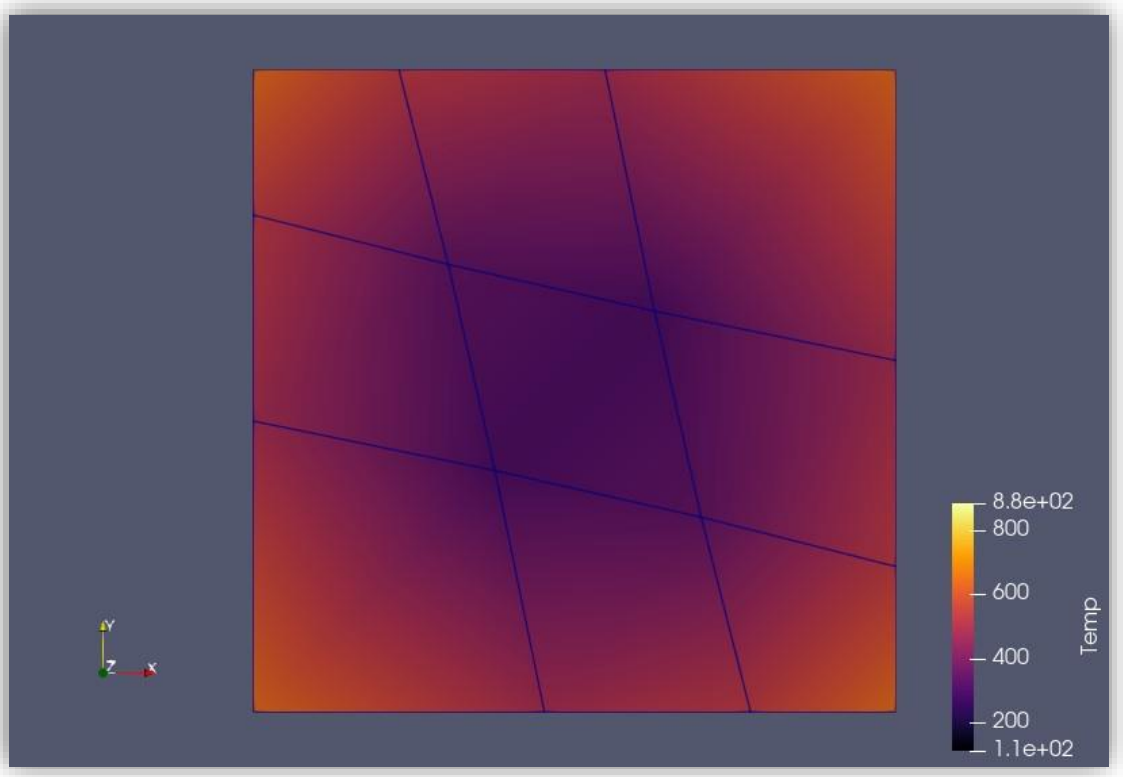
```
C:\Users\48602\Desktop\SemestrV\MES\Projekt_MES\cmake-build-debug-mingw\Projekt_MES.exe
-----------------TESTOWANIE Z DOKLADNOSCIA: 0.1-----------------
     PROGRAM       TEST              PROGRAM       TEST
Min: 100 | 99.99911415  Max: 166.936215 | 166.9357382
Min: 100.0000004 | 99.99873674  Max: 207.2324122 | 207.2333219
Min: 100.0000027 | 99.99913623  Max: 236.2848484 | 236.2872276
Min: 100.0000142 | 99.99886349  Max: 259.4615454 | 259.4652944
Min: 100.0000563 | 99.99856369  Max: 279.0262121 | 279.031226
Min: 100.0001828 | 99.99833307  Max: 296.1144047 | 296.1205956
Min: 100.0005072 | 99.99828777  Max: 311.377458 | 311.3847538
Min: 100.0012387 | 99.9986389   Max: 325.2269343 | 325.2352745
Min: 100.0027228 | 99.99973316  Max: 337.9416938 | 337.9510266
Min: 100.0054802 | 100.0020919  Max: 349.720719 | 349.7309984

Test OK
```

Przykładowe wyniki z postprocesora:

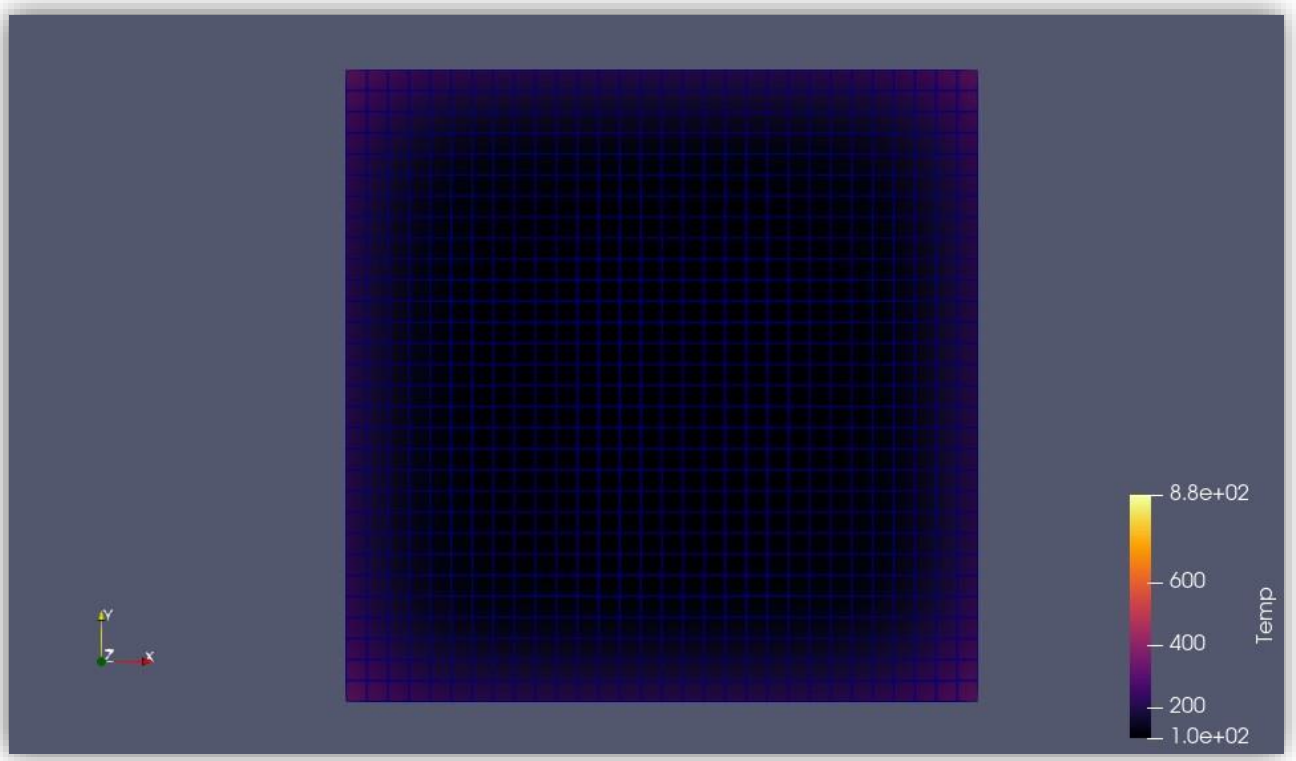
Siatka 4x4 – stan początkowy (po 1 s)



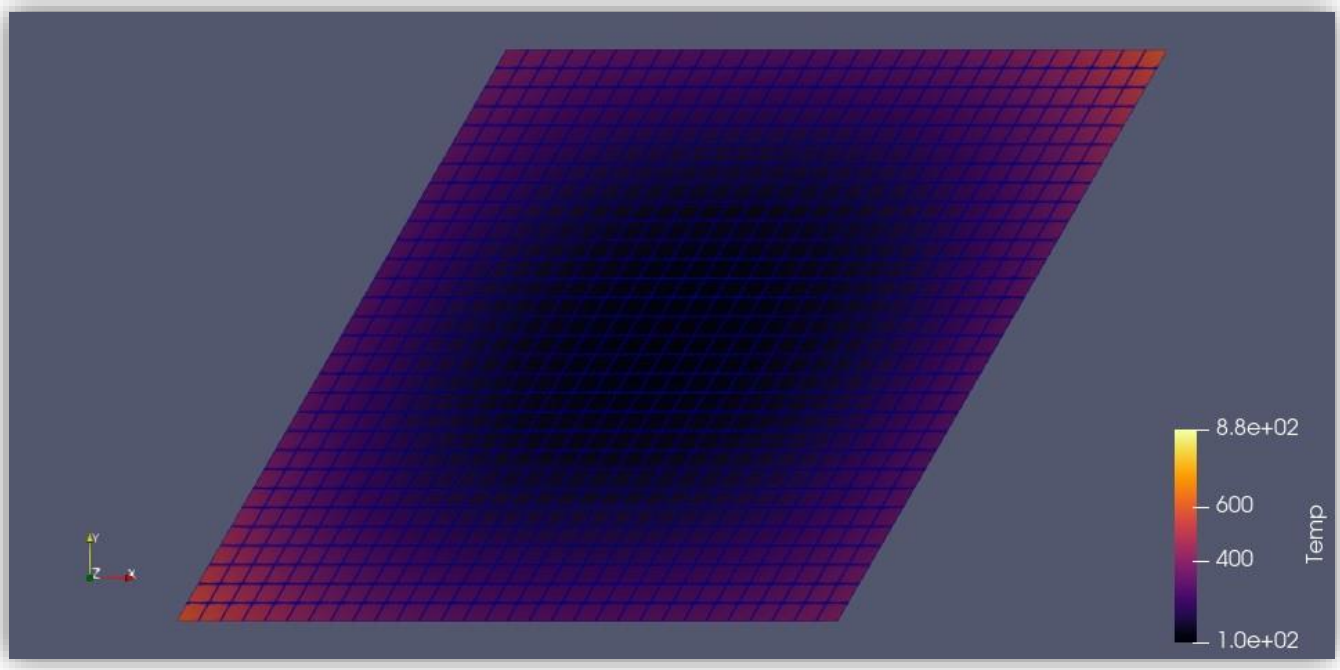
Siatka 4x4 mix – stan po 4 s



Siatka 31x31 – stan po 20 s



Siatka 31x31 trapez – stan po 60 s



6. Wnioski

Obliczenia wykorzystujące MES niewątpliwe wymagają dużej wiedzy teoretycznej z zakresu badanego zjawiska. Znajomość opisu matematycznego zjawisk fizycznych, warunków brzegowych itp. jest niezbędna do poprawnego przeprowadzenia symulacji. Mimo to sama metoda opierająca się na podziale badanego obiektu na małe elementy skończone i prowadzeniu obliczeń dla każdego z osobna powoduje, że MES jest przenośnym i szeroko stosowanym rozwiązaniem. Prócz rozwiązywania zadań opisujących procesy cieplne (jak miało to miejsce w omawianym projekcie) metodę elementów skończonych stosuje się w wielu innych problemach inżynierskich takich jak: analiza naprężeń czy elektromagnetyzm. Jak też łatwo zauważyć jest to świetny przykład podejścia „dziel i zwyciężaj", czyli jednej ze skuteczniejszych metod projektowania algorytmów w programowaniu.

Prócz błędów jakie możemy spowodować poprzez niepoprawne ustawienie parametrów lub przyjęcie niepasującego modelu istnieje również ryzyko wystąpienia błędów obliczeniowych związanych z niedokładnością wykorzystywanych metod numerycznych. W wielu przypadkach błędy obliczeń są na akceptowalnym poziomie szczególnie w przypadku opisów mniej skomplikowanych zjawisk. W naszym oprogramowaniu w łatwy sposób mogliśmy nieco zwiększyć dokładność obliczeń stosując schemat całkowania o większej ilości punktów, oczywiście kosztem zużycia większej ilości zasobów procesora i pamięci operacyjnej. Mówiąc natomiast o wydajności to w moim kodzie zauważyłem znaczny jej spadek w przypadku coraz większych siatek, czego powodem był zastosowany przeze mnie algorytm rozwiązywania układów równań, który zapewnia poprawność, nie jest jednak zbyt wydajny obliczeniowo. Samo wyliczanie macierzy i przygotowywanie układu równań wykonywało się w każdym przypadku poniżej 1s, główna część obliczeń stanowi jednak samo rozwiązywanie układu, szczególnie jeśli prowadzimy długą symulacje lub gdy krok czasowy jest bardzo niewielki. Z tego powodu warto podkreślić, że MES jest czasochłonnym procesem obliczeniowym szczególnie, gdy symulacja jest bardziej skomplikowana niż ta realizowana na zajęciach i bierze się od uwagę zmiany stanu materiału w trakcie jej trwania. Prawdopodobnie dobrym pomysłem w celu osiągniecia lepszych wyników, prócz zmiany algorytmu obliczania układów równań, byłaby próba zrównoleglenia oprogramowania.

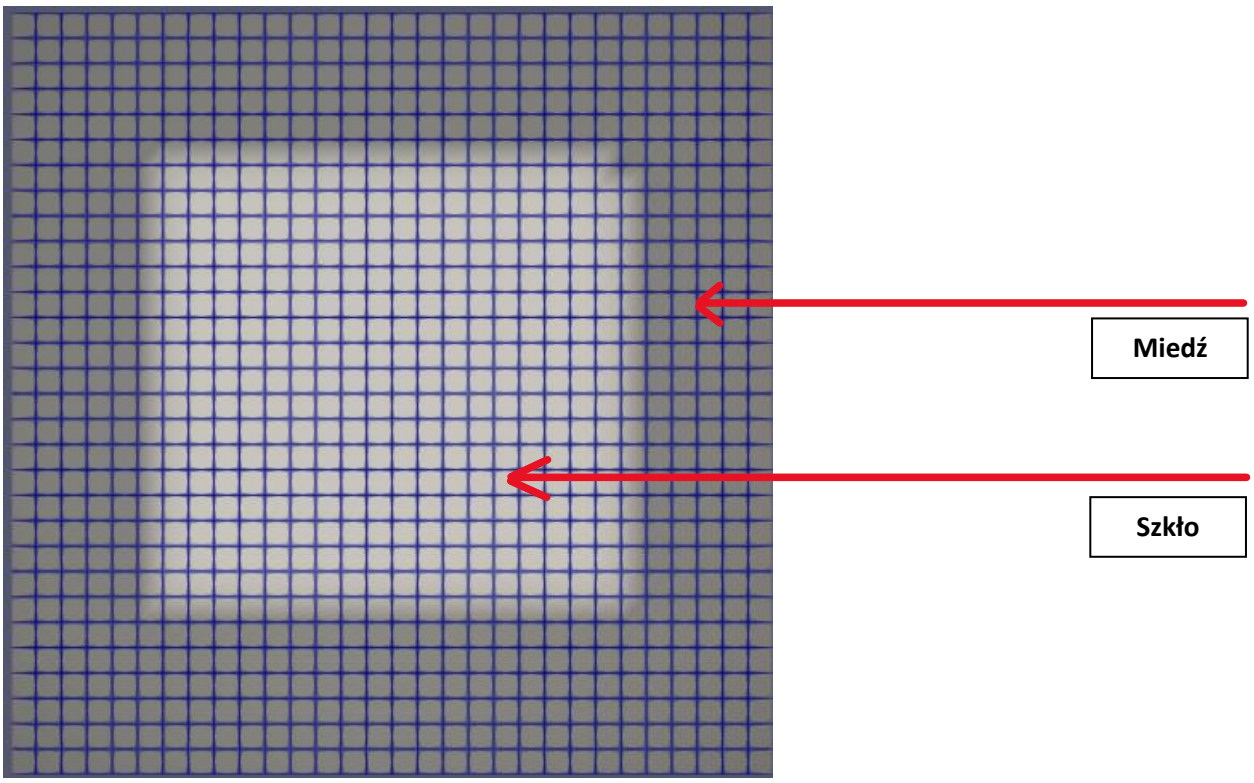
Jeśli chodzi o samo zadanie rozwiązywane w trakcie zajęć to udało się zasymulować proces nagrzewania materiału (prawdopodobnie był to jakiś metal) przy temperaturze otoczenia 1200°C i temperaturze początkowej obiektu 100°C. Podczas przeprowadzanego doświadczenia braliśmy pod uwagę konwekcyjny warunek brzegowy. Proces ten w praktyce mógłby przykładowo służyć zbadaniu po jakim czasie metal wsadzony do piec w celu obróbki plastycznej nagrzeje się do określonej temperatury.

7. Rozwiązanie własne

1. Opis problemu

Problem jaki rozwiązuje jest w zasadzie rozszerzeniem tego co zrealizowaliśmy w ramach zajęć i ma charakter raczej teoretycznych rozważań. Na bazie siatki 31x31 w kształcie kwadratu stworzyłem prosty model składający się z dwóch materiałów, a mianowicie izolatora i przewodnika – miedzi z zewnątrz oraz szkła wewnątrz. Budowę taką można spotkać na przykład w płytkach drukowanych

(PCB), które produkuje się z różnego rodzaju laminatów m.in. szklanych oraz pokrywa cienką warstwą miedzi. Mój model nie odzwierciedla dokładnie takiej płytki i tak jak wspominałem jest to bardziej teoretyczny przykład, który służy bardziej pokazaniu jak w łatwy sposób zmodyfikować oprogramowanie, żeby rozwiązać bardziej skomplikowane układy. Celem symulacji jest obserwacja zachowania izolatora/przewodnika w trakcie nagrzewania, dlatego wybrałem miedź, która jest świetnym przewodnikiem oraz szkło które dobrze sprawdza się jako izolator.



2. Charakterystyka danych materiałowych oraz źródła

Dane ogólne przeprowadzonej symulacji:

- Całkowity czas symulacji : 450 [s]
- Krok czasowy : 5 [s]
- Współczynnik wnikania ciepła : 300 [$W/m^2 \cdot$K]
- Temperatura otoczenia : 700 [°C]
- Temperatura początkowa materiału (całego) : 20 [°C]

Miedź

- Współczynnik przewodzenia ciepła : 370 [$W/m \cdot$K]
- Gęstość : 8960 [kg/m^3]
- Ciepło właściwe : 385 [$J/kg \cdot$K]

Szkło (wartości dla szkła używanego np. w oknach)

- Współczynnik przewodzenia ciepła : 0.8 [$W/m \cdot$K]
- Gęstość : 2500 [kg/m^3]
- Ciepło właściwe : 600 [$J/kg \cdot$K]

Źródła, z których zaczerpnięto informacje odnośnie własności szkła oraz miedzi.

https://www.naukowiec.org/tablice/fizyka/cieplo-wlasciwe_3311.html
https://izosystems.pl/content/16-wspolczynniki-lambda

3. Realizacja obliczeń

W celu obliczenia jak zachowuje się ciało anizotropowe należało w pierwszej kolejności zmodyfikować model w taki sposób żeby można było wziąć pod uwagę dwa różne materiały. Do tego zadanie zmodyfikowałem plik wejściowy dodając wiersz zawierający id elementów, które są jakimś innym materiałem (pozostałem elementy będą materiałem bazowym). Zrobiłem to w analogiczny sposób przy wprowadzaniu warunku brzegowego z tą różnicą, że korzystałem z id elementów a nie punktów.

```
*BC
31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18,
*ISOLATION
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,
```

Następnie w programie głównym określiłem właściwości wprowadzanego materiału:

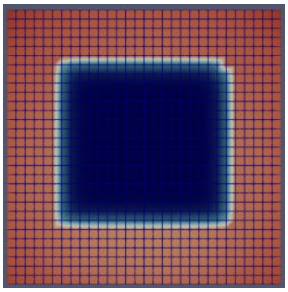
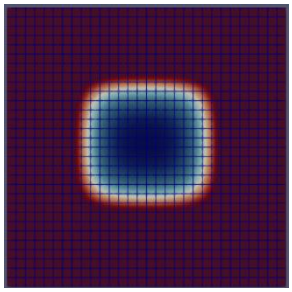
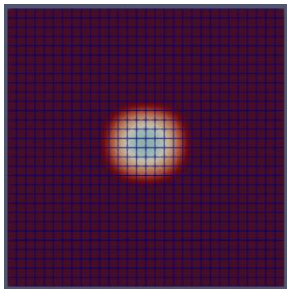
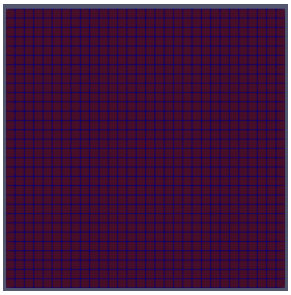
```
// szkło
elementData.Conductivity =  0.8;
elementData.Density = 2500;
elementData.SpecificHeat = 600;
```

Pozostało jeszcze policzyć macierz H, Hbc, C oraz wektor P oddzielnie dla każdego materiału korzystając z wczytanych flag (1 oznacza szkło, 0 oznacza miedź).

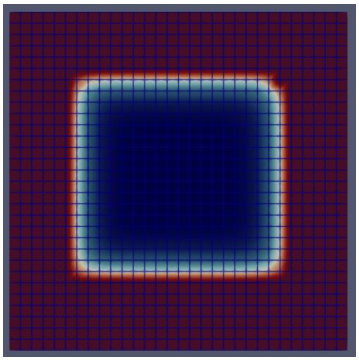
```
for (int i = 0; i < globalData.Elements_number; ++i){
    if(grid.elements[i].isolation == 0){
        uElement.calculateH_C(grid.elements[i], globalData);
        uElement.calculateHbc_P(grid.elements[i], globalData);
        grid.elements[i].calculateGlobalH(globalData);
    }else{
        uElement.calculateH_C(grid.elements[i], elementData);
        uElement.calculateHbc_P(grid.elements[i], elementData);
        grid.elements[i].calculateGlobalH(elementData);
    }
```

4. Interpretacja wyników

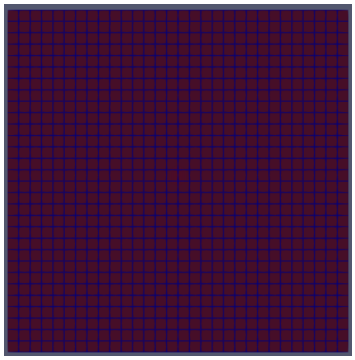
Przebieg symulacji dla odpowiednio 30s, 150 s, 300s i 420s:

Materiał zewnętrzny (miedź) nagrzał się do wartości temperatury otoczenia już po 50s natomiast materiał wewnętrzny potrzebował znacznie dłuższego nagrzewania i osiągnął 700°C dopiero po ok. 370s. Cały układ przeszedł w stan ustalony po 420 sekundach nagrzewania. Stosunek powierzchni szkła do miedzi wynosi 4:5 więc miedzi jest trochę więcej, wciąż jednak można zaobserwować siedmiokrotnie dłuższy czas nagrzewania się szkła.



50 s



420s

5. Wnioski

- Porównanie dwóch materiałów, z których jeden jest izolatorem a drugi przewodnikiem jest dosyć trywialnym zagadnieniem, symulacja pokazuje jednak jak zmodyfikować stworzone oprogramowanie MES, aby służyło w różnego rodzaju zagadnieniach.
- Prócz zmiany materiału z izotropowego na anizotropowy (można dodać oczywiście więcej rodzajów materiałów) można w dowolny sposób zmieniać kształt badanego ciała i tworzyć skomplikowane struktury. Utrudni się wówczas proces tworzenia samej siatki, natomiast sam program liczący macierze dla poszczególnych elementów może zostać niezmieniony.
- Niewątpliwym plusem tego podejścia jest to, że oprogramowanie, a przynajmniej jego część może być przenośna i służyć w rozwiązywaniu wielu problemów w zależności od danych wejściowych. Przedstawiony przykład pokazuje właśnie takie podejście - prócz zmiany danych początkowych oraz lekkiej modyfikacji programu głównego, klasy odpowiedzialne za obliczanie całek oraz układu równań pozostały takie same jak wcześniej.

Źródła:

- *Materiały z ćwiczeń laboratoryjnych*
- *http://ww1.metal.agh.edu.pl/~milenin/Dydaktyka/MES/MES2005a.pdf*
- *https://komes.pl/analiza-mes-zastosowanie/*
- *http://www.imio.polsl.pl/Dopobrania/MES.pdf*
- *https://gmsystem.pl/blog/na-czym-polega-symulacja-i-modelowanie-komputerowe-z-zastosowaniem-metody-elementow-skonczonych-mes/*
- *https://eduinf.waw.pl/inf/alg/001_search/0077.php*