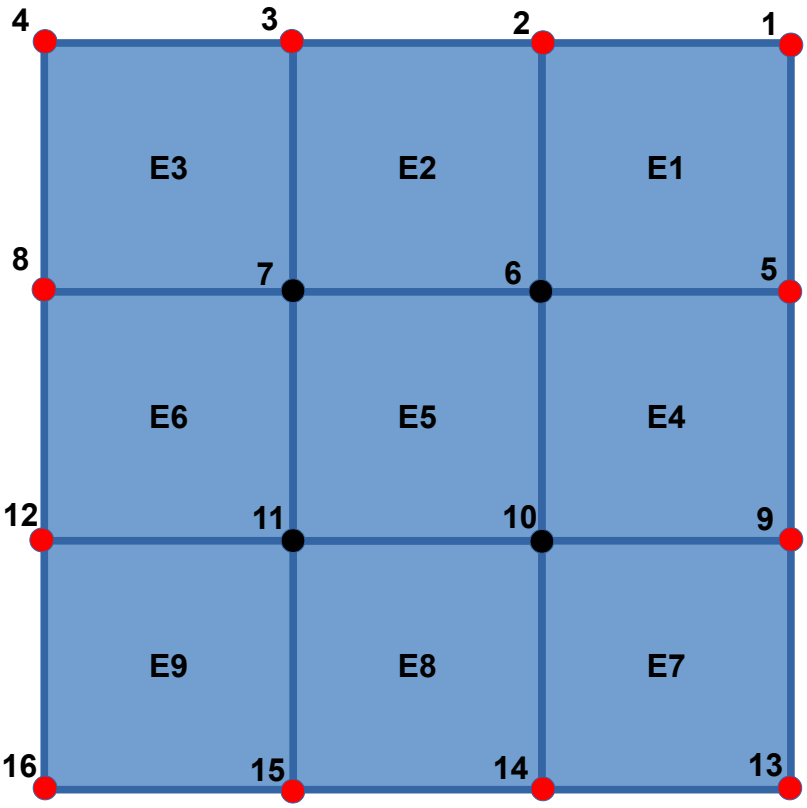
16.01.2024

Szymon Ociepka
Informatyka Techniczna
grupa 6
rok3

Szymon Ociepka

METODA ELEMENTÓW SKOŃCZONYCH



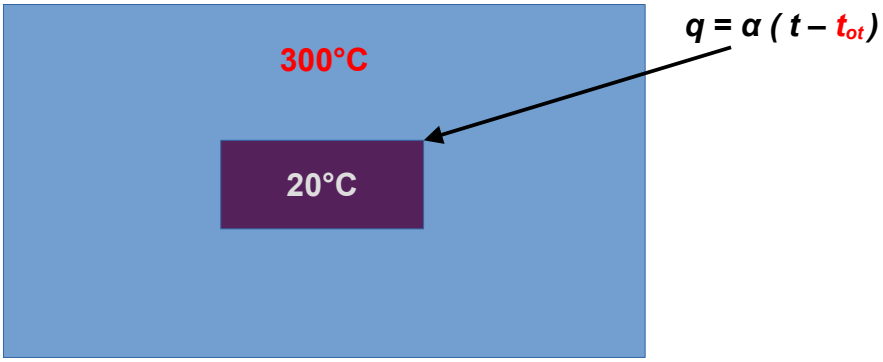
rysunek 1: siatka elementów skończonych dla 9 elementów

Metoda elementów skończonych (MES) polega na rozwiązywaniu równań różniczkowych opisujących dane zjawisko fizyczne w określonym obiekcie. W niniejszym przypadku analizowanym obiektem będzie płyta.

W ramach tej metody obiekt zostanie podzielony na dyskretne elementy skończone, co umożliwia przybliżenie rozwiązania równania różniczkowego na każdym z tych elementów. Następnie, zdefiniowane zostaną odpowiednie warunki brzegowe, które opisują zewnętrzne oddziaływania na analizowany układ, np. temperaturę na krawędziach płyty. Celem analizy będzie określenie, jak transport ciepła zachodzi w materiale płyty w zadanych warunkach.

Tworzona jest siatka elementów skończonych. W tym kroku należy podzielić obiekt na mniejsze elementy geometryczne, które stanowią podstawę obliczeń. W naszym przypadku tych elementów jest 9, przez co węzłów jest 16 (rysunek 1). Następnie rozwiązując równanie różniczkowe przy użyciu MES obliczenia zostaną przeprowadzone dla poszczególnych elementów, aby uzyskać lokalne wartości temperatury. Warunki brzegowe zostaną wprowadzone jako istotne dane wejściowe, które określą wpływ otoczenia na obiekt.

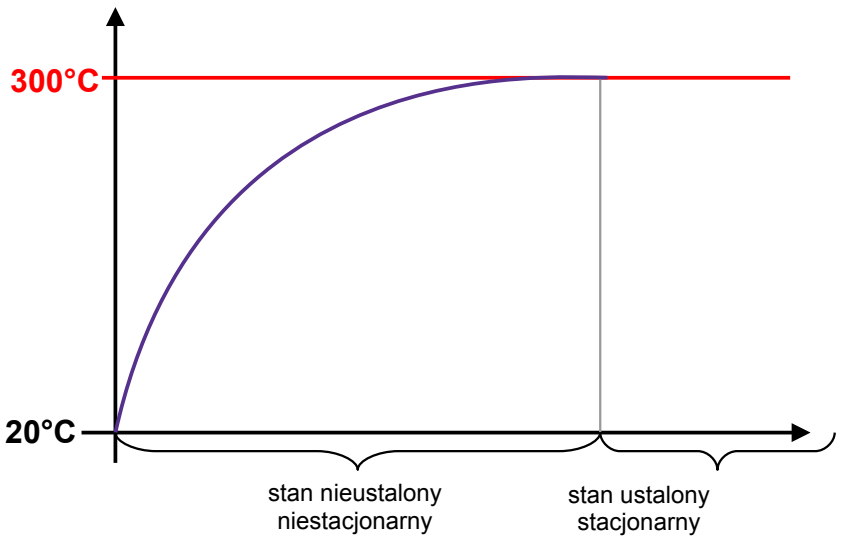
Analiza problemu



Rozważmy obiekt o pewnej temperaturze początkowej, który umieszczamy na przykład w piecu o wyższej temperaturze otoczenia. Proces nagrzewania obiektu jest wynikiem różnicy temperatur pomiędzy powierzchnią obiektu a temperaturą otoczenia. Różnica ta prowadzi do przepływu ciepła, opisanego jako gęstość strumienia ciepła *q*, która jest bezpośrednio zależna od różnicy temperatur oraz współczynnika *α*. Współczynnik α określa zdolność powierzchni obiektu do wymiany ciepła z otoczeniem – im wyższa wartość α, tym intensywniejszy transfer ciepła.

Po umieszczeniu obiektu w piecu i upływie czasu t, temperatura obiektu zaczyna wzrastać. Strumień ciepła q w chwili początkowej t_0 wynika z różnicy temperatur, na przykład $t_{ot}-t_0=280°C$, przy założeniu stałej wartości współczynnika α. W miarę upływu czasu, gdy temperatura powierzchni obiektu osiągnie np. 100°C°C, różnica temperatur zmniejszy się do $t_{ot}-t_0=200°C$, co prowadzi do zmniejszenia strumienia ciepła.

Strumień ciepła zmienia się dynamicznie w każdej chwili czasowej, ponieważ zależy od różnicy temperatur, która zmniejsza się wraz z ogrzewaniem się obiektu. Proces ten jest zatem nieliniowy i wymaga ciągłego monitorowania zmian parametrów w funkcji czasu, aby w pełni opisać dynamikę wymiany ciepła.



Im gęściej realizujemy iteracje na osi czasu (*oś x*), tym dokładniej zostanie odwzorowana zmiana temperatury obiektu oraz gęstość strumienia ciepła, która jest kluczowym czynnikiem wpływającym na tę zmianę. Taka szczegółowa analiza pozwala lepiej opisać dynamikę procesu nagrzewania obiektu.

Na wykresie zauważamy, że w momencie, gdy temperatura obiektu osiągnie wartość temperatury otoczenia (300°C), system wchodzi w stan ustalony (stacjonarny). W tym stanie temperatura obiektu przestaje się zmieniać w czasie, ponieważ różnica temperatur między obiektem a otoczeniem zanika, co skutkuje zerowym strumieniem ciepła.

W naszej analizie skupimy się na implementacji procesu niestacjonarnego (przedstawionego po lewej stronie wykresu), w którym temperatura obiektu dynamicznie wzrasta w czasie, od wartości początkowej do osiągnięcia stanu równowagi termicznej z otoczeniem.

Implementacja (c++)

1. Parametry NPC i npc

```
#define NPC 4  //4, 9, 16
```
Parametr NPC definiuje liczbę punktów całkowania w elemencie.

```
#define npc 2   //2, 3, 4
```
Parametr npc określa liczbę punktów całkowania na krawędziach elementu.

2. Struktury danych

*Załadowanie danych z plików tekstowych do struktury **GlobalData**, która przechowuje globalne parametry symulacji, gdzie:*

- *SimulationTime* – całkowity czas symulacji,
- *SimulationStepTime* – krok czasowy symulacji,
- *Conductivity* – przewodność cieplna materiału (k),
- *Alfa* – współczynnik wymiany ciepła z otoczeniem (α),
- *Tot* – temperatura otoczenia,
- *InitialTemp* – temperatura początkowa obiektu,
- *Density* – gęstość materiału (ρ),
- *SpecificHeat* – ciepło właściwe (c),
- *NodesNumber* – liczba węzłów w siatce,
- *ElementsNumber* – liczba elementów w siatce.

Node

Reprezentuje węzeł siatki elementów skończonych:

```
struct Node {
    double x, y; //współrzędne węzła w układzie globalnym
    bool BC;      //flaga określająca, czy węzeł spełnia warunek brzegowy (true/false)
};
```

Jakobian

Przechowuje dane dotyczące transformacji współrzędnych w elemencie:

```
struct Jakobian {
    double J[2][2];    // Macierz Jakobiego
    double J1[2][2];  // Odwrócona Macierz Jakobiego
    double detJ;       // Wyznacznik Jakobianu
};
```

ElemUniv

Uniwersalne dane dla elementu, związane z całkowaniem numerycznym:

```
struct ElemUniv {
    double Ne[NPC][4];// dN/dksi dla każdego punktu całkowania i każdej funkcji kształtu
    double Nn[NPC][4];// dN/deta dla każdego punktu całkowania i każdej funkcji kształtu
    double Nc[NPC][4];// wartości funkcji kształtu, potrzebne do obliczeń macierzy C
    double multiplied_weights[NPC];      // pomnożone wagi
    double weights[npc];                 // wagi
    double NPC_gaussPoints[NPC][2];  // Punkty Gaussa dla każdego punktu całkowania NPC
    double npc_gaussPoints[npc * 4][2];// Punkty Gaussa dla każdego punktu całkowania npc krawędzi elementu

    struct Surface {
        double N[npc][4]; // Macierz funkcji kształtu dla 4 ścianek, każda ma 4 funkcje kształtu
    } surface[4]; // 4 powierzchnie dla każdej ściany
};
```

Element

Reprezentuje pojedynczy element siatki

```
struct Element {
    int ID[4] = {};            // identyfikatory węzłów elementu
    Jakobian Jacobian[NPC] = {};// dane Jakobiego dla każdego punktu całkowania
    double dN_dx[NPC][4] = {};  // pochodne funkcji kształtu względem x
    double dN_dy[NPC][4] = {};  // pochodne funkcji kształtu względem y
    double H[4][4] = {};        // macierz H
    double C[4][4] = {};        // macierz C
    double P[4] = {};           // wektor P
};
```

Grid

Siatka elementów skończonych:

- nodes – tablica węzłów siatki,
- elements – tablica elementów siatki.

Relacje między strukturami

- Grid zawiera wszystkie węzły i elementy siatki.

- Każdy element (Element) odwołuje się do węzłów poprzez identyfikatory (ID) oraz używa danych z ElemUniv (punktów całkowania i funkcji kształtu).
- Dane globalne (GlobalData) są używane do obliczeń w całej siatce, takich jak krok czasowy, przewodność cieplna i inne parametry materiałowe.

Liczenie Jakobianu

$$\begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} \quad (1)$$

(A) Macierz Jakobiego (B)

Jakobian – twierdzenie matematyczne o pochodnej funkcji złożonej, dzięki któremu jesteśmy w stanie powiązać układ ξ i η (A) z układem x i y (B). Innymi słowy jakobian przekształcenia łączy te dwa układy.

Wyznacznik macierzy jakobiego – mówi o stosunku pól naszego elementu skończonego do elementu uniwersalnego (o wymiarach 2x2)

Zaimplementowane liczenie jakobianu znajduję się w funkcji *ObliczJakobian()*. Obliczenia zaczynamy od pochodnych funkcji kształtu względem ξ (Ne) oraz η (Nn) w funkcji `ObliczPochodneFunkcjiKsztaltu`, gdzie ip = 0; ip < NPC.

```
for (int ip = 0; ip < NPC; ++ip) {
    double xi = elemUniv.NPC_gaussPoints[ip][0];
    double eta = elemUniv.NPC_gaussPoints[ip][1];

    elemUniv.Ne[ip][0] = -0.25 * (1 - eta);
    elemUniv.Ne[ip][1] = 0.25 * (1 - eta);
    elemUniv.Ne[ip][2] = 0.25 * (1 + eta);
    elemUniv.Ne[ip][3] = -0.25 * (1 + eta);

    elemUniv.Nn[ip][0] = -0.25 * (1 - xi);
    elemUniv.Nn[ip][1] = -0.25 * (1 + xi);
    elemUniv.Nn[ip][2] = 0.25 * (1 + xi);
    elemUniv.Nn[ip][3] = 0.25 * (1 - xi);
}
```

Mając te dane wyznaczamy składowe macierzy jakobiego sumując iloczyny odpowiednich funkcji kształtu ze składowymi x oraz y wierzchołków dla danego elementu.

```
for (int k = 0; k < 4; ++k) {//wyznaczamy jakobian przekształcenia
        int nodeIndex = element.ID[k] - 1; // -1 zeby indeks był od zera
        Node& node = grid.nodes[nodeIndex];

        dXdXi += elemUniv.Ne[ip][k] * node.x;
        dXdEta += elemUniv.Nn[ip][k] * node.x;
        dYdXi += elemUniv.Ne[ip][k] * node.y;
        dYdEta += elemUniv.Nn[ip][k] * node.y;
    }
```

Po wyliczeniu tych składowych otrzymujemy jakobian dla danego punktu całkowania. Wyliczone wartości przypisujemy do danego Jakobianu.

```
//otrzymany jakobian dla danego punktu całkowania
element.Jacobian[ip].J[0][0] = dXdXi;
element.Jacobian[ip].J[0][1] = dXdEta;
element.Jacobian[ip].J[1][0] = dYdXi;
element.Jacobian[ip].J[1][1] = dYdEta;
```

Liczymy wyznacznik jakobiego z tej macierzy.

```
//licze wyznaczik detJ
element.Jacobian[ip].detJ = dXdXi * dYdEta - dXdEta * dYdXi;
```

Odwracamy obliczony wyznacznik jakobiego.

```
double invDetJ = 1.0 / element.Jacobian[ip].detJ; //odwrócony wyznacznik jakobiego
```

Następnie tworzymy odwrócony jakobian.

```
//odwracam Jakobian
element.Jacobian[ip].J1[0][0] = dYdEta * invDetJ;
element.Jacobian[ip].J1[0][1] = -dXdEta * invDetJ;
element.Jacobian[ip].J1[1][0] = -dYdXi * invDetJ;
element.Jacobian[ip].J1[1][1] = dXdXi * invDetJ;
```

Ten wynik widoczny powyżej jest informacją w jaki sposób układ x, y łączy się z układem ξ, η. Powtarzamy ten proces tyle razy ile jest punktów całkowania NPC.

Liczenie $\frac{\partial N}{\partial x}$ oraz $\frac{\partial N}{\partial y}$ wykorzystując odwróconą macierz Jakobiego

Mnożymy odpowiednie wiersze przez odpowiednie kolumny:

```
for (int i = 0; i < 4; ++i) {// wyliczanie pochodnych dN/dx oraz dN/dy
        element.dN_dx[ip][i] = element.Jacobian[ip].J1[0][0] * elemUniv.Ne[ip][i] +
                               element.Jacobian[ip].J1[1][0] * elemUniv.Nn[ip][i];

        element.dN_dy[ip][i] = element.Jacobian[ip].J1[0][1] * elemUniv.Ne[ip][i] +
                               element.Jacobian[ip].J1[1][1] * elemUniv.Nn[ip][i];
}
```

Oblicanie macierzy H dla punktu całkowania

$$[H]=\int_V k(t)\left(\left\{\frac{\partial\{N\}}{\partial x}\right\}\left\{\frac{\partial\{N\}}{\partial x}\right\}^T+\left\{\frac{\partial\{N\}}{\partial y}\right\}\left\{\frac{\partial\{N\}}{\partial y}\right\}^T\right)dV \quad (2)$$

lokalna macierz przewodzenia ciepła [H]

k – przewodność (conductivity)
dV – wyznacznik Jakobiego (element.Jacobian[ip].detJ)

```
//wartosci iloczynów pochodnych dN/dx oraz dN/dy w danych współrzędnych
double dNdx_i_dNdx_j = element.dN_dx[ip][i] * element.dN_dx[ip][j];
double dNdy_i_dNdy_j = element.dN_dy[ip][i] * element.dN_dy[ip][j];

//Obliczam H lokalne ze wzoru i dodaje do finalnej macierzy H
element.H[i][j] += conductivity * (dNdx_i_dNdx_j + dNdy_i_dNdy_j) * element.Jacobian[ip].detJ *
elemUniv.multiplied_weights[ip];
```

Liczenie fragmentu macierzy H - macierzy Hbc

W tym fragmencie do naszej H lokalnej obliczonej w poprzednim kroku będzie dodawany warunek brzegowy.

$$[H]+=\int_S \alpha\{N\}\{N\}^T dS \quad (3)$$

dodawany warunek brzegowy

Wartości funkcji kształtu w odpowiednich punktach npc ścian bocznych elementów wyglądają następująco:

```
for (int s = 0; s < 4; ++s) { // 4 powierzchnie
   for (int ip = 0; ip < npc; ++ip) {//obliczamy pochodne funkcji kształtu względem ksi (Ne) oraz eta (Nn)
      double xi = elemUniv.npc_gaussPoints[npc * s + ip][0];
      double eta = elemUniv.npc_gaussPoints[npc * s + ip][1];

      elemUniv.surface[s].N[ip][0] = 0.25 * (1 - xi) * (1 - eta);   // N1
      elemUniv.surface[s].N[ip][1] = 0.25 * (1 + xi) * (1 - eta);  // N2
      elemUniv.surface[s].N[ip][2] = 0.25 * (1 + xi) * (1 + eta); // N3
      elemUniv.surface[s].N[ip][3] = 0.25 * (1 - xi) * (1 + eta);  // N4
   }
}
```

Przechodzimy do funkcji *ObliczBC()*. Najpierw sprawdzamy czy dane dwa węzły mają warunek brzegowy = true, aby uniknąć liczenia macierzy Hbc która takiego warunku nie posiada = false:

```
        if (grid.nodes[node1].BC && grid.nodes[node2].BC) { // Sprawdzenie BC
```

Obliczany jest wyznacznik detJ naszej krawędzi z twierdzenia Pitagorasa:

```
        double dx = grid.nodes[node2].x - grid.nodes[node1].x;
        double dy = grid.nodes[node2].y - grid.nodes[node1].y;
```

```
double detJ_surface = sqrt(dx * dx + dy * dy) / 2.0; // Długość krawędzi / 2 - dł. znormalizowanego układu
```

Obliczenie macierzy Hbc (ze wzoru 3) oraz dodanie jej do macierzy H:

```
for (int ip = 0; ip < npc; ip++) {
    // Wartości funkcji kształtu na krawędzi
    double N[4] = {
        elemUniv.surface[s].N[ip][0],
        elemUniv.surface[s].N[ip][1],
        elemUniv.surface[s].N[ip][2],
        elemUniv.surface[s].N[ip][3]
        };

    //Iteracja po funkcjach kształtu (lokalne węzły)
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 4; ++j) {
            element.H[i][j] += alfa * N[i] * N[j] * detJ_surface * elemUniv.weights[ip];
        }
        element.P[i] += alfa * N[i] * tot * detJ_surface * elemUniv.weights[ip];
    }
```

Agregacja do macierzy globalnych

Agregacja w analizie metodą elementów skończonych (MES) polega na sumowaniu macierzy i wektorów lokalnych (obliczonych dla poszczególnych elementów skończonych) do macierzy i wektorów globalnych (odnoszących się do całej siatki).

Funkcja *AgregujDoMacierzyGlobalnej()* łączy wyniki z lokalnych macierzy H, C i wektorów P w jeden spójny globalny system równań. Celem agregacji jest połączenie informacji z lokalnych macierzy elementów skończonych w jeden system równań liniowych dla całego modelu.

Przykładowy fragment kodu agregacji dla macierzy H:

```
// Agregacja lokalnej macierzy H do macierzy globalnej HG
  for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        int global_i = element.ID[i] - 1;
        int global_j = element.ID[j] - 1;

        HG[global_i][global_j] += element.H[i][j];
    }
  }
```

Liczenie macierzy C

$$[C] \mathrel{+}= \int_V \rho \, c_p \left(\{N\} \{N\}^T \right) dV \qquad (4)$$

lokalna macierz pojemności cieplnej

ρ – gęstość (*Density*)
c_p – ciepło właściwe (*SpecificHeat*)
dV – wyznacznik Jakobiego (element.Jacobian[ip].detJ)

Obliczenia zaczynamy od żywych funkcji kształtu względem ξ (Ne) oraz η (Nn) w funkcji ObliczPochodneFunkcjiKsztaltu, gdzie ip = 0; ip < NPC.

```
for (int ip = 0; ip < NPC; ++ip) {
    double xi = elemUniv.NPC_gaussPoints[ip][0];
    double eta = elemUniv.NPC_gaussPoints[ip][1];

    elemUniv.Nc[ip][0] = 0.25 * (1 - xi) * (1 - eta);
    elemUniv.Nc[ip][1] = 0.25 * (1 + xi) * (1 - eta);
    elemUniv.Nc[ip][2] = 0.25 * (1 + xi) * (1 + eta);
    elemUniv.Nc[ip][3] = 0.25 * (1 - xi) * (1 + eta);
}
```

Następnie wracając do funkcji *ObliczJakobian()* obliczamy macierzy C ze wzoru (4):

```
//OBLICZANIE C
double Hclocal[4][4] = {};  // Temporary Hc matrix for this integration point

for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        element.C[i][j] += Density * SpecificHeat * elemUniv.Nc[ip][i] * elemUniv.Nc[ip][j] *
                           element.Jacobian[ip].detJ * elemUniv.multiplied_weights[ip];
    }
}
```

Liczenie macierzy C

$$\left([H]+\frac{[C]}{\Delta\tau}\right)\{t_1\}-\left(\frac{[C]}{\Delta\tau}\right)\{t_0\}+\{P\}=0 \qquad (5)$$

równanie dynamiczne przewodnictwa cieplnego

gdzie:

$\Delta\tau = \frac{SimulationTime}{SimulationStepTime}$,

$t_0 = InitialTemp$.

Przedstawiony wzór opisuje równanie dynamiczne przewodnictwa cieplnego. Jest to kluczowe równanie stosowane w analizie nieustalonego (niestacjonarnego) przepływu ciepła, które pozwala na obliczenie wektora temperatury w węzłach ($\{t_1\}$) w kolejnych krokach czasowych $\Delta\tau$. Posiadając H, C, P globalne, początkowe wartości temperatury t_0 jak i $\Delta\tau$ jesteśmy w stanie tę temperaturę policzyć, wykorzystując ten wzór (5).

Metoda Eliminacji Gaussa

$$Ax = b \tag{6}$$

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad x=\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad b=\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

macierz n x n **szukane rozwiązanie** **wektor danych**

gdzie:

A = $\left([H]+\frac{[C]}{\Delta\tau}\right)$,

x = $\{t_1\}$,

b = $\left(\frac{[C]}{\Delta\tau}\right)\{t_0\}+\{P\}$.

Do obliczenia t_1 również stosuję metodę eliminacji Gaussa w funkcji *gaussTemperatura()*. Metoda eliminacji Gaussa jest jednym z podstawowych algorytmów stosowanych do rozwiązywania układów równań liniowych. Jest to metoda bezpośrednia, co oznacza, że po skończonej liczbie kroków dostajemy dokładne rozwiązanie (w granicach błędu numerycznego). Głównym celem metody eliminacji Gaussa jest przekształcenie pierwotnego układu równań do postaci, w której łatwo można znaleźć rozwiązanie. Dokonuje się tego poprzez eliminację niewiadomych - stąd nazwa metody.

Fragmenty funkcji *gaussTemperatura():*

Sprowadzanie macierzy do postaci trójkątnej górnej:

```
for (int k = 0; k < NodesNumber; k++) {
   for (int i = k + 1; i < NodesNumber; i++) {
      for (int j = NodesNumber; j >= 0; j--) {
         HGreversed[i][j] -= HGreversed[k][j] * (HGreversed[i][k] / HGreversed[k][k]);
      }
   }
}
```

Postępowanie wsteczne które pozwala wyznaczyć wartości t_1, t_2, …, t_n dla kolejnych węzłów:

```
for (int i = NodesNumber - 1; i >= 0; i--) {
   t1[i] = HGreversed[i][NodesNumber];
   for (int j = i + 1; j < NodesNumber; j++) {
      t1[i] -= HGreversed[i][j] * t1[j];
   }
   t1[i] /= HGreversed[i][i];
}
```

Wyniki

Test1_4_4

Tabele 1-2 przedstawiają wyniki działania zaimplementowanego programu dla pliku *Test1_4_4.txt* oraz wyniki testowe pochodzące z prezentacji UPEL. W Tabeli 1 zaprezentowano minimalne temperatury obiektu w poszczególnych chwilach czasowych, natomiast Tabela 2 zawiera wartości maksymalne temperatury obiektu w tych samych momentach.

Siatka 4x4 | min. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
50	110.0379723556	110.0379723554	110.0625459086	110.03797659406167
100	168.8370097663	168.8370097658	169.0081558855	168.83701715655656
150	242.8008462722	242.8008462714	243.1396607138	242.80085524391868
200	318.6145887045	318.6145887033	319.1078339333	318.61459376004086
250	391.2557917895	391.2557917880	391.8805194951	391.2557916738893
300	459.0369089191	459.0369089173	459.7691537189	459.03690325635404
350	521.5862853957	521.5862853936	522.4038945342	521.5862742337766
400	579.0344613924	579.0344613902	579.9178325761	579.0344449687701
450	631.6892582330	631.6892582307	632.6213639759	631.6892368621455
500	679.9076191304	679.9076191280	680.8738089489	679.9075931513394

*Tabela 1: wyniki **najniższej** temperatury obiektu w danej chwili dla Siatki **4x4** dla trzech schematów całkowania **npc = 2, 3, 4***

Siatka 4x4 | max. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
50	365.8154726251	365.8154726237	366.4678366871	365.8154705784631
100	502.5917142786	502.5917142766	503.4934818278	502.5917120896439
150	587.3726667097	587.3726667073	588.3806680032	587.372666691486
200	649.3874821805	649.3874821780	650.4454123609	649.3874834542602
250	700.0684182945	700.0684182919	701.1500937282	700.0684204214381
300	744.0633414735	744.0633414709	745.1537299307	744.0633443187048
350	783.3828462176	783.3828462150	784.4716117857	783.382849723737
400	818.9921835720	818.9921835694	820.0714232218	818.9921876836681
450	851.4310377964	851.4310377938	852.4944294184	851.4310425916341
500	881.0576293886	881.0576293861	882.1000579631	881.057634906017

*Tabela 2: wyniki **najwyższej** temperatury obiektu w danej chwili dla Siatki **4x4** dla trzech schematów całkowania **npc = 2, 3, 4***

Test2_4_4_MixGrid

Tabele 3-4 przedstawiają wyniki działania zaimplementowanego programu dla pliku *Test2_4_4_MixGrid.txt* oraz wyniki testowe pochodzące z prezentacji UPEL. W Tabeli 3 zaprezentowano minimalne temperatury obiektu w poszczególnych chwilach czasowych, natomiast Tabela 4 zawiera wartości maksymalne temperatury obiektu w tych samych momentach.

Siatka MixGrid4x4 | min. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
50	95.1518489973	95.1590503646	95.1445644547	95.15184673458245
100	147.6444190737	147.6558659016	147.7767520427	147.64441665454345
150	220.1644557510	220.1780763727	220.4690449679	220.1644549730314
200	296.7364383530	296.7508284743	297.2035229031	296.7364399006366
250	370.9682724189	370.9825963129	371.5750186947	370.968275802604
300	440.5601435998	440.5739685564	441.2816278998	440.5601440058566
350	504.8912021091	504.9043332480	505.7041117005	504.8911996551285
400	564.0015163225	564.0138842014	564.8851644994	564.0015111915015
450	618.1738632501	618.1854601638	619.1102788787	618.1738556427995
500	667.7655569118	667.7764039681	668.7392858664	667.7655470268747

*Tabela 3: wyniki **najniższej** temperatury obiektu w danej chwili dla Siatki **MixGrid4x4** dla trzech schematów całkowania **npc = 2, 3, 4***

Siatka MixGrid4x4 | max. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
50	374.6863331884	374.6683438950	375.3370167446	374.6863325385064
100	505.9681112789	505.9543142752	506.8566194206	505.96811082245307
150	586.9978492792	586.9894526095	587.9907364963	586.9978503916302
200	647.2855822357	647.2801311263	648.3298817462	647.28558387732
250	697.3339844638	697.3298790593	698.4041884460	697.3339863103786
300	741.2191101273	741.2156580841	742.3001496751	741.2191121514377
350	781.2095685594	781.2407682508	782.3413894911	781.209569726045
400	817.3915046241	817.4204296924	818.5112627305	817.3915065469778
450	850.2373167651	850.2640353728	851.3385441462	850.2373194670416
500	880.1676019293	880.1922333881	881.2452048909	880.1676054000437

*Tabela 4: wyniki **najwyższej** temperatury obiektu w danej chwili dla Siatki **MixGrid4x4** dla trzech schematów całkowania **npc = 2, 3, 4***

Test3_31_31_kwadrat

Tabele 5-6 przedstawiają wyniki działania zaimplementowanego programu dla pliku *Test3_31_31_kwadrat.txt* oraz wyniki testowe pochodzące z prezentacji UPEL. W Tabeli 5 zaprezentowano minimalne temperatury obiektu w poszczególnych chwilach czasowych, natomiast Tabela 6 zawiera wartości maksymalne temperatury obiektu w tych samych momentach.

Siatka kwadrat31x31 | min. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
1	100.0000000003	100.0000000003	100.0000000003	99.99969812978378
2	100.0000000053	100.0000000053	100.0000000053	100.00053467957446
3	100.0000000515	100.0000000515	100.0000000516	100.00084733335379
4	100.0000003344	100.0000003344	100.0000003354	100.00116712763896
5	100.0000016382	100.0000016382	100.0000016431	100.00150209858216
6	100.0000064712	100.0000064712	100.0000064905	100.001852708951
7	100.0000215294	100.0000215294	100.0000215936	100.00222410506852
8	100.0000622127	100.0000622127	100.0000623982	100.00263047992797
9	100.0001597834	100.0001597834	100.0001602593	100.00310216686808
10	100.0003713449	100.0003713449	100.0003724497	100.00369558647527
11	100.0007922355	100.0007922355	100.0007945898	100.00450560745507
12	100.0015698461	100.0015698461	100.0015745060	100.00567932588369
13	100.0029174838	100.0029174838	100.0029261338	100.00742988613344
14	100.0051267975	100.0051267975	100.0051419797	100.01004886564658
15	100.0085774636	100.0085774636	100.0086028334	100.01391592562979
16	100.0137432142	100.0137432142	100.0137838126	100.01950481085419
17	100.0211937597	100.0211937597	100.0212562892	100.02738525124852
18	100.0315926141	100.0315926141	100.0316857062	100.0382207726261
19	100.0456912010	100.0456912010	100.0458256649	100.05276279329537
20	100.0643198699	100.0643198699	100.0645089128	100.07184163487159

*Tabela 5: wyniki **najniższej** temperatury obiektu w danej chwili dla Siatki **kwadrat31x31** dla trzech schematów całkowania **npc = 2, 3, 4***

Siatka kwadrat31x31 | max. temperatura (°C)				
czas(s)	***npc = 2***	***npc = 3***	***npc = 4***	***wyniki testowe***
1	149.5569518081	149.5569518078	149.7035206728	149.5566275788947
2	177.4449279501	177.4449279495	177.6708210706	177.44482649738018
3	197.2669629217	197.2669629210	197.5474295603	197.2672291500534
4	213.1527872915	213.1527872907	213.4758566029	213.15348263983788
5	226.6825834191	226.6825834182	227.0411399549	226.6837398631218
6	238.6070648059	238.6070648049	238.9962887391	238.60869878203812
7	249.3466919425	249.3466919415	249.7630448603	249.34880985057373
8	259.1650791551	259.1650791541	259.6058238810	259.1676797521773
9	268.2406890050	268.2406890039	268.7036294657	268.24376548847937
10	276.7010978633	276.7010978621	277.1844239018	276.70463950306436
11	284.6412831887	284.6412831874	285.1434713015	284.64527660833346
12	292.1342190509	292.1342190496	292.6539654879	292.1386492100023
13	299.2374099453	299.2374099440	299.7735837426	299.242260871447
14	305.9971215275	305.9971215262	306.5487304339	306.00237684844643
15	312.4512302135	312.4512302121	313.0173952920	312.4568735346492
16	318.6312061364	318.6312061350	319.2111423962	318.637221302136
17	324.5635314899	324.5635314885	325.1565328268	324.56990275925733
18	330.2707391734	330.2707391719	330.8761664630	330.27745133351596
19	335.7721890480	335.7721890465	336.3894605917	335.77922748329735
20	341.0846585343	341.0846585328	341.7132422946	341.0920092636545

Tabela 6: wyniki ***najwyższej*** *temperatury obiektu w danej chwili dla Siatki* ***kwadrat31x31*** dla *trzech schematów całkowania* ***npc = 2, 3, 4***

Wnioski

1. **Dokładność metody:**
   - Wyniki przedstawione w tabelach dla różnych schematów całkowania (npc=2,3,4) są bardzo zbliżone do wyników testowych z prezentacji UPEL, co świadczy o poprawnej implementacji metody.
   - Nawet dla różnych siatek (w tym dla plików *MixGrid* oraz *kwadrat31x31*), metoda zachowuje wysoką dokładność, co potwierdza jej uniwersalność w modelowaniu zjawisk cieplnych.
2. **Zbieżność wyników:**
   - Dla większej liczby punktów całkowania (npc=4) wyniki wykazują większą zgodność z wynikami testowymi, co jest zgodne z oczekiwaniami teoretycznymi. Większa liczba punktów całkowania zapewnia lepszą aproksymację wartości w integralnych macierzach lokalnych.
   - Różnice między wynikami dla npc=2,3 a wynikami testowymi są niewielkie, co wskazuje na skuteczność metody nawet przy mniejszej liczbie punktów całkowania.

3. **Stabilność metody:**
    - Metoda zachowuje stabilność niezależnie od rozmiaru siatki (od małych siatek, takich jak 4x4, do dużych, jak *kwadrat31x31*). Oznacza to, że implementacja jest dobrze przystosowana zarówno do analizy drobno, jak i gruboziarnistej.
4. **Efektywność obliczeniowa:**
    - Przy zwiększeniu liczby punktów całkowania npc, choć wyniki są dokładniejsze, wzrasta również czas obliczeń ze względu na bardziej skomplikowane obliczenia w macierzach lokalnych.
    - Implementacja jest zoptymalizowana do pracy z różnymi schematami całkowania, co pozwala użytkownikowi na elastyczne dostosowanie poziomu dokładności do wymagań obliczeniowych.
5. **Poprawność implementacji warunków brzegowych:**
    - Wyniki minimalnych temperatur oraz maksymalnych temperatur wskazują, że warunki brzegowe zostały zaimplementowane prawidłowo. Wartości brzegowe oraz ich wpływ na pole temperatury w obiekcie są zgodne z wynikami testowymi.
6. **Uniwersalność metody:**
    - Poprawne wyniki dla różnych siatek (*MixGrid* i *kwadrat31x31*) dowodzą, że metoda może być stosowana w różnych przypadkach geometrycznych oraz w siatkach o różnym stopniu złożoności.

Podsumowanie

Metoda elementów skończonych została poprawnie zaimplementowana, co potwierdzają wyniki dla plików testowych oraz dla różnych schematów całkowania i siatek. Implementacja zapewnia zarówno wysoką dokładność, jak i stabilność obliczeń w modelowaniu procesów cieplnych. Drobne różnice w wynikach dla różnych wartości npc potwierdzają wpływ wyboru schematu całkowania na dokładność wyników, jednocześnie wskazując, że nawet prostsze schematy (npc=2) mogą być wystarczające dla wielu przypadków praktycznych.