2023

Sprawozdanie z realizacji projektu na zaliczenie przedmiotu MES

KACPER PAPUGA

Spis treści

Wstęp teoretyczny

Metoda Elementów Skończonych to metoda numeryczna, pozwalająca na przybliżone rozwiązanie skomplikowanych problemów inżynieryjnych z zakresu m.in. mechaniki konstrukcji, mechaniki płynów czy wymiany ciepła, poprzez podział badanego obiektu na mniejsze elementy zwane elementami skończonymi. Główna idea MES polega na tym, że dowolną ciągłą wartość można zamienić na model dyskretny oparty na ograniczonej ilości węzłów, tworzących ograniczoną ilość elementów skończonych. Taki podział umożliwia zamianę stosowania metod analitycznych, które dla złożonych problemów symulacyjnych wymagają zazwyczaj dużego uproszczenia oraz wielu założeń, na metody numeryczne. Zaletą takiego podejścia jest możliwość modelowania zjawisk zachodzących w ośrodkach o skomplikowanych kształtach, które mogą być aproksymowane z dużą dokładnością za pomocą elementów skończonych. Ponadto metoda MES może być stosowana do modelowania własności materiałów wielofazowych, których poszczególne elementy mogą być objętościowo różne (różne elementy wchodzące w skład siatki), a także uwzględniać różne warunki brzegowe.

Opis modelu z warunkami brzegowymi

W ramach implementacji programu będziemy rozpatrywać symulację niestacjonarnej wymiany ciepła, dla problemu dwuwymiarowego.

Podstawowym równaniem opisującym zjawiska cieplne jest równanie Fourier'a. Dla procesu niestacjonarnego przyjmuje ono postać:

$$\frac{\partial}{\partial x}\left(k_x(t)\frac{\partial t}{\partial x}\right)+\frac{\partial}{\partial y}\left(k_y(t)\frac{\partial t}{\partial y}\right)+(Q-c\rho\frac{\partial t}{\partial T})=0$$

$k_x(t), k_x(t), k_x(t)$ – anizotropowe współczynniki przewodzenia ciepła zależne od temperatury t
Q – generowane ciepło
c – pojemność cieplna materiału
ρ – gęstość materiału

Rozwiązanie tego równania polega na poszukiwaniu minimum takiego funkcjonału, dla którego powyższe równanie jest równaniem Eulera. Ponadto uwzględniając, że dla naszego modelu współczynniki przewodzenia ciepła będą jednakowe we wszystkich kierunkach (tak jak dla materiałów izotropowych), otrzymujemy funkcjonał w postaci:

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

Kolejnym etapem tworzenia naszego modelu, jest założenie warunków brzegowych. Bezpośrednie wprowadzenie warunków brzegowych do powyższego funkcjonału nie jest możliwe, dlatego warunki te wprowadzamy dodając :

$$\int_S\frac{\alpha}{2}(t-t_\infty)^2\,dS+\int_S qt dS$$

S – powierzchnia, na której zadane są warunki brzegowe
α – efektywny współczynnik wymiany ciepła
q – strumień ciepła
t_∞- temperatura otoczenia

Następnie dokonujemy dyskretyzacji problemu poprzez podzielenie rozpatrywanego obszaru na elementy i przedstawieniu temperatury wewnątrz elementu, jako funkcji wartości węzłowych:

$$t=\sum_{i=1}^{n}N_i\,t_i=\{N\}^T\{t\}$$

Uwzględniając to wszystko równanie naszego modelu możemy przedstawić jako:

$$[H]\{t\} + [C]\frac{\partial}{\partial T}\{t\} + \{P\} = 0$$

Gdzie:

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

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

$$\{P\} = -\int_S \alpha\{N\}t_\infty dS$$

Temperatury w węzłach $\{t\}$ zależą od czasu. Przedstawiając je przy wykorzystaniu funkcji kształtu zależnych od czasu, w postaci:

$$\{t\} = \{N_0, N_1\}\begin{Bmatrix}\{t_0\}\\ \{t_1\}\end{Bmatrix}$$

Otrzymujemy po odpowiednich przekształceniach niejawny schemat wyznaczenia temperatury $\{t_1\}$:

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

Opis Metody Elementów Skończonych

Pierwszym etapem MES jest dyskretyzacja naszego modelu, poprzez utworzenie siatki modelu. Daje nam to podział na elementy 2D, które początkowo rozpatrujemy jako elementy lokalne. Dla każdego elementu konieczne jest obliczenie macierzy H, C oraz wektora P, których wzory przedstawiono w poprzednim rozdziale.

W celu obliczenia macierzy H, musimy wyznaczyć pochodne funkcji kształtu po współrzędnych globalnych dla każdego elementu, na które wzór wyprowadza się z poniższej zależności.

$$\frac{\partial N(x,y)}{\partial \xi} = \frac{\partial N(x,y)}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N(x,y)}{\partial y}\frac{\partial y}{\partial \xi}$$

$$\frac{\partial N(x,y)}{\partial \eta} = \frac{\partial N(x,y)}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N(x,y)}{\partial y}\frac{\partial y}{\partial \eta}$$

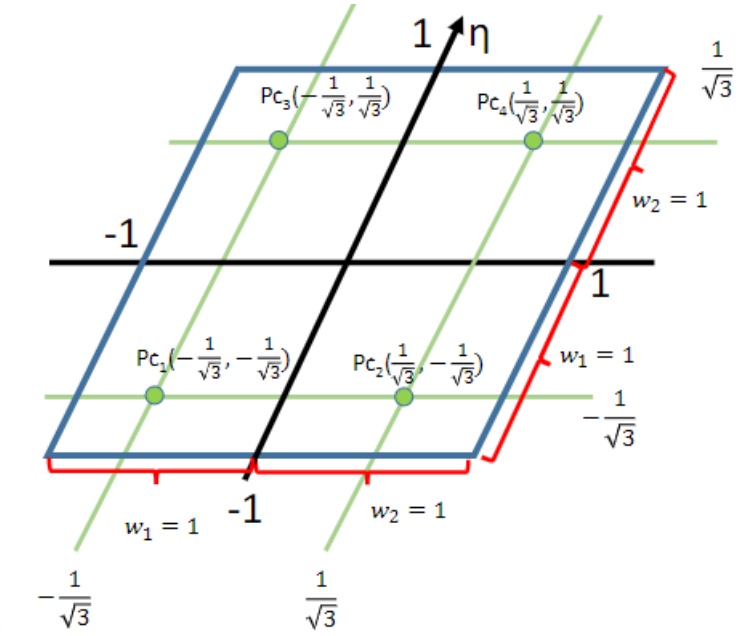
Zapisując te zależności w postaci macierzowej oraz korzystając z zasady rozwiązywania układu równań poprzez mnożenie przez odwrotność,

$$[A]\{b\} = \{c\} \Rightarrow \{b\} = [A]^{-1}\{C\}$$

otrzymujemy równanie pozwalające nam obliczyć szukane pochodne funkcji kształtu po współrzędnych globalnych każdego elementu:

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

Aby to zrobić korzystamy z faktu znajomości wartości funkcji kształtu dla każdego węzła wyznaczonego według schematu całkowania, dla elementu w układzie lokalnym.



Schemat elementu lokalnego 2D, z zaznaczonymi wagami i punktami dla 2 punktowego schematu całkowania metodą Gaussa (slajd z prezentacji z ćwiczeń)

Pochodne współrzędnych globalnych po współrzędnych lokalnych konieczne do wyznaczenia jakobianu przekształcenia wyznaczamy poprzez interpolację (reszta analogicznie):

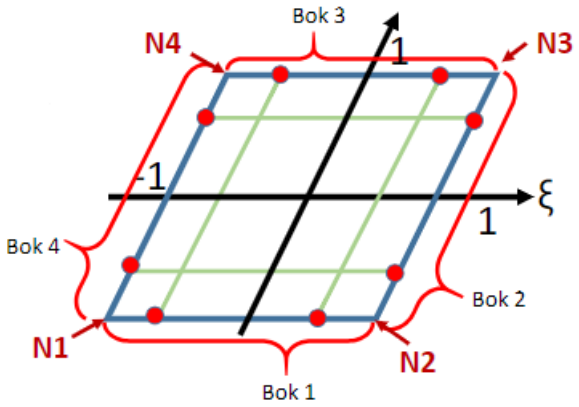
$$\frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3 + \frac{\partial N_4}{\partial \xi} x_4$$

Dzięki powyższym obliczeniom, możemy obliczyć macierz H dla każdego elementu według wzoru:

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

Kolejnym etapem jest wyznaczenie macierzy HBC której agregacja do macierzy H, pozwala uwzględnić część warunku brzegowego narzucanego podczas wymiany ciepła, która jest niezależna od temperatury otoczenia (zgodnie ze wzorem opisywanym w 2 części sprawozdania). Aby to zrobić, należy wykonać odpowiednie całkowanie po „bokach" elementów lokalnych, na które został nałożony konwekcyjny warunek brzegowy.

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



Przykład elementu lokalnego i jego numeracji boków, na których obliczamy konwekcyjny warunek brzegowy

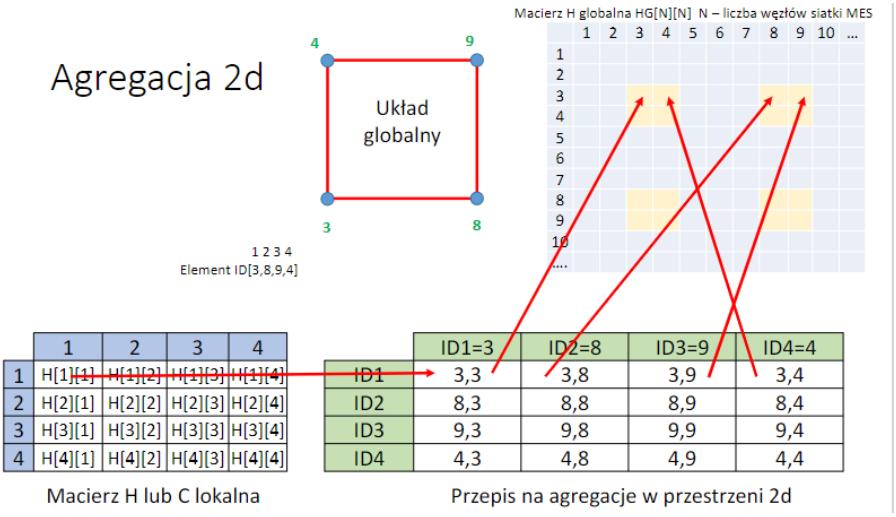
Analogicznie do wyznaczenia macierzy HBC dla każdego elementu siatki, wyznaczamy wektor P, który uwzględnia temperaturę otoczenia, wpływającą na konwekcyjny warunek brzegowy podczas wymiany ciepła. Wzór z którego korzystamy to:

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

Kolejnym etapem jest wyznaczenie macierzy C, dla każdego elementu siatki. W tym celu korzystamy z funkcji kształtu obliczonych dla każdego elementu globalnego.

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

Po wyznaczeniu H, HBC, P oraz C dla każdego elementu, należy je zagregować do jednej macierzy globalnej, aby móc rozwiązać niejawne równanie przewodzenia ciepła. Agregacja jest możliwa dzięki wykorzystaniu globalnych indeksów każdego węzła siatki. Sposób jej wykonania przedstawia poniższy slajd z prezentacji z ćwiczeń:



Po wykonaniu agregacji możemy przejść do rozwiązania równania niejawnej wymiany ciepła. Wiedząc, że wartości temperatury w węzłach $\{t\}$ zależą od czasu oraz przyjmując, że wektor $\{t_0\}$ reprezentuje wartości węzłowe temperatury w początkowej chwili wymiany, to po kroku czasowym, nasz wektor przyjmie postać:

$$\{t\} = \{N_0, N_1\} \begin{Bmatrix} \{t_0\} \\ \{t_1\} \end{Bmatrix}.$$

W powyższym równaniu $\{N_0\}$ oraz $\{N_1\}$ są funkcjami kształtu zależnymi od czasu, a $\{t_1\}$ to temperatury węzłowe po zadanym kroku czasowy. Zakładając, że dla małych kroków czasowych zależność węzłowych temperatur od czasu jest liniowa, otrzymujemy układ równań wymiany ciepła w następującej postaci:

$$[H]\{t_0\} + [C]\frac{\{t_1\} - \{t_0\}}{\Delta T} + \{P\} = 0 .$$

Na podstawie tego równania można wyznaczyć jawny bądź niejawny schemat wyznaczania temperatury końcowej, ale z powodu słabej stabilności rozwiązania jawnego, przyjmujemy niejawny schemat rozwiązania równania w postaci:

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

Charakterystyka kodu

Etap 1) wczytanie danych z pliku

Stworzona klasa Grid, służy do pobrania danych z pliku zapisanego w odpowiedniej formie.

```
class Grid:
    def __init__(self, filename = None):
        self.GlobalData = GlobalData(filename)
        self.Elements = Element(int(self.GlobalData.ElementsNumber))
        self.Nodes = Node(int(self.GlobalData.NodesNumber))
        self.BC = BorderConditions()

        if filename:
            self.loadDataFromFile(filename)

    def loadDataFromFile(self, filename: str):
        with open(filename, "r") as file:
            lines = file.readlines()
            startLineNodes = 0
            endLineNodes = len(lines)

            for line in lines:              //wyszukiwanie fragmentów pliku z zapisanymi węzłami
                if line.strip() == "*Node":
                    startLineNodes = lines.index(line)
                    endLineNodes = startLineNodes + int(self.GlobalData.NodesNumber) + 1
                    break
            ...
            file.seek(0)
                                            //zapis węzłów ze znalezionego zakresu do wcześniej stworzonej
                                            // struktury, umożliwiającej przechowywanie danych
            for line in lines[startLineNodes:endLineNodes]:
                parts = line.strip().split(',')
                if len(parts) >= 3:
                    for part in parts:
                        epsilon = 10**-20
                        if float(part) < epsilon:
                            part = 0.0

                    key = int(parts[0]) - 1
                    value1 = float(parts[1])
                    value2 = float(parts[2])

                    self.Nodes.XY[key] = [value1,value2]
            ...
```

Agreguje ona dane z pliku do obiektu klasy Grid, który przechowuje te dane w klasach Node, Element, GlobalData, BorderCondition. Wczytanie danych polega najpierw na znalezieniu fragmentu pliku przechowującego odpowiednie dane, a w następnej kolejności odczytaniu sekwencji tych danych z ich zapisem do odpowiedniej klasy. Ponadto dla klasy Node, podczas odczytywania danych BC, czyli oznaczeń na które wierzchołki nałożono warunek brzegowy, przypisujemy konkretne oznaczenie w strukturze Node, do tego samego indeksu co indeks węzła.

```
class Node:
    def __init__(self, nodes):
        self.XY = np.empty([nodes,2], dtype=float)
        self.nodeBC = np.zeros([nodes,1], dtype=int)

class Element:
    def __init__(self, elements):
        self.ID = np.empty([elements,4], dtype=int)

class BorderConditions:
    def __init__(self):
        self.BCs = np.array([],dtype=int)
```

Etap 2) stworzenie schematów punktowych całkowania:

Punkty całkowania oraz ich wagi, dla schematów całkowania Gaussa w 1D oraz 2D, zapisujemy jako wartości stałe. Dla scheamtu w 1D, przechowujemy je w tablicy A (wagi), oraz x (współrzędne), natomiast dla schematu 2D, punkty są krotkami zapisanymi w tablicy P.

```
class gaussSchema1D:
    def __init__(self, A, x):
        self.A = A
        self.x = x

class gaussSchema2D:
    def __init__(self, A, P):
        self.A = A
        self.P = P

def gaussIntegrationSchemaiD(integrationPointsNumber, iDimensions):

    //obsługa iDimensions == 1
```

```
    elif(iDimensions == 2):
        if integrationPointsNumber == 2:
            ...
        elif integrationPointsNumber == 3:
            c = -pow(3/5,1/2); d = 8.0 / 9.0; e = 5.0 / 9.0
            P[0] = [c,c]; P[1] = [0,c]; P[2] = [-c,c]; P[3] = [c,0]; P[4] = [0,0]; P[5] = [-c,0]; P[6] = [c,-c]; P[7] = [0,-c];
P[8] = [-c,-c]
            A[0] = A[2] = A[6] = A[8] = e*e
            A[1] = A[3] = A[5] = A[7] = e*d
            A[4] = d*d
        elif integrationPointsNumber == 4:
            ...
        else:
            raise ValueError("\n\tgaussIntegrationSchemaiD: integrationPointsNumber must be in [2,3,4]\n")

        return gaussSchema2D(A,P)
```

Etap 3) Stworzenie klasy UniversalElement2D oraz JacobyMatrix

Przechowuje ona dane wspólne dla wszystkich elementów w układzie lokalnym, tj. funkcje kształtu oraz ich pochodne obliczone na podstawie schematu całkowania, zaimplementowanego w powyższej klasie. Obiekt tej klasy jest przekazywany innym funkcjom obliczającym m.in. element realny, dzięki czemu minimalizujemy ilość obiektów tej klasy.

```
class UniversalElement2D:
    def __init__(self, integrationPointsNumber):
        self.schema2D = gauss.gaussIntegrationSchemaiD(integrationPointsNumber,2)
        P = self.schema2D.P

        self.B = np.zeros([4*integrationPointsNumber,2])
        self.W = np.zeros([4*integrationPointsNumber,1])

        //zdefiniowanie “na sztywno” współrzędnych punktów rozpatrywanych dla warunków
        //brzegowych wraz z ich wagami dla 2/3/4 punktowego schematu całkowania


        //zdefiniowanie tablic pochodnych funkcji kształtu po ksi i eta dla wszystkich punktów
        //dla danego schemata całkowania

        self.shapeFun = ShapeFunctions2D()


        for i in range(integrationPointsNumber**2):
            for j in range(4):
                self.dNdKsiTab[i][j] = self.shapeFun.dNdKsiFun[j](self.schema2D.P[i,1])
                self.dNdEtaTab[i][j] = self.shapeFun.dNdEtaFun[j](self.schema2D.P[i,0])
```

```
    #obliczanie macierzy Hbc dla wszystkich punktow B[i,j]


    for i in range(4):
      for j in range(integrationPointsNumber):
        self.BCis_shapeFuncsTab[i*integrationPointsNumber+j, 0] =
self.shapeFun.N1(self.B[i*integrationPointsNumber+j,0],self.B[i*integrationPointsNumber+j,1])
        self.BCis_shapeFuncsTab[i*integrationPointsNumber+j, 1] =
self.shapeFun.N2(self.B[i*integrationPointsNumber+j,0],self.B[i*integrationPointsNumber+j,1])
        self.BCis_shapeFuncsTab[i*integrationPointsNumber+j, 2] =
self.shapeFun.N3(self.B[i*integrationPointsNumber+j,0],self.B[i*integrationPointsNumber+j,1])
        self.BCis_shapeFuncsTab[i*integrationPointsNumber+j, 3] =
self.shapeFun.N4(self.B[i*integrationPointsNumber+j,0],self.B[i*integrationPointsNumber+j,1])

    //obliczenie wartości funkcji kształtu dla wszystkich punktów danego schematu całkowania

    for i in range(integrationPointsNumber**2):
      for j in range(4):
        self.shapeFunTab[i][j] = self.shapeFun.NFun[j](self.schema2D.P[i,0], self.schema2D.P[i,1])
```

Klasa JacobyMatrix jest odpowiedzialna za obliczenie jakobianu dla każdego punktu całkowania. Oblicza jakobian tylko dla elementu 2D, dlatego sprowadza się do obliczenia pochodnych funkcji kształtu po współrzędnych globalnych danego punktu, które przekazujemy jako parametr funkcji.

```
class JacobyMatrix:
  def __init__(self, nodes_,universalElement ,iPC, integrationPointsNumber):
    self.nodes = nodes_
    self.shapeDerivatives = universalElement
    self.jacobian = np.zeros((2,2), dtype=float)

    self.dXdKsi = 0; self.dYdKsi = 0; self.dXdEta = 0; self.dYdEta = 0

    for i in range(4):
      self.dXdKsi += self.nodes[i,0] * self.shapeDerivatives.dNdKsiTab[iPC][i]
      self.dYdKsi += self.nodes[i,1] * self.shapeDerivatives.dNdKsiTab[iPC][i]
      self.dXdEta += self.nodes[i,0] * self.shapeDerivatives.dNdEtaTab[iPC][i]
      self.dYdEta += self.nodes[i,1] * self.shapeDerivatives.dNdEtaTab[iPC][i]


    self.jacobian = np.array([[self.dXdKsi, self.dYdKsi], [self.dXdEta, self.dYdEta]])
```

Etap 4) Klasa RealElement

Element rzeczywisty określa właściwości każdego elementu siatki. Dla każdego punktu całkowania oblicza jakobian, po czym wykorzystując metodę LU obliczania wyznacznika macierzy, oblicza go w celu wyznaczenia wartości pochodnych funkcji kształtu po współrzędnych globalnych.

```
class RealElement2D:

  def __init__(self,integrationPointsNumber,universalElement,nodes):

  //definicja tablic pochodnych funkcji kształtu
  ...
    for i in range(integrationPointsNumber**2):
      self.jacobyObj = ue.JacobyMatrix(nodes,universalElement,i, integrationPointsNumber)
      self.jacobian = self.jacobyObj.jacobian

      lu, piv = lu_factor(self.jacobian)
      identity = np.eye(self.jacobian.shape[0])

      self.detJ_iPC[i,0] = np.prod(np.diagonal(lu))
      self.scaleMatrix = lu_solve((lu,piv), identity)

      for j in range(4):
        self.dNdXTab[i][j] = self.scaleMatrix[0][0]*self.dNdKsiTab[i][j] +
self.scaleMatrix[0][1]*self.dNdEtaTab[i][j]
        self.dNdYTab[i][j] = self.scaleMatrix[1][0]*self.dNdKsiTab[i][j] +
self.scaleMatrix[1][1]*self.dNdEtaTab[i][j]
```

Etap 5) Klasa MatrixH

Klasa ta umożliwia obliczenie macierzy H dla każdego elementu. Pierwsza pętla oblicza macierze H dla każdego punktu całkowania, natomiast druga pętla agreguje te wartości tworząc macierz H dla całego elementu.

```
class matrixH:

  def __init__(self,integrationPointsNumber,realElement, heatTransferCoeff = 30):

    ...
    for i in range(integrationPointsNumber**2):
      self.HiMatrixesTable[i] = heatTransferCoeff * (np.outer(self.dNdXTab[i],self.dNdXTab[i]) +
np.outer(self.dNdYTab[i], self.dNdYTab[i])) * self.dV[i,0]
```

```
    self.gaussWages = self.realElement.universalElement2D.schema2D.A
    self.H = np.zeros((4,4),dtype=float)
    for i in range(integrationPointsNumber**2):
      self.H += self.HiMatrixesTable[i] * self.gaussWages[i]
```

Etap 5) Macierz HBC

Macierz HBC, to macierz wprowadzająca część warunku brzegowego, który jest nałożony na całą modelowaną powierzchnię (w przeciwieństwie do wektora P). Korzystając z zapisanych podczas pobierania danych z pliku warunków brzegowych, których indeksy odpowiadają wierzchołkom, sprawdzamy czy na dana parę wierzchołków został nałożony warunek brzegowy. Jeśli tak, to obliczamy odpowiednią całkę powierzchniową, dla ściany elementu, na który został nałożony warunek brzegowy. Po obliczeniu cząstkowych macierzy HBCs dla każdej ściany, sumujemy te wartości do wspólnej macierzy HBC całego elementu.

```
class matrixHBC:
  def
__init__(self,integrationPointsNumber,universalElement,nodes,nodesBounderConditionFlags,alpha):
    self.HBCs = np.zeros((4,4,4), dtype=float)


    for side in range(4):
      if(nodesBounderConditionFlags[side][0] == 1 and nodesBounderConditionFlags[(side + 1) % 4][0]
== 1):
        x1, y1 = nodes[side]
        x2, y2 = nodes[(side + 1) % 4]
        L = np.sqrt((x2 - x1)**2 + (y2 - y1)**2
        detJ = L/2.0
        for borderPoint in range (integrationPointsNumber):
          self.HBCs[side] += alpha * np.outer(BCis_shapeFuncsTab[side*integrationPointsNumber +
borderPoint], BCis_shapeFuncsTab[side*integrationPointsNumber + borderPoint]) * W[side *
integrationPointsNumber + borderPoint]
        self.HBCs[side] *= detJ

    self.HBC = np.zeros((4,4),dtype=float)
    for i in range(4):
      self.HBC += self.HBCs[i]
```

Etap 6) Wektor P

Wektor P obliczamy analogicznie do macierzy HBC, z tym, że obliczamy inną całkę, zależną od temperatury otoczenia. Wektor ten odwzorowuje drugą część warunku brzegowego, nałożonego na wybrane ściany modelu.

```
class vectorP:
   def
__init__(self,integrationPointsNumber,universalElement,nodes,nodesBounderConditionFlags,alpha,
Tambient):


      for side in range(4):
         if(nodesBounderConditionFlags[side][0] == 1 and nodesBounderConditionFlags[(side + 1) % 4][0]
== 1):
            x1, y1 = nodes[side]
            x2, y2 = nodes[(side + 1) % 4]
            L = np.sqrt((x2 - x1)**2 + (y2 - y1)**2)
            detJ = L/2.0
            for borderPoint in range (integrationPointsNumber):
               self.vectorsP[side] += alpha * BCis_shapeFuncsTab[side*integrationPointsNumber +
borderPoint] * W[side * integrationPointsNumber + borderPoint] * Tambient
            self.vectorsP[side] *= detJ

      self.vectorP = np.zeros((1,4), dtype=float)
      for i in range(4):
         self.vectorP += self.vectorsP[i]
```

Etap 7) Macierz C

Macierz C obrazuje ilość skumulowanego ciepła dla elementu siatki. Do jej obliczenia konieczna jest znajomość gęstości materiału oraz jego przewodności cieplnej. Podobnie jak wcześniej obliczamy najpierw macierze C dla każdego punktu całkowania, a następnie agregujemy je do macierzy C dla całego elementu.

```
class matrixC:
   def __init__(self,integrationPointsNumber,realElement,specificHeat, density):

      self.CMatrixesTable = np.zeros((integrationPointsNumber**2,4,4),dtype=float)

      for i in range(integrationPointsNumber**2):
         self.CMatrixesTable[i] = specificHeat * density * np.outer(self.NTab[i], self.NTab[i]) * self.dV[i,0]

      for i in range(integrationPointsNumber**2):
         self.C += self.CMatrixesTable[i] * self.gaussWages[i]
```

Etap 8) Agregacja macierzy H,HBC, C oraz wektora P

Aby obliczyć równanie niejawne wymiany ciepła, potrzebujemy globalnych macierzy H,HBC, C oraz wektora P, które opisują cały układ. Dzięki zastosowaniu indeksów dla każdego węzła każdego elementu możemy zidentyfikować nasz element rzeczywisty, w globalnej macierzy obrazującej cały układ. Aby to zrobić najpierw tworzymy sobie tablicę odwzorowań indeksów lokalnych na indeksy w macierzy globalnej, a następnie przepisujemy tam wartości z każdego elementu.

```
class Agregation:
  def __init__(self, N):
    ...

  def aggregate(self, H, HBC, P, C, elementIDs):
    H_HBC = H + HBC

    n = len(elementIDs)
    self.IDsFromLocalToGLobalH_HBC = np.zeros((n,n,2),dtype=int)
    self.IDsFromLocalToGLobalP = np.zeros((n),dtype=int)

    for i in range(n):
      for j in range(n):
        self.IDsFromLocalToGLobalH_HBC[i][j][0] = elementIDs[i]
        self.IDsFromLocalToGLobalH_HBC[i][j][1] = elementIDs[j]

    for side in range(n):
      self.IDsFromLocalToGLobalP[side] = elementIDs[side]

    for i in range(n):
      for j in range(n):
        self.HG[self.IDsFromLocalToGLobalH_HBC[i][j][0], self.IDsFromLocalToGLobalH_HBC[i][j][1]] +=
H_HBC[i,j]

    for side in range(n):
      self.PG[self.IDsFromLocalToGLobalP[side]] += P[0][side]

    for i in range(n):
      for j in range(n):
        self.CG[self.IDsFromLocalToGLobalH_HBC[i][j][0], self.IDsFromLocalToGLobalH_HBC[i][j][1]] +=
C[i,j]
```

Etap 9) Rozwiązanie niejawnego układu równań wymiany ciepła

W celu rozwiązania symulacji, w której temperatura startowa każdego kolejnego kroku to temperatura wynikowa poprzedniego, układamy odpowiednią pętlę, w której wyniki każdej iteracji zapisywane są w dodatkowej tablicy. Do rozwiązania układu równań wykorzystujemy

bibliotekę scipy, która umożliwia rozwiązywanie układów równań przy wykorzystaniu macierzy rzadkich, co poprawia szybkość działania naszego kodu ponieważ nasze macierze H oraz C zawierają dużo elementów zerowych.

```
class Solve:
    def __init__(self, H, P, C, initialTemperature, simulationStepTime, simulationTime):

        H_sparse = scipy.sparse.csr_matrix(H)
        C_sparse = scipy.sparse.csr_matrix(C)

        self.t_nonstationary_table = np.empty((int(simulationTime/simulationStepTime), len(P)),
dtype=float)
        self.t_nonstationary_end = np.full((len(P)), 0)

        t0 = np.full((len(P)), initialTemperature)

        A = H_sparse + C_sparse/simulationStepTime

        for i in range(int(simulationTime/simulationStepTime)):
            B = (C_sparse/simulationStepTime) @ t0 - (-P)
            self.t_nonstationary_table[i] = scipy.sparse.linalg.spsolve(A,B)
            t0 = self.t_nonstationary_table[i].copy()
```

Etap 10) Utworzenie plików, do zobrazowania symulacji za pomocą programu ParaView

```
class ParaView:
    def __init__(self, selectedGrid, nodesTemperatureDataSimulationTab):
        self.create_vtk_simulation_files()

    def create_vtk_file(self,filename, iteration):
        with open(filename, 'w') as f:
            //zapis danych do pliku w odpowiedniej formie

    def create_vtk_simulation_files(self):
        directory = "simulation"
        if os.path.exists(directory):
            shutil.rmtree(directory)
        os.makedirs(directory)
        for i in range(len(self.dataTab)):
            filename = os.path.join(directory, f"step{i+1}.vtk")
            self.create_vtk_file(filename, i)
```

Porównanie wyników oprogramowania z wynikami testowymi zamieszczonymi na UPEL-u

W celu umożliwienia debugowania, po uruchomieniu programu z opcją -v lub --verbose, program wypisze wszystkie kroki wykonywanych obliczeń do pliku output.txt. Na podstawie tego pliku, porównałem wartości z tymi zamieszczonymi na UPEL. Poniżej przedstawiam wyniki końcowe symulacji dla wszystkich siatek testowych.

Wyniki dla sitaki 4x4:

```
PS C:\Users\kacpe\OneDrive\Pulpit\MES\projekt> .\main.py -p
==================== Krok 1 ============================
Czas: 50.00, Max temp: 365.82, Min temp: 110.04
--------------------------------------------------------

==================== Krok 2 ============================
Czas: 100.00, Max temp: 502.59, Min temp: 168.84
--------------------------------------------------------

==================== Krok 3 ============================
Czas: 150.00, Max temp: 587.37, Min temp: 242.80
--------------------------------------------------------

==================== Krok 4 ============================
Czas: 200.00, Max temp: 649.39, Min temp: 318.61
--------------------------------------------------------

==================== Krok 5 ============================
Czas: 250.00, Max temp: 700.07, Min temp: 391.26
--------------------------------------------------------

==================== Krok 6 ============================
Czas: 300.00, Max temp: 744.06, Min temp: 459.04
--------------------------------------------------------

==================== Krok 7 ============================
Czas: 350.00, Max temp: 783.38, Min temp: 521.59
--------------------------------------------------------

==================== Krok 8 ============================
Czas: 400.00, Max temp: 818.99, Min temp: 579.03
--------------------------------------------------------

==================== Krok 9 ============================
Czas: 450.00, Max temp: 851.43, Min temp: 631.69
--------------------------------------------------------

==================== Krok 10 ===========================
Czas: 500.00, Max temp: 881.06, Min temp: 679.91
--------------------------------------------------------
```

```
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
```

Wyniki dla siatki 4x4 mixed

```
PS C:\Users\kacpe\OneDrive\Pulpit\MES\projekt> .\main.py
=================== Krok 1 ============================
Czas: 50.00, Max temp: 374.67, Min temp: 95.16
--------------------------------------------------------

=================== Krok 2 ============================
Czas: 100.00, Max temp: 505.95, Min temp: 147.66
--------------------------------------------------------

=================== Krok 3 ============================
Czas: 150.00, Max temp: 586.99, Min temp: 220.18
--------------------------------------------------------

=================== Krok 4 ============================
Czas: 200.00, Max temp: 647.28, Min temp: 296.75
--------------------------------------------------------

=================== Krok 5 ============================
Czas: 250.00, Max temp: 697.33, Min temp: 370.98
--------------------------------------------------------

=================== Krok 6 ============================
Czas: 300.00, Max temp: 741.22, Min temp: 440.57
--------------------------------------------------------

=================== Krok 7 ============================
Czas: 350.00, Max temp: 781.24, Min temp: 504.90
--------------------------------------------------------

=================== Krok 8 ============================
Czas: 400.00, Max temp: 817.42, Min temp: 564.01
--------------------------------------------------------

=================== Krok 9 ============================
Czas: 450.00, Max temp: 850.26, Min temp: 618.19
--------------------------------------------------------

=================== Krok 10 ===========================
Czas: 500.00, Max temp: 880.19, Min temp: 667.78
--------------------------------------------------------
```

```
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
```

Wyniki dla siatki 31x31 kwadrat

```
Temperatury max i min w kazdym kroku symulacji (krok -> [s])
Czas: 1.00, Max temp: 149.56, Min temp: 100.00
Czas: 2.00, Max temp: 177.44, Min temp: 100.00
Czas: 3.00, Max temp: 197.27, Min temp: 100.00
Czas: 4.00, Max temp: 213.15, Min temp: 100.00
Czas: 5.00, Max temp: 226.68, Min temp: 100.00
Czas: 6.00, Max temp: 238.61, Min temp: 100.00
Czas: 7.00, Max temp: 249.35, Min temp: 100.00
Czas: 8.00, Max temp: 259.17, Min temp: 100.00
Czas: 9.00, Max temp: 268.24, Min temp: 100.00
Czas: 10.00, Max temp: 276.70, Min temp: 100.00
Czas: 11.00, Max temp: 284.64, Min temp: 100.00
Czas: 12.00, Max temp: 292.13, Min temp: 100.00
Czas: 13.00, Max temp: 299.24, Min temp: 100.00
Czas: 14.00, Max temp: 306.00, Min temp: 100.01
Czas: 15.00, Max temp: 312.45, Min temp: 100.01
Czas: 16.00, Max temp: 318.63, Min temp: 100.01
Czas: 17.00, Max temp: 324.56, Min temp: 100.02
Czas: 18.00, Max temp: 330.27, Min temp: 100.03
Czas: 19.00, Max temp: 335.77, Min temp: 100.05
Czas: 20.00, Max temp: 341.08, Min temp: 100.06
```

```
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
100.0382207726261 330.27745133351596
100.05276279329537 335.77922748329735
100.07184163487159 341.0920092636545
```

Wyniki dla siatki 31x31 trapez

Temperatury max i min w kazdym kroku symulacji (krok -> [s])
Czas: 1.00, Max temp: 166.94, Min temp: 100.00
Czas: 2.00, Max temp: 207.23, Min temp: 100.00
Czas: 3.00, Max temp: 236.29, Min temp: 100.00
Czas: 4.00, Max temp: 259.47, Min temp: 100.00
Czas: 5.00, Max temp: 279.03, Min temp: 100.00
Czas: 6.00, Max temp: 296.12, Min temp: 100.00
Czas: 7.00, Max temp: 311.38, Min temp: 100.00
Czas: 8.00, Max temp: 325.24, Min temp: 100.00
Czas: 9.00, Max temp: 337.95, Min temp: 100.00
Czas: 10.00, Max temp: 349.73, Min temp: 100.01
Czas: 11.00, Max temp: 360.72, Min temp: 100.01
Czas: 12.00, Max temp: 371.04, Min temp: 100.02
Czas: 13.00, Max temp: 380.77, Min temp: 100.03
Czas: 14.00, Max temp: 389.99, Min temp: 100.05
Czas: 15.00, Max temp: 398.75, Min temp: 100.07
Czas: 16.00, Max temp: 407.10, Min temp: 100.10
Czas: 17.00, Max temp: 415.08, Min temp: 100.15
Czas: 18.00, Max temp: 422.73, Min temp: 100.21
Czas: 19.00, Max temp: 430.08, Min temp: 100.28
Czas: 20.00, Max temp: 437.15, Min temp: 100.36
Czas: 21.00, Max temp: 443.96, Min temp: 100.47
Czas: 22.00, Max temp: 450.54, Min temp: 100.60
Czas: 23.00, Max temp: 456.89, Min temp: 100.74
Czas: 24.00, Max temp: 463.05, Min temp: 100.91
Czas: 25.00, Max temp: 469.01, Min temp: 101.11
Czas: 26.00, Max temp: 474.79, Min temp: 101.33
Czas: 27.00, Max temp: 480.41, Min temp: 101.58
Czas: 28.00, Max temp: 485.87, Min temp: 101.86
Czas: 29.00, Max temp: 491.18, Min temp: 102.17
Czas: 30.00, Max temp: 496.35, Min temp: 102.50
Czas: 31.00, Max temp: 501.39, Min temp: 102.87
Czas: 32.00, Max temp: 506.30, Min temp: 103.27
Czas: 33.00, Max temp: 511.10, Min temp: 103.70
Czas: 34.00, Max temp: 515.78, Min temp: 104.16
Czas: 35.00, Max temp: 520.35, Min temp: 104.65
Czas: 36.00, Max temp: 524.82, Min temp: 105.17
Czas: 37.00, Max temp: 529.20, Min temp: 105.73
Czas: 38.00, Max temp: 533.48, Min temp: 106.32
Czas: 39.00, Max temp: 537.67, Min temp: 106.93
Czas: 40.00, Max temp: 541.77, Min temp: 107.58
Czas: 41.00, Max temp: 545.79, Min temp: 108.26
Czas: 42.00, Max temp: 549.73, Min temp: 108.97
Czas: 43.00, Max temp: 553.60, Min temp: 109.71
Czas: 44.00, Max temp: 557.39, Min temp: 110.48
Czas: 45.00, Max temp: 561.11, Min temp: 111.28
Czas: 46.00, Max temp: 564.77, Min temp: 112.11
Czas: 47.00, Max temp: 568.36, Min temp: 112.96
Czas: 48.00, Max temp: 571.88, Min temp: 113.85
Czas: 49.00, Max temp: 575.35, Min temp: 114.76
Czas: 50.00, Max temp: 578.76, Min temp: 115.69
Czas: 51.00, Max temp: 582.11, Min temp: 116.65
Czas: 52.00, Max temp: 585.40, Min temp: 117.64
Czas: 53.00, Max temp: 588.64, Min temp: 118.65
Czas: 54.00, Max temp: 591.83, Min temp: 119.69
Czas: 55.00, Max temp: 594.97, Min temp: 120.75
Czas: 56.00, Max temp: 598.07, Min temp: 121.83
Czas: 57.00, Max temp: 601.11, Min temp: 122.93
Czas: 58.00, Max temp: 604.11, Min temp: 124.06
Czas: 59.00, Max temp: 607.07, Min temp: 125.21
Czas: 60.00, Max temp: 609.98, Min temp: 126.37

Wartości z pliku testowego
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
100.06607630236047 398.7329444084316
100.09890604703071 407.0839999228044
100.14239869485296 415.06740851188135
100.19846510735009 422.71755059301114
100.2690415128534 430.063898923412
100.35604999410768 437.13194022620934
100.46136360581374 443.9438861744528
100.58677695817498 450.51922965500603
100.73398255211671 456.8751855104281
100.90455274455185 463.02704374695475
101.09992694675618 468.9884555278435
101.32140349391727 474.77166692761165
101.57013554447936 480.38771163654076
101.84713035050619 485.8465710811325
102.15325126484217 491.1573084394593
102.48922190137662 496.328181562518
102.85563192917733 501.3667387153518
103.25294405113215 506.2799002224972
103.68150178722517 511.07402846921707
104.14153774807718 515.7549882221302
104.63318214384935 520.3281988536518
105.15647132609867 524.7986797574265
105.71135620545562 529.171090007135
106.29771042636658 533.4497631244197
106.91533821210513 537.6387376719981
107.56398181948711 541.741784267583
108.24332856395998 545.7624295164479
108.9530173926518 549.7039772807208
109.69264499625561 553.5695276382133
110.46177146086819 557.3619938296619
111.259925468644 561.0841174486852
112.08660906181586 564.738482091664
112.94130198867555 568.3275256537182
113.82346565282369 571.8535514309826
114.7325466886738 575.3187381674231
115.66798018705074 578.7251491659451
116.62919259496024 582.074740567812
117.61560431334811 585.3693688909699
118.62663201608873 588.6107979064777
119.66169071257625 591.8007049224022
120.72019557527858 594.9406865361285
121.801563552473 598.0322639087257
122.90521478518018 601.0768876087712
124.0305738460866 604.0759420675566
125.17707081701873 607.0307496828312
126.34414222032 609.9425746041862

Wnioski

Wykonane w ramach ćwiczeń oprogramowanie do symulacji wymiany ciepła z narzuconymi warunkami brzegowymi poprawnie przechodzi testy porównawcze dla siatek testowych. Dalszy rozwój oprogramowania umożliwia wprowadzenie dodatkowych faz materiału, innych warunków brzegowych, czy zmianę elementów skończonych w celu modelowania bardziej skomplikowanych geometrycznie obiektów. Napisany kod został wzbogacony o kilka optymalizacji takich jak:

➔ obliczanie wyznacznika macierzy za pomocą rozkładu macierzy LU :

Wyznacznik macierzy $\mathbf{A}$ tej postaci można obliczyć korzystając z twierdzenia Cauchy'ego:

$$\mathbf{det(A) = det(L \cdot U) = det(L) \cdot det(U)},$$

oraz z faktu, że wyznacznik macierzy trójkątnej jest iloczynem elementów na przekątnej:

$$\mathbf{det(L)} = l_{11} \cdot l_{22} \cdot \ldots \cdot l_{nn},$$
$$\mathbf{det(U)} = u_{11} \cdot u_{22} \cdot \ldots \cdot u_{nn}.$$

➔ obliczanie równania symulacji wymiany ciepła przy wykorzystaniu macierzy rzadkich

Ponadto w celu sprawdzenia czasu działania mojego kodu oraz sprawdzenia optymalności jego napisania wykorzystałem profiler cProfile, który umożliwił mi sprawdzenie czasu i ilości wywołań poszczególnych funkcji w czasie działania programu:

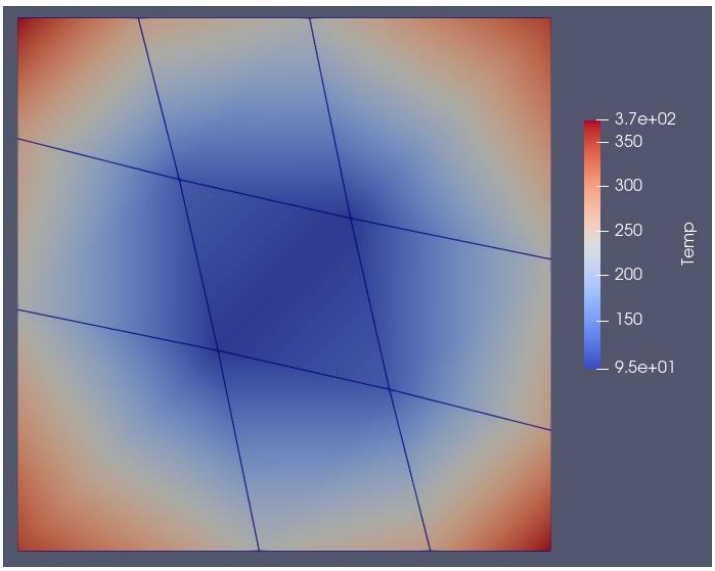
```
        8962 function calls (8942 primitive calls) in 0.075 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    30    0.013    0.000    0.013    0.000 {built-in method builtins.print}
    36    0.008    0.000    0.009    0.000 C:\Python311\Lib\site-packages\scipy\linalg\_decomp_lu.py:89(lu_solve)
    10    0.007    0.001    0.007    0.001 {built-in method io.open}
    10    0.004    0.000    0.004    0.000 {built-in method nt.unlink}
    10    0.003    0.000    0.003    0.000 {method '__exit__' of '_io._IOBase' objects}
     9    0.002    0.000    0.018    0.002 C:\Users\kacpe\OneDrive\Pulpit\MES\projekt\real_element.py:17(__init__)
    10    0.002    0.000    0.014    0.001 C:\Users\kacpe\OneDrive\Pulpit\MES\projekt\visualizeData.py:15(create_vtk_file)
     9    0.002    0.000    0.002    0.000 C:\Users\kacpe\OneDrive\Pulpit\MES\projekt\aggregation.py:13(aggregate)
```

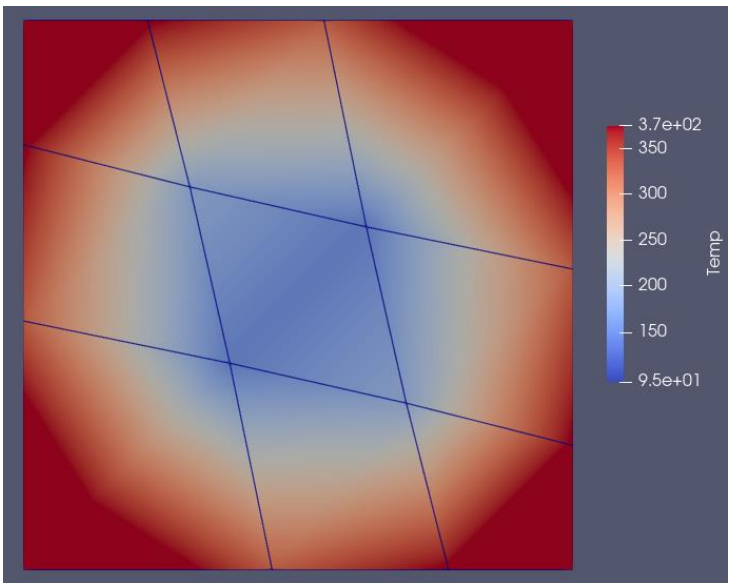
Wyniki dla siatki 4x4 oraz 4 punktowego schematu całkowania. Wynik około 0,075 [s], przy czym około 0,02[s] to czas związany z operacjami wypisywania wyników symulacji na terminal oraz tworzenia plików do symulacji.

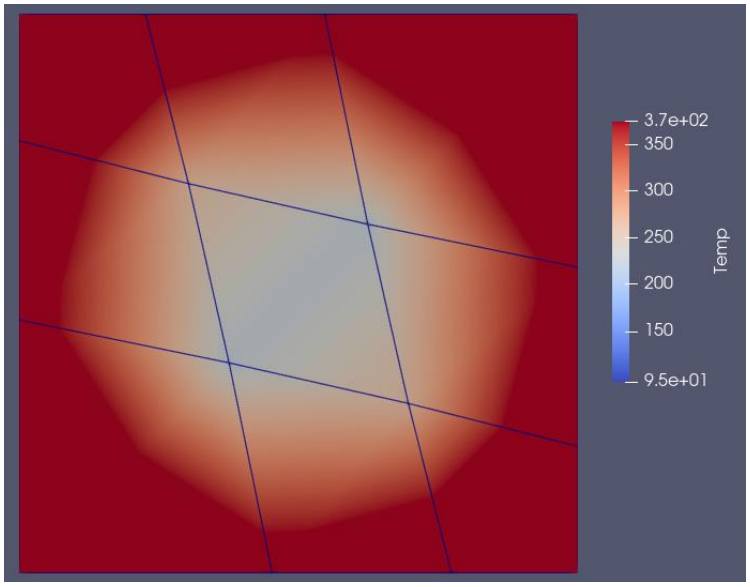
Dla siatki 31x31 kwadrat i 4 punktowego schematu całkowania całkowity czas wynosi trochę ponad 3 [s]. Najdłużej wykonywana operacja to rozkład macierzy lu, który jest optymalizacją obliczania wyznacznika jakobianu.

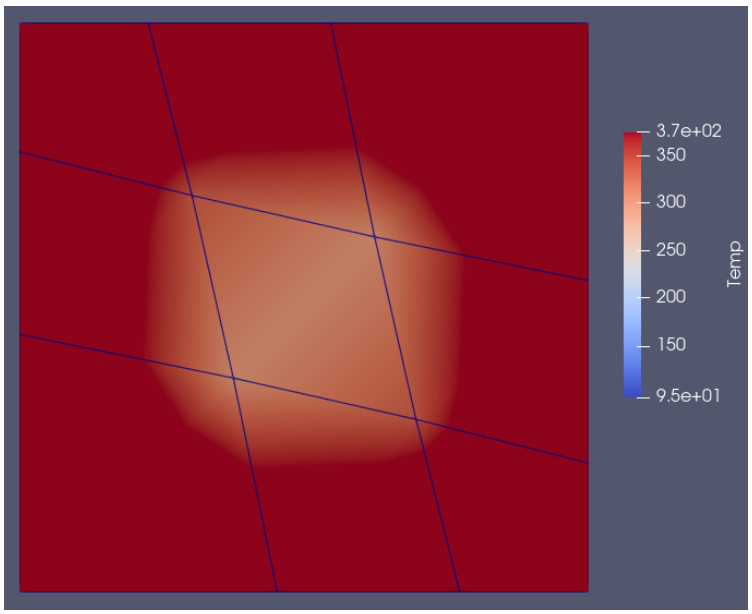
```
         1193808 function calls (1193788 primitive calls) in 3.264 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 14400    0.991    0.000    1.115    0.000 C:\Python311\Lib\site-packages\scipy\linalg\_decomp_lu.py:89(lu_solve)
   900    0.329    0.000    2.312    0.003 C:\Users\kacpe\OneDrive\Pulpit\MES\projekt\real_element.py:17(__init__)
 14400    0.218    0.000    0.273    0.000 C:\Users\kacpe\OneDrive\Pulpit\MES\projekt\universal_element.py:168(__init__)
 43680    0.161    0.000    0.195    0.000 C:\Python311\Lib\site-packages\numpy\core\numeric.py:858(outer)
```

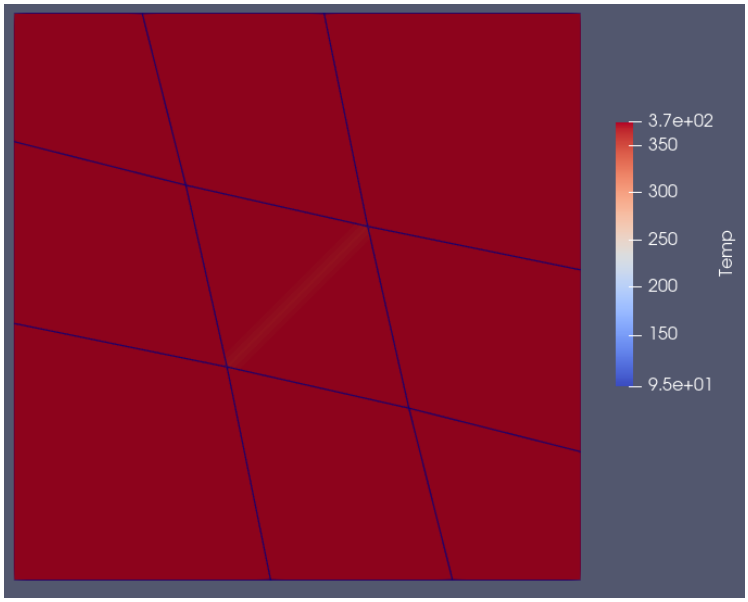
Kolejne kroki symulacji dla siatki 4x4 mixed:

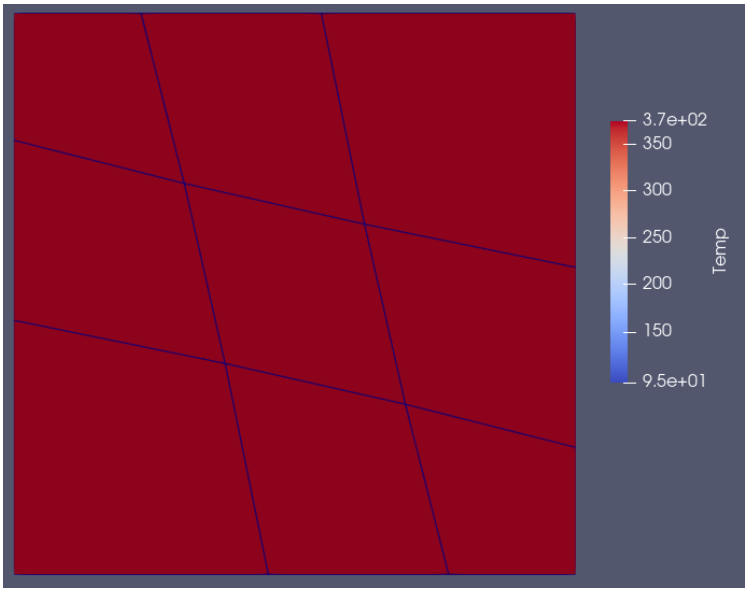





3.7e+02
350
300
250
200
150
9.5e+01
Temp


3.7e+02
350
300
250
200
150
9.5e+01
Temp


3.7e+02
350
300
250
200
150
9.5e+01
Temp



Wartość analogiczna dla kroków 7,8,9,10