

METODA ELEMENTÓW SKOŃCZONYCH



TEMAT: Stworzenie programu MES do analizy przepływu ciepła w języku Python

AUTOR: Przemysław Janiszewski nr. 411890, grupa 2

Spis treści:

WSTĘP TEORETYCZNY	3
OPIS MODELU Z WARUNKAMI BRZEGOWYMI.....	3
ZGRUBNY OPIS MES – PODSTAWOWE POJĘCIA:	5
ZGRUBNA CHARAKTERYSTYKA KODU:	7
PORÓWNANIE WYNIKÓW OPROGRAMOWANIA Z TESTAMI.....	15
WNIOSKI	22
SYMULACJA (SIATKA 4_4_MIX):	23

Wstęp teoretyczny

Metoda Elementów Skończonych (FEM, z ang. Finite Element Method) to zaawansowana metoda numerycznego rozwiązywania problemów brzegowych. Polega ona na wyznaczeniu ciągłego rozkładu badanej zmiennej na podstawie dyskretnego zestawu jej punktów. W procesie tym niezwykle ważne są tzw. elementy skończone, które powstają w wyniku podziału dziedziny problemu, na skończoną liczbę części. MES znajduje szerokie zastosowanie w różnych dziedzinach, takich jak mechanika konstrukcji, przepływ ciepła, przepływ cieczy, dynamika, kinematyka i statyka maszyn, a także oddziaływanie elektrostatyczne, magnetostatyczne i elektromagnetyczne. Główną zaletą MES jest możliwość uzyskiwania rozwiązań dla obszarów o skomplikowanych kształtach, dla których nie jest możliwe przeprowadzenie ścisłych obliczeń analitycznych. Ponadto, MES umożliwia modelowanie materiałów wielofazowych oraz materiałów, których właściwości są funkcją temperatury. Metoda posiada także, wady, z których wyróżnić możemy dużą złożoność modelowania jak i obliczeń – które zazwyczaj trwają tak długo, że nie mogą być przeprowadzane w czasie rzeczywistym.

Opis modelu z warunkami brzegowymi

W zaprojektowanym przeze mnie programie zająłem się analizą niestacjonarnego (zmennego w czasie) przepływu ciepła. Punktem wyjścia do całego programu, jest poniższe równanie Fouriera opisujące badane zjawisko:

$$\operatorname{div}(k(t)\nabla t) = c \cdot \rho \cdot \frac{\partial t}{\partial \tau}$$

$$\frac{\partial}{\partial x} \left(k_x(t) \frac{\partial t}{\partial x} \right) + \frac{\partial}{\partial y} \left(k_y(t) \frac{\partial t}{\partial y} \right) + \frac{\partial}{\partial z} \left(k_z(t) \frac{\partial t}{\partial z} \right) - c\rho \frac{dt}{d\tau} = 0$$

Rozwiązanie powyższego równania, można sprowadzić do zadania polegającego na poszukiwaniu minimum odpowiedniego funkcjonału, wyznaczonego według zasad rachunku wariacyjnego. Przy założeniu, że $k_x(t) = k_y(t) = k_z(t) = k(t)$ (materiał izotropowy) otrzymujemy

$$\int_V \left(\frac{k(t)}{2} \left(\left(\frac{\partial t}{\partial x} \right)^2 + \left(\frac{\partial t}{\partial y} \right)^2 + \left(\frac{\partial t}{\partial z} \right)^2 \right) + c\rho \frac{\partial t}{\partial \tau} \right) dV$$

By uzyskać poprawne wyniki, konieczne jest narzucenie odpowiednich warunków brzegowych. Bezpośrednie ich wprowadzenie do funkcjonału nie jest możliwe. Narzuca się je więc w postaci całek, gdzie pierwszy składnik reprezentuje intensywności przepływu ciepła na granicy styku ciała i płynu, a drugi składnik reprezentuje strumień ciepła przepływający przez powierzchnię ciała. (Trzeci warunek brzegowy)

$$\int_S \left(\frac{\alpha}{2} (t - t_\infty)^2 \right) dS + \int_S qt dS = 0$$

Rozwiążanie problemu (z dołożonymi warunkami brzegowymi) i po uprzedniej dyskretyzacji, sprowadza się do obliczenia pochodnych cząstkowych otrzymanego w ten sposób funkcjonału. W konsekwencji otrzymujemy następujący układ równań:

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

gdzie:

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

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

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

Zgrubny opis MES – podstawowe pojęcia:

Macierz sztywności (dla elementu 2D). Reprezentuje ona zdolność elementu do oporu przeciwko deformacji. W kontekście problemów termicznych, macierz sztywności jest często związana z przewodnictwem ciepła w materiale:

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

Macierz HBC: Jest to macierz sztywności brzegowej. Reprezentuje ona wpływ warunków brzegowych na sztywność systemu. W kontekście problemów termicznych, może to być związane z przepływem ciepła przez brzegi analizowanego obszaru.

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

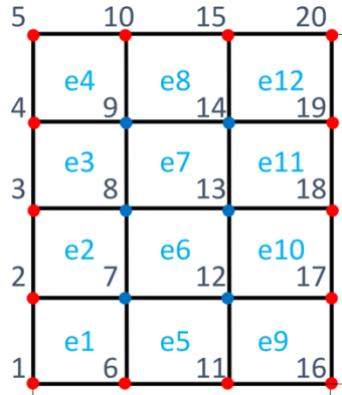
Macierz C: Jest to macierz pojemności cieplnej. Reprezentuje ona zdolność elementu do magazynowania energii cieplnej. W kontekście problemów termicznych, macierz pojemności cieplnej jest związana z gęstością i ciepłem właściwym materiału.

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

Wektor P: Jest to wektor obciążen. Każdy element tego wektora reprezentuje obciążenie cieplne przyłożone do konkretnego węzła w systemie. W kontekście problemów termicznych, wektor obciążen jest często związany z generowanym ciepłem lub przepływem ciepła do lub z systemu.

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

Dyskretyzacja: MES polega na podziale ciągłego obszaru na dyskretny zbiór prostych podobszarów, nazywanych elementami skończonymi. Każdy element skończony jest zdefiniowany przez zestaw węzłów, a poszukiwane rozwiązanie jest interpolowane w każdym elemencie za pomocą prostych funkcji bazowych:



Interpolacja: W MES, poszukiwane rozwiązanie jest interpolowane na dyskretnym zbiorze węzłów, które powstają w wyniku dyskretyzacji dziedziny. Interpolacja jest realizowana za pomocą prostych funkcji bazowych, które są zdefiniowane lokalnie dla każdego elementu skońzonego.

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

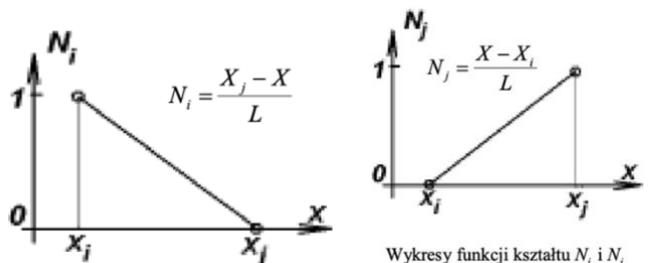
Rozwiązywanie układu równań: MES prowadzi do układu równań algebraicznych, które można rozwiązać za pomocą różnych technik numerycznych. Otrzymujemy w ten sposób przybliżone rozwiązanie oryginalnego problemu brzegowego:

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

Funkcje kształtu: to zestaw funkcji opisujących zmianę wartości danej wielkości wewnątrz elementu skońzonego. Są one zdefiniowane lokalnie i używane do interpolacji rozwiązania na dyskretnym zbiorze węzłów. Są zawsze tak zbudowane, aby w węzłach, których dotyczą, ich wartości wynosiły 1, a w pozostałych węzłach przyjmowały wartość 0 :

$$N_i = \frac{X_j - X}{L} \quad N_j = \frac{X - X_i}{L}$$

$$t = N_i t_i + N_j t_j = [N] \{t\}$$



Zgrubna charakterystyka kodu:

- Klasy występujące w kodzie:

`class Global_Data` – odpowiedzialna za wczytanie i przechowywanie parametrów opisujących konkretną symulację.

`class Node` – odpowiedzialna za reprezentowanie węzła elementu skończonego i zarządzanie jego składowymi

`class Element` – odpowiedzialna za reprezentowanie elementu skończonego i zarządzanie jego składowymi

`class Grid` – odpowiedzialna za wczytanie i zarządzanie węzłami i elementami

`class Matrix_H` – odpowiedzialna za tworzenie i zarządzanie lokalnymi macierzami sztywności dla wszystkich elementów siatki.

`class Matrix_HBC` – odpowiedzialna za tworzenie i zarządzanie lokalnymi macierzami sztywności brzegowej dla wszystkich elementów siatki.

`class Matrix_C` – odpowiedzialna za tworzenie i zarządzanie lokalnymi macierzami pojemności ciepła dla wszystkich elementów siatki.

`class VectorP` – odpowiedzialna za tworzenie i zarządzanie lokalnymi wektorami obciążen dla wszystkich elementów siatki.

Element_uniwersalny:

`class dNdksi` – odpowiedzialna za obliczanie pochodnych funkcji kształtu względem ksi, w wszystkich punktach całkowania

`class dNdeteta` – odpowiedzialna za obliczanie pochodnych funkcji kształtu względem eta, w wszystkich punktach całkowania

`class NFun` – odpowiedzialna za obliczanie wartości funkcji kształtu w wszystkich punktach całkowania

`class Solver` – odpowiedzialna za rozwiązywanie układu równań macierzowych

`class Aggregation` – odpowiedzialna za globalną agregację lokalnych składowych elementów tj: H,HBC,P,C

`class Paraview` – odpowiedzialna za generowanie plików paraview

Algorytm MES z racji swojego skomplikowania, podzielony jest na kilka pomniejszych etapów:

- Wczytanie z pliku danych opisujących warunki symulacji (Global Data) oraz parametry siatki:

```
"""TWORZENIE LISTY NODEÓW"""
is_node_section = False
with open(f"{self.file_path}", "r") as file:
    for line in file:
        if line.strip() == "*Node": is_node_section = True
        elif line.strip().split(',')[0] == "*Element": is_node_section = False

        elif is_node_section and line.strip():
            parts = line.strip().split(",")
            if len(parts) == 3:
                x, y = float(parts[1].strip()), float(parts[2].strip())
                self.nodes.append(Node(x,y))

"""TWORZENIE LISTY ELEMENTÓW"""
is_element_section = False
with open(f"{self.file_path}", "r") as file:
    for line in file:
        if line.strip().split(',')[0] == "*Element": is_element_section = True
        elif line.strip() == "*BC": is_element_section = False

        elif is_element_section and line.strip():
            parts = line.strip().split(",")
            if len(parts) == 5:
                vec_4 = [int(x) for x in parts][1:]
                vec_4_points = [self.nodes[i-1] for i in vec_4]
                vec_element = Element(vec_4_points)
                vec_element.nodes_IDs = [int(x) - 1 for x in parts][1:] # Bo numerujemy od 0 w listach
                self.elements.append(vec_element)

"""GLOBAL DATA"""
def __init__(self,file_PATH: str):
    variables = {}
    self.file_path = file_PATH
    with open(f'{self.file_path}', 'r') as file:
        for i in range(10):
            line = file.readline()
            parts = line.split()
            if len(parts) == 2:
                variable_name = parts[0]
                variable_value = parts[1]
                variables[variable_name] = int(variable_value)
            if len(parts) == 3:
                variable_name = f'{parts[0]}_{parts[1]}'
                variable_value = parts[2]
                variables[variable_name] = int(variable_value)
    self.variables = variables
```

- **Obliczenie macierzy H, polegające na:**

- Obliczaniu pochodnych cząstkowych: Pochodne cząstkowe funkcji kształtu są obliczane względem zmiennych ξ i η dla każdego punktu całkowania na powierzchni elementu.

```
N1dksi = lambda eta,ksi: -0.25 * (1 - eta)
N2dksi = lambda eta,ksi: 0.25 * (1 - eta)
N3dksi = lambda eta,ksi: 0.25 * (1 + eta)
N4dksi = lambda eta,ksi: -0.25 * (1 + eta)
funkcje = [N1dksi,N2dksi,N3dksi,N4dksi]

N1deta = lambda eta,ksi: -0.25 * (1 - ksi)
N2deta = lambda eta,ksi: -0.25 * (1 + ksi)
N3deta = lambda eta,ksi: 0.25 * (1 + ksi)
N4deta = lambda eta,ksi: 0.25 * (1 - ksi)
funkcje = [N1deta, N2deta, N3deta, N4deta]
```

```
if points == 2:
    self.matrix = np.zeros((4, 4))
    for i in range(4):
        for j in range(4):
            self.matrix[i][j] = self.funkcje[i](*gauss_points["p2"][j])
    self.matrix = self.matrix.T

elif points == 3:
```

- Transformacji do układu globalnego: Pochodne cząstkowe są przekształcane do układu globalnego (x, y) za pomocą macierzy Jacobiego. Jakobian przekształcenia mierzy, jak “dużo” przestrzeni jest “rozciągane” lub “kurczone” podczas przekształcenia z jednego układu współrzędnych na inny. Dlatego, kiedy przekształcamy punkt całkowania z jednego układu na inny, musimy “uwzględnić” ten efekt. Robimy to, mnożąc wyrażenie przez Jakobian przekształcenia. W ten sposób, element objętości dV w nowym układzie współrzędnych jest równy elementowi objętości w starym układzie współrzędnych.

```
@staticmethod
def przejscie(vec_dNidksi, vec_dNideta, vec2, vec4):
    dxdksi = vec_dNidksi[0] * vec4[0].x + vec_dNidksi[1] * vec4[1].x + vec_dNidksi[2] * vec4[2].x + vec_dNidksi[3] * vec4[3].x
    dydksi = vec_dNidksi[0] * vec4[0].y + vec_dNidksi[1] * vec4[1].y + vec_dNidksi[2] * vec4[2].y + vec_dNidksi[3] * vec4[3].y
    dxdata = vec_dNideta[0] * vec4[0].x + vec_dNideta[1] * vec4[1].x + vec_dNideta[2] * vec4[2].x + vec_dNideta[3] * vec4[3].x
    dydata = vec_dNideta[0] * vec4[0].y + vec_dNideta[1] * vec4[1].y + vec_dNideta[2] * vec4[2].y + vec_dNideta[3] * vec4[3].y

    matrix = np.array([[dxdksi,dydksi],
                      [dxdata, dydata]])

    determinant = np.linalg.det(matrix)
    reverse_mat = np.array([[dydata,-dydksi],
                           [-dxdata, dxdksi]])

    wynik = (1. / determinant) * reverse_mat @ vec2.T

    return wynik, determinant
```

- Agregacja macierzy wyliczonych w punktach: po uzyskaniu macierzy cząstkowych, są on przemnażane przez odpowiednie wagi i zsumowywany w celu utworzenia macierzy H dla elementu.

```

self.H_matrix = [] # Tablica poszczególnych macierzy cząstkowych - macierzy H
for i in range(self.points ** 2):
    Hpcj = self.k * (self.matrix_dx[i] + self.matrix_dy[i]) * list_jacobian[i]
    self.H_matrix.append(Hpcj)

if self.points == 2:
    for i, w in enumerate(gauss_weights["w2"]):
        self.H_matrix[i] *= w[0] * w[1]
elif self.points == 3:
    for i, w in enumerate(gauss_weights["w3"]):
        self.H_matrix[i] *= w[0] * w[1]
elif self.points == 4:
    for i, w in enumerate(gauss_weights["w4"]):
        self.H_matrix[i] *= w[0] * w[1]
else:
    raise ("Błąd liczby punktów")

self.H_matrix = np.sum(self.H_matrix, axis=0)

```

- Powtórzenie dla wszystkich elementów: Powyższe kroki są powtarzane dla każdego elementu w układzie, a wyniki globalnie agregowane , aby uzyskać globalną macierz H (w kodzie realizowane w tym samym momencie co agregacja macierzy HBC)

```

def fill_matrix_H(self):
    for element, id_mat in zip(self.grid.elements, self.where_ids_H):
        H = element.matrix_H
        HBC = element.matrix_HBC
        for i in range(4):
            for j in range(4):
                x,y = id_mat[i][j]
                self.global_H[x][y] += H[i][j]
                self.global_H[x][y] += HBC[i][j]

```

- **Obliczenie macierzy HBC, polegające na:**

- Obliczaniu wartości funkcji kształtu: Wartości funkcji kształtu są obliczane dla każdego punktu całkowania na brzegu elementu

```
N1 = lambda ksi, eta: 0.25 * (1 - ksi) * (1 - eta)
N2 = lambda ksi, eta: 0.25 * (1 + ksi) * (1 - eta)
N3 = lambda ksi, eta: 0.25 * (1 + ksi) * (1 + eta)
N4 = lambda ksi, eta: 0.25 * (1 - ksi) * (1 + eta)
N = [N1,N2,N3,N4]
edges_list = [edges["edge_d"],edges["edge_r"],edges["edge_t"],edges["edge_l"]]

for j in range(4): # Wylicz dla konkretnego punktu funkcje kształtu
    mat[0][j] = N[j](*point)
```

- Transformacji do układu globalnego: tym razem realizowana poprzez przemnożenie przez połowę długości boku elementu. Wynika to z liniowości przekształcenia.

```
def generate_jacobi_list_for_element(self,vec_4):
    if vec_4.to_list()[0].BC == 1 and vec_4.to_list()[1].BC == 1:
        jacobi_1 = np.sqrt((vec_4.to_list()[0].x - vec_4.to_list()[1].x) ** 2 + (vec_4.to_list()[0].y - vec_4.to_list()[1].y) ** 2) / 2
    else: jacobi_1 = 0.0 # ""Nie uwzględniaj danej krawędzi w liczeniu macierzy HBC"""

    if vec_4.to_list()[1].BC == 1 and vec_4.to_list()[2].BC == 1:
        jacobi_2 = np.sqrt((vec_4.to_list()[1].x - vec_4.to_list()[2].x) ** 2 + (vec_4.to_list()[1].y - vec_4.to_list()[2].y) ** 2) / 2
    else: jacobi_2 = 0.0

    if vec_4.to_list()[2].BC == 1 and vec_4.to_list()[3].BC == 1:
        jacobi_3 = np.sqrt((vec_4.to_list()[2].x - vec_4.to_list()[3].x) ** 2 + (vec_4.to_list()[2].y - vec_4.to_list()[3].y) ** 2) / 2
    else: jacobi_3 = 0.0

    if vec_4.to_list()[3].BC == 1 and vec_4.to_list()[0].BC == 1:
        jacobi_4 = np.sqrt((vec_4.to_list()[3].x - vec_4.to_list()[0].x) ** 2 + (vec_4.to_list()[3].y - vec_4.to_list()[0].y) ** 2) / 2
    else: jacobi_4 = 0.0
```

- Agregacja macierzy wyliczonych w punktach: po uzyskaniu wyniku jest on przemnożony przez odpowiednie wagi, współczynnik przejmowania ciepła i zsumowywany w celu utworzenia macierzy HBC dla elementu.

```
if jacobi_list[k] != 0.0:
    wynik += gauss[f"w{self.points}"][i] * self.k * (mat.T @ mat) * jacobi_list[k]
warunki_brzegowe.append(wynik)
return np.sum(warunki_brzegowe, axis=0)
```

- Powtórzenie dla wszystkich elementów: Powyższe kroki są powtarzane dla każdego elementu w układzie, a wyniki globalnie agregowane , aby uzyskać globalną macierz HBC (w kodzie realizowane w tym samym momencie co agregacja macierzy H)

```
def fill_matrix_H(self):
    for element, id_mat in zip(self.grid.elements, self.where_ids_H):
        H = element.matrix_H
        HBC = element.matrix_HBC
        for i in range(4):
            for j in range(4):
                x,y = id_mat[i][j]
                self.global_H[x][y] += H[i][j]
                self.global_H[x][y] += HBC[i][j]
```

- **Obliczenie Macierzy C, polegające na:**

- Obliczaniu pochodnych cząstkowych: Podobnie jak dla macierzy H

```
if self.points == 2:
    vec_el_ksi = dNdksi(2)
    vec_el_eta = dNdet(2)
    self.vec_el_N = NFun(2)

elif self.points == 3:
```

- Transformacji do układu globalnego: Również analogicznie jak dla macierzy H

```
@staticmethod
def jacobian_calculate(vec_dNidksi,vec_dNideta,vec4):
    dxdksi = vec_dNidksi[0] * vec4[0].x + vec_dNidksi[1] * vec4[1].x + vec_dNidksi[2] * vec4[2].x + vec_dNidksi[3] * vec4[3].x
    dydksi = vec_dNidksi[0] * vec4[0].y + vec_dNidksi[1] * vec4[1].y + vec_dNidksi[2] * vec4[2].y + vec_dNidksi[3] * vec4[3].y
    dxdata = vec_dNideta[0] * vec4[0].x + vec_dNideta[1] * vec4[1].x + vec_dNideta[2] * vec4[2].x + vec_dNideta[3] * vec4[3].x
    dydata = vec_dNideta[0] * vec4[0].y + vec_dNideta[1] * vec4[1].y + vec_dNideta[2] * vec4[2].y + vec_dNideta[3] * vec4[3].y

    matrix = np.array([[dxdksi,dydksi],
                      [dxdata, dydata]])

    determinant = np.linalg.det(matrix)

    return determinant
```

- Agregacja macierzy wyliczonych w punktach: Tutaj w odróżnieniu od macierzy H, mnożymy uzyskane wyniki przez ρ i c by uwzględnić wpływ gęstości i ciepła właściwego na zdolność elementu do magazynowania energii cieplnej.

```
self.C_matrix = [] # Tablica poszczególnych macierzy częstkowych - macierz C
for i in range(self.points ** 2):
    Cpc_i = self.cp * self.rho * self.matrix_N[i] * list_jacobian[i]
    self.C_matrix.append(Cpc_i)

if self.points == 2:
    for i, w in enumerate(gauss_weights["w2"]):
        self.C_matrix[i] *= w[0] * w[1]
elif self.points == 3:
    for i, w in enumerate(gauss_weights["w3"]):
        self.C_matrix[i] *= w[0] * w[1]
elif self.points == 4:
    for i, w in enumerate(gauss_weights["w4"]):
        self.C_matrix[i] *= w[0] * w[1]
else:
    raise ("Błąd liczby punktów")

self.C_matrix = np.sum(self.C_matrix, axis=0)
```

- Powtórzenie dla wszystkich elementów: Powyższe kroki są powtarzane dla każdego elementu w układzie, a wyniki globalnie agregowane, aby uzyskać globalną macierz C:

```
def fill_matrix_C(self):
    for element, id_mat in zip(self.grid.elements, self.where_ids_C):
        C = element.matrix_C
        for i in range(4):
            for j in range(4):
                x, y = id_mat[i][j]
                self.global_C[x][y] += C[i][j]
```

- Obliczenie Wektora P, polegające na:

- Obliczaniu wartości funkcji kształtu: Podobnie jak dla macierzy HBC

```
N1 = lambda ksi, eta: 0.25 * (1 - ksi) * (1 - eta)
N2 = lambda ksi, eta: 0.25 * (1 + ksi) * (1 - eta)
N3 = lambda ksi, eta: 0.25 * (1 + ksi) * (1 + eta)
N4 = lambda ksi, eta: 0.25 * (1 - ksi) * (1 + eta)
N = [N1,N2,N3,N4]
edges_list = [edges["edge_d"],edges["edge_r"],edges["edge_t"],edges["edge_l"]]
jacobi_list = self.generate_jacobi_list_for_element(vec_4) # Jakobiany dla poszczególnych boków [uwzględniony warunek BC]
```

- Transformacji do układu globalnego: Również analogicznie jak dla macierzy HBC

```
def generate_jacobi_list_for_element(self,vec_4):
    if vec_4.to_list()[0].BC == 1 and vec_4.to_list()[1].BC == 1:
        jacobi_1 = np.sqrt((vec_4.to_list()[0].x - vec_4.to_list()[1].x) ** 2 + (vec_4.to_list()[0].y - vec_4.to_list()[1].y) ** 2) / 2
    else: jacobi_1 = 0.0 # """wyzeruje macierz - bo wtedy nie uwzględniamy danej krawędzi [ Bez tego warunku wszystkie krawędzie są zewnętrzne ]"""
    if vec_4.to_list()[1].BC == 1 and vec_4.to_list()[2].BC == 1:
        jacobi_2 = np.sqrt((vec_4.to_list()[1].x - vec_4.to_list()[2].x) ** 2 + (vec_4.to_list()[1].y - vec_4.to_list()[2].y) ** 2) / 2
    else: jacobi_2 = 0.0
    if vec_4.to_list()[2].BC == 1 and vec_4.to_list()[3].BC == 1:
        jacobi_3 = np.sqrt((vec_4.to_list()[2].x - vec_4.to_list()[3].x) ** 2 + (vec_4.to_list()[2].y - vec_4.to_list()[3].y) ** 2) / 2
    else: jacobi_3 = 0.0
    if vec_4.to_list()[3].BC == 1 and vec_4.to_list()[0].BC == 1:
        jacobi_4 = np.sqrt((vec_4.to_list()[3].x - vec_4.to_list()[0].x) ** 2 + (vec_4.to_list()[3].y - vec_4.to_list()[0].y) ** 2) / 2
    else: jacobi_4 = 0.0
```

- Agregacja macierzy wyliczonych w punktach: po uzyskaniu wyniku jest on przemnożony przez odpowiednie wagę, współczynnik przejmowania ciepła, temperaturę otoczenia i zsumowywany w celu utworzenia wektora P dla elementu.

```
    if jacobi_list[k] != 0.0:
        wynik += gauss[f"w{self.points}"][i] * self.alfa * mat * self.tot * jacobi_list[k]
    warunki_brzegowe.append(wynik)

return np.sum(warunki_brzegowe, axis=0)
```

- Powtórzenie dla wszystkich elementów: Powyższe kroki są powtarzane dla każdego elementu w układzie, a wyniki globalnie agregowane, by uzyskać globalny wektor P.

```
def fill_vector_P(self):
    for element, id_vec in zip(self.grid.elements, self.where_ids_P):
        P = element.vectorP
        for i in range(4):
            x = id_vec[0][i]
            self.global_P[x][0] += P[i]
```

- Rozwiązania układu równań, niejawnym schematem (bardziej stabilnym numerycznie, niż jawny):

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

Niejawny schemat

jawny schemat

- Wykorzystana została w tym celu biblioteka: scipy.sparse zoptymalizowana do obsługi rzadkich układów macierzowych układów równań.

```
def solve(self):
    matric_c_per_step = self.global_C / self.SimulationStepTime
    matrix_glob = self.global_H + matric_c_per_step

    T_opt = []

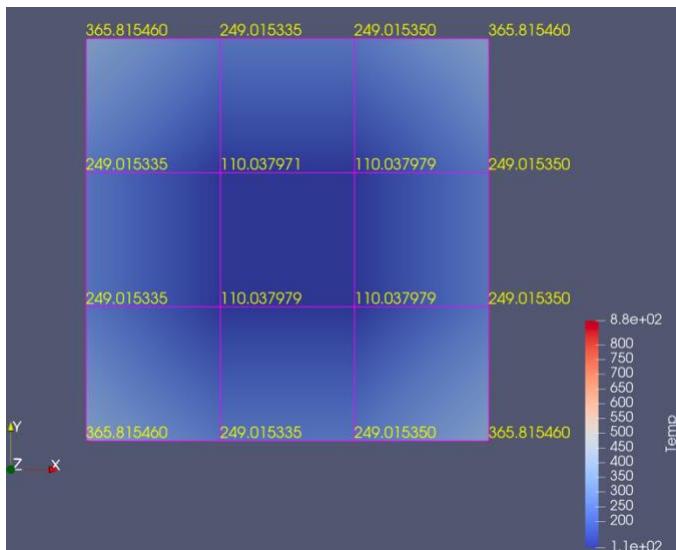
    for time in range(self.SimulationStepTime, self.SimulationTime + self.SimulationStepTime, self.SimulationStepTime):
        T1 = spsolve(csr_matrix(matrix_glob), self.global_P + (matric_c_per_step @ self.T0))

        T1 = np.array(T1).reshape(-1,1)

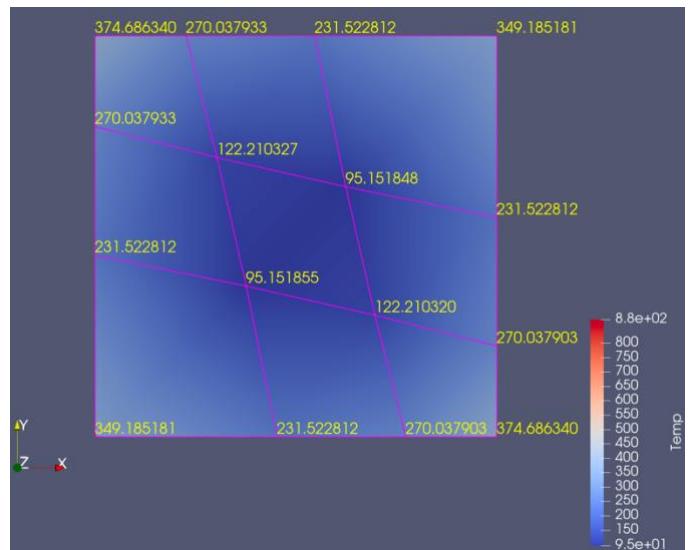
        self.print_T(T1, time)
        T_opt.append(T1)
        self.T0 = T1

    return np.array(T_opt)
```

- Wygenerowanie plików .vtk dla programu paraview i wizualnego sprawdzenia wyników:



Siatka testowa – simple (Test1.txt)



Siatka testowa – mixgrid (Test2.txt)

Porównanie wyników oprogramowania z testami

1. Test wczytywania siatki i parametrów symulacji (Siatka 4_4_mix // 2pkt schemat)

```
-----TEST_GRID-----
Test: SimulationTime: 500 , SimulationStepTime: 50
Test: Conductivity: 25 , Alfa: 300
Test: Tot: 1200 , InitialTemp: 100
Test: Density: 7800 , SpecificHeat: 700
Test: Nodes_number: 16 , Elements_number: 9
-----
1) x: 0.100000001 y: 0.0049999989 BC: 1
2) x: 0.0546918176 y: 0.0049999989 BC: 1
3) x: 0.0226540919 y: 0.0049999989 BC: 1
4) x: 0.0 y: 0.0049999989 BC: 1
5) x: 0.100000001 y: -0.0403081849 BC: 1
6) x: 0.0623899326 y: -0.0326100662 BC: 0
7) x: 0.0303522106 y: -0.0253522098 BC: 0
8) x: 0.0 y: -0.0176540911 BC: 1
9) x: 0.100000001 y: -0.072345905 BC: 1
10) x: 0.069647789 y: -0.0646477863 BC: 0
11) x: 0.0376100652 y: -0.0573899336 BC: 0
12) x: 0.0 y: -0.0496918149 BC: 1
13) x: 0.100000001 y: -0.0949999988 BC: 1
14) x: 0.0773459077 y: -0.0949999988 BC: 1
15) x: 0.0453081839 y: -0.0949999988 BC: 1
16) x: 0.0 y: -0.0949999988 BC: 1
-----
1) vec[0]: (0.100000001, 0.0049999989) , vec[3]: (0.100000001, -0.0403081849)
2) vec[0]: (0.0546918176, 0.0049999989) , vec[3]: (0.0623899326, -0.0326100662)
3) vec[0]: (0.0226540919, 0.0049999989) , vec[3]: (0.0303522106, -0.0253522098)
4) vec[0]: (0.100000001, -0.0403081849) , vec[3]: (0.100000001, -0.072345905)
5) vec[0]: (0.0623899326, -0.0326100662) , vec[3]: (0.069647789, -0.0646477863)
6) vec[0]: (0.0303522106, -0.0253522098) , vec[3]: (0.0376100652, -0.0573899336)
7) vec[0]: (0.100000001, -0.072345905) , vec[3]: (0.100000001, -0.0949999988)
8) vec[0]: (0.069647789, -0.0646477863) , vec[3]: (0.0773459077, -0.0949999988)
9) vec[0]: (0.0376100652, -0.0573899336) , vec[3]: (0.0453081839, -0.0949999988)
-----
SimulationTime 500
SimulationStepTime 50
Conductivity 25
Alfa 300
Tot 1200
InitialTemp 100
Density 7800
SpecificHeat 700
Nodes number 16
Elements number 9
*Node
1, 0.100000001, 0.0049999989
2, 0.0546918176, 0.0049999989
3, 0.0226540919, 0.0049999989
4, 0., 0.0049999989
5, 0.100000001, -0.0403081849
6, 0.0623899326, -0.0326100662
7, 0.0303522106, -0.0253522098
8, 0., -0.0176540911
9, 0.100000001, -0.072345905
10, 0.069647789, -0.0646477863
11, 0.0376100652, -0.0573899336
12, 0., -0.0496918149
13, 0.100000001, -0.0949999988
14, 0.0773459077, -0.0949999988
15, 0.0453081839, -0.0949999988
16, 0., -0.0949999988
*Element, type=DC2D4
1, 1, 2, 6, 5
2, 2, 3, 7, 6
3, 3, 4, 8, 7
4, 5, 6, 10, 9
5, 6, 7, 11, 10
6, 7, 8, 12, 11
7, 9, 10, 14, 13
8, 10, 11, 15, 14
9, 11, 12, 16, 15
*BC
1, 2, 3, 4, 5, 8, 9, 12, 13, 14, 15, 16
```

2. Test obliczeń pochodnych cząstkowych po ksi i eta (schemat 2pkt):

```
-----ELEMENT_UNIIVERSALNY_TEST-----
dNdksi: [[-0.39433757  0.39433757  0.10566243 -0.10566243]
 [-0.39433757  0.39433757  0.10566243 -0.10566243]
 [-0.10566243  0.10566243  0.39433757 -0.39433757]
 [-0.10566243  0.10566243  0.39433757 -0.39433757]]
dNdeteta: [[-0.39433757 -0.10566243  0.10566243  0.39433757]
 [-0.10566243 -0.39433757  0.39433757  0.10566243]
 [-0.39433757 -0.10566243  0.10566243  0.39433757]
 [-0.10566243 -0.39433757  0.39433757  0.10566243]]
dN/dKsi
-0.394338 0.394338 0.105662 -0.105662
-0.394338 0.394338 0.105662 -0.105662
-0.105662 0.105662 0.394338 -0.394338
-0.105662 0.105662 0.394338 -0.394338
dN/dEta
-0.394338 -0.105662 0.105662 0.394338
-0.105662 -0.394338 0.394338 0.105662
-0.394338 -0.105662 0.105662 0.394338
-0.105662 -0.394338 0.394338 0.105662
```

3 . Test obliczeń lokalnych macierzy H – (porównanie dla pierwszych 5):

```
-----MACIERZ_H_TEST-----
[[ 17.76238271 -3.39971499 -10.96295267 -3.39971506]
 [ -3.39971499 14.65084276 -5.14961161 -6.10151617]
 [-10.96295267 -5.14961161 21.26217502 -5.14961075]
 [ -3.39971506 -6.10151617 -5.14961075 14.65084198]]
[[ 20.95840051 -5.19066342 -13.27218673 -2.49555037]
 [ -5.19066342 14.13800425 -4.44552642 -4.50181441]
 [-13.27218673 -4.44552642 23.37692347 -5.65921033]
 [ -2.49555037 -4.50181441 -5.65921033 12.6565751 ]]
[[ 21.12793339 -5.49124732 -12.36735175 -3.26933431]
 [ -5.49124732 16.21364391 -5.49124826 -5.23114833]
 [-12.36735175 -5.49124826 21.12793348 -3.26933437]
 [ -3.26933431 -5.23114833 -3.26933437 11.76981611]]
[[ 20.95840224 -2.49554802 -13.27218766 -5.19066657]
 [ -2.49554802 12.65657462 -5.65921257 -4.50181403]
 [-13.27218766 -5.65921257 23.37692413 -4.4455239 ]
 [ -5.19066657 -4.50181403 -4.4455239 14.1380045 ]]
[[ 24.43980717 -4.61747587 -15.2048546 -4.6174767 ]
 [ -4.61747587 12.49999981 -4.61747545 -3.26504849]
 [-15.2048546 -4.61747545 24.43980512 -4.61747507]
 [ -4.6174767 -3.26504849 -4.61747507 12.50000025]]
```

H dla elementu - 1	
17.7624 -3.39971 -10.963 -3.39972	
-3.39971 14.6508 -5.14961 -6.10152	
-10.963 -5.14961 21.2622 -5.14961	
-3.39972 -6.10152 -5.14961 14.6508	
H dla elementu - 2	
20.9584 -5.19066 -13.2722 -2.49555	
-5.19066 14.138 -4.44553 -4.50181	
-13.2722 -4.44553 23.3769 -5.65921	
-2.49555 -4.50181 -5.65921 12.6566	
H dla elementu - 3	
21.1279 -5.49125 -12.3674 -3.26933	
-5.49125 16.2136 -5.49125 -5.23115	
-12.3674 -5.49125 21.1279 -3.26933	
-3.26933 -5.23115 -3.26933 11.7698	
H dla elementu - 4	
20.9584 -2.49555 -13.2722 -5.19067	
-2.49555 12.6566 -5.65921 -4.50181	
-13.2722 -5.65921 23.3769 -4.44552	
-5.19067 -4.50181 -4.44552 14.138	
H dla elementu - 5	
24.4398 -4.61748 -15.2049 -4.61748	
-4.61748 12.5 -4.61748 -3.26505	
-15.2049 -4.61748 24.4398 -4.61748	
-4.61748 -3.26505 -4.61748 12.5	

4. Test obliczeń lokalnych macierzy HBC – (porównanie dla pierwszych 5):

```
-----MACIERZ_HBC_TEST-----
[[9.06163682 2.26540917 0.          2.26540924]
 [2.26540917 4.53081834 0.          0.          ]
 [0.          0.          0.          0.          ]
 [2.26540924 0.          0.          4.53081848]]
[[3.20377257 1.60188629 0.          0.          ]
 [1.60188629 3.20377257 0.          0.          ]
 [0.          0.          0.          0.          ]
 [0.          0.          0.          0.          ]]
[[2.26540919 1.1327046 0.          0.          ]
 [1.1327046 4.53081829 1.13270455 0.          ]
 [0.          1.13270455 2.2654091 0.          ]
 [0.          0.          0.          0.          ]]
[[3.20377201 0.          0.          1.60188601]
 [0.          0.          0.          0.          ]
 [0.          0.          0.          0.          ]
 [1.60188601 0.          0.          3.20377201]]
```

Macierz Hbc dla elementu nt 1	
9.06164 2.26541 0 2.26541	
2.26541 4.53082 0 0	
0 0 0 0	
2.26541 0 0 4.53082	
Macierz Hbc dla elementu nt 2	
3.20377 1.60189 0 0	
1.60189 3.20377 0 0	
0 0 0 0	
0 0 0 0	
Macierz Hbc dla elementu nt 3	
2.26541 1.1327 0 0	
1.1327 4.53082 1.1327 0	
0 1.1327 2.26541 0	
0 0 0 0	
Macierz Hbc dla elementu nt 4	
3.20377 0 0 1.60189	
0 0 0 0	
0 0 0 0	
1.60189 0 0 3.20377	
Macierz Hbc dla elementu nt 5	
0 0 0 0	
0 0 0 0	
0 0 0 0	
0 0 0 0	

5. Test obliczeń lokalnych dla wektorów P:

```
-----VECTOR_P_TEST-----
[[16310.9462742 8155.473012 0. 8155.4732622]
 [5766.790626 5766.790626 0. 0. ]
 [4077.736542 8155.4729202 4077.7363782 0. ]
 [5766.789618 0. 0. 5766.789618]
 [0. 0. 0. 0. ]
 [ 0. 5766.790284 5766.790284 0. ]
 [4077.736884 0. 4077.736794 8155.473678]
 [ 0. 0. 5766.790284 5766.790284]
 [ 0. 8155.473102 16310.946204 8155.473102]
-----
```

Wektor {P} dla elementu - 1
 16310.9 8155.47 0 8155.47
 Wektor {P} dla elementu - 2
 5766.79 5766.79 0 0
 Wektor {P} dla elementu - 3
 4077.74 8155.47 4077.74 0
 Wektor {P} dla elementu - 4
 5766.79 0 0 5766.79
 Wektor {P} dla elementu - 5
 0 0 0 0
 Wektor {P} dla elementu - 6
 0 5766.79 5766.79 0
 Wektor {P} dla elementu - 7
 4077.74 0 4077.74 8155.47
 Wektor {P} dla elementu - 8
 0 0 5766.79 5766.79
 Wektor {P} dla elementu - 9
 0 8155.47 16310.9 8155.47

6. Test obliczeń lokalnych macierzy C:

```
-----MACIERZ_C_TEST-----
[[1139.58553695 543.34302896 258.44665067 543.34304086]
 [ 543.34302896 1033.78657887 490.44356182 258.44665067]
 [ 258.44665067 490.44356182 927.9876684 490.44357372]
 [ 543.34304086 258.44665067 490.44357372 1033.78662648]]
[[687.25729177 325.99548394 160.87926377 339.3916895 ]
 [325.99548394 616.72464397 304.1253656 160.87926377]
 [160.87926377 304.1253656 599.77681842 317.52157116]
 [339.3916895 160.87926377 317.52157116 670.30946621]]
[[417.1450592 195.34765972 104.28626453 221.79739895]
 [195.34765972 364.24557969 195.34765919 104.28626453]
 [104.28626453 195.34765919 417.14505708 221.79739841]
 [221.79739895 104.28626453 221.79739841 470.04453658]]
[[687.25723209 339.39166817 160.87925465 325.99545718]
 [339.39166817 670.3094406 317.52156143 160.87925465]
 [160.87925465 317.52156143 599.77680513 304.12535044]
 [325.99545718 160.87925465 304.12535044 616.72459662]]
[[590.73508105 295.3675505 147.68377846 295.36754693]
 [295.3675505 590.73512097 295.36756689 147.68377846]
 [147.68377846 295.36756689 590.73514661 295.36756332]
 [295.36754693 147.68377846 295.36756332 590.73510669]]
```

C dla elementu - 1
 1139.59 543.343 258.447 543.343
 543.343 1033.79 490.444 258.447
 258.447 490.444 927.988 490.444
 543.343 258.447 490.444 1033.79

 C dla elementu - 2
 687.257 325.995 160.879 339.392
 325.995 616.725 304.125 160.879
 160.879 304.125 599.777 317.522
 339.392 160.879 317.522 670.309

 C dla elementu - 3
 417.145 195.348 104.286 221.797
 195.348 364.246 195.348 104.286
 104.286 195.348 417.145 221.797
 221.797 104.286 221.797 470.045

 C dla elementu - 4
 687.257 339.392 160.879 325.995
 339.392 670.309 317.522 160.879
 160.879 317.522 599.777 304.125
 325.995 160.879 304.125 616.725

 C dla elementu - 5
 590.735 295.368 147.684 295.368
 295.368 590.735 295.368 147.684
 147.684 295.368 590.735 295.368
 295.368 147.684 295.368 590.735

7. Test obliczeń globalnej Macierzy H (H_local + HBC_local):

```
MatrixH
[[ 26.82401953 -1.13430582 0. 0. -1.13430582 -10.96295267 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ -1.13430582 43.34383419 -3.58877714 0. -6.10151617 -7.64516197 -13.27218673 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 0. -3.58877714 40.7351194 -4.35854273 0. -4.50181441 -7.71486073 -12.36735175 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 0. 0. -4.35854273 28.7444622 0. 0. -5.23114833 -4.35854371 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ -1.13430582 -6.10151617 0. 0. 43.3438347 -7.64515876 0. 0. -3.58878056 -13.27218766 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ -10.96295267 -7.64516197 -4.50181441 0. -7.64515876 71.01513191 -10.2766862 0. -4.50181403 -10.27668927 -15.2048546 0. 0. 0. 0. 0. 0. 0. ]
[ 0. -13.27218673 -7.71486073 -5.23114833 0. -10.2766862 71.02366242 -7.71485936 0. -3.26504849 -10.27668552 -13.27218706 0. 0. 0. 0. 0. 0. ]
[ 0. 0. -12.36735175 -4.35854371 0. 0. -7.71485936 40.73511927 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 0. 0. 0. 0. -3.58878056 -4.50181403 0. 0. 40.73511887 -7.71485807 0. 0. -4.35854273 -12.36735139 0. 0. 0. 0. 0. ]
[ 0. 0. 0. 0. -13.27218766 -10.27668927 -3.26504849 0. -7.71485807 71.02366272 -10.27668542 0. -5.23114854 -7.71485883 -13.27218644 0. 0. ]
[ 0. 0. 0. 0. -15.2048546 -10.27668552 -4.5018142 0. -10.27668542 71.01513152 -7.64516194 0. -4.5018146 -7.64516177 -10.96295347 ]
[ 0. 0. 0. 0. 0. 0. -13.27218706 -3.58877803 0. 0. -4.35854273 -5.23114854 0. 0. 28.7444626 -4.35854326 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 0. -12.36735139 -7.71485883 -4.5018146 0. -4.35854326 40.73511911 -3.58877846 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 0. -13.27218644 -7.64516177 -6.10151559 0. -3.58877846 43.34383403 -1.13430562 ]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. -10.96295347 -1.13430562 0. 0. 0. -1.13430562 26.82401988]]
```

8. Test obliczeń globalnego Wektora P:

```
VectorP
[[16310.9462742]
[13922.263638]
[ 9844.527168]
[ 8155.4729202]
[13922.2628802]
[ 0.]
[ 0.]
[ 9844.5266622]
[ 9844.526502]
[ 0.]
[ 0.]
[13922.263386]
[ 8155.473678]
[ 9844.527078]
[13922.263386]
[16310.946204]]
```

9. Test obliczeń globalnej Macierzy C:

```
MatrixC
[[1139.58553695 543.34302896 0. 0. 543.34304886 258.44665067 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 543.34302896 1721.04387064 325.99548394 0. 258.44665067 829.83525131 160.87926377 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 0. 325.99548394 1033.86978317 195.34765972 0. 160.87926377 525.92276454 104.28626453 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 0. 0. 195.34765972 364.24557969 0. 0. 104.28626453 195.34765919 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 543.34304086 258.44665067 0. 0. 1721.04385857 829.83524189 0. 0. 325.99545718 160.87925465 0. 0. 0. 0. 0. 0. 0. 0. ]
[ 258.44665067 829.83525131 160.87926377 0. 829.83524189 289.34165626 612.88912166 0. 160.87925465 612.88910837 147.68377846 0. 0. 0. 0. 0. 0. ]
[ 0. 160.87926377 525.92276454 104.28626453 0. 612.88912166 2260.33332365 525.92276908 0. 147.68377846 612.88914727 160.8792636 0. 0. 0. 0. 0. ]
[ 0. 0. 104.28626453 195.34765919 0. 0. 525.92276908 1033.86969204 0. 0. 160.8792636 325.99547401 0. 0. 0. 0. 0. 0. ]
[ 0. 0. 0. 0. 325.99545718 160.87925465 0. 0. 1033.86972303 525.92278434 0. 0. 195.34769222 104.28628146 0. 0. 0. ]
[ 0. 0. 0. 0. 160.87925465 612.88910837 147.68377846 0. 525.92278434 2260.33346304 612.88915626 0. 104.28628146 525.92282834 160.87926932 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 147.68377846 612.88914727 160.8792636 0. 612.88915626 289.34169287 829.83521492 0. 160.87926932 829.83522175 258.4466355 ]
[ 0. 0. 0. 0. 0. 0. 0. 0. 195.34769222 104.28628146 0. 0. 829.83521492 1721.0438031 0. 0. 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 0. 0. 104.28628146 525.92282034 160.87926932 0. 195.34769193 0. 0. 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 160.87926932 829.83522175 258.4466355 0. 1033.86979013 325.99548436 0. 0. ]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 258.4466355 543.34301082 0. 0. 0. 543.34301082 1139.58550126]]
```

10. Test max/min temperatury w węzłach siatki dla poszczególnych kroków czasowych:

```

VectorT // time=50
min: [95.15184899723258] ; max: [374.68633318838175]

VectorT // time=100
min: [147.64441907368297] ; max: [505.9681112788898]

VectorT // time=150
min: [220.16445575098794] ; max: [586.9978492792005]

VectorT // time=200
min: [296.73643835297565] ; max: [647.2855822356539]

VectorT // time=250
min: [370.96827241893254] ; max: [697.3339844638065]

VectorT // time=300
min: [440.5601435998324] ; max: [741.2191101272799]

VectorT // time=350
min: [504.8912021091295] ; max: [781.2095685593976]

VectorT // time=400
min: [564.0015163225324] ; max: [817.39150466240944]

VectorT // time=450
min: [618.1738632500773] ; max: [850.2373167651089]

VectorT // time=500
min: [667.765556911745] ; max: [880.1676019293311]

```

Temperature results

```

Time = 50 min_T= 95.152, max_T = 374.69
Time = 100 min_T= 147.64, max_T = 505.97
Time = 150 min_T= 220.16, max_T = 587
Time = 200 min_T= 296.74, max_T = 647.29
Time = 250 min_T= 370.97, max_T = 697.33
Time = 300 min_T= 440.56, max_T = 741.22
Time = 350 min_T= 504.89, max_T = 781.21
Time = 400 min_T= 564, max_T = 817.39
Time = 450 min_T= 618.17, max_T = 850.24
Time = 500 min_T= 667.77, max_T = 880.17

```

11. Test wydajności – pomiar czasu wykonania:

```

t1 = time.time()
matrix_H = Matrix_H(points, path) # Uzyskaj macierze sztywności H dla wszystkich elementów
matrix_HBC = Matrix_HBC(points, grid, global_data) # Uzyskaj macierze HBC dla wszystkich elementów
matrix_vecP = VectorP(points, grid, global_data) # Uzyskaj wektor P dla wszystkich elementów
matrix_C = Matrix_C(points, path) # Uzyskaj macierz C dla wszystkich elementów

for i, element in enumerate(grid.elements):
    element.matrix_H = matrix_H.get_H_matrices()[i] # Dodaj macierz H do elementów
    element.matrix_HBC = matrix_HBC.get_HBC_matrices()[i] # Dodaj macierz HBC do elementów
    element.vectorP = matrix_vecP.get_vectorP_matrices()[i] # Dodaj VectorP do elementów
    element.matrix_C = matrix_C.get_C_matrices()[i] # Dodaj Macierz C do elementów

aggregate = Aggregation(grid, global_data) # Klasa do agregacji macierzy
# aggregate.test_H_global() # Test złożenia macierzy H_global
# aggregate.test_P_global() # Test złożenia wektora P_global
# aggregate.test_C_global() # Test złożenia macierzy C_global

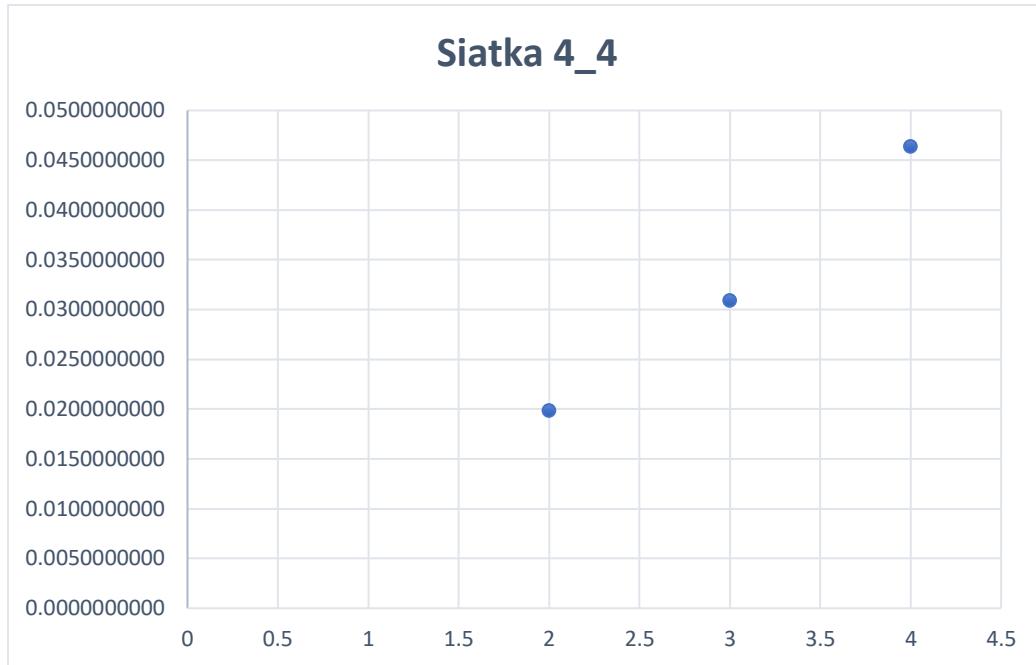
solver = Solver(global_data,aggregate.global_H,aggregate.global_P,aggregate.global_C) # Klasa do rozwiązywania układu równań
t_opt = solver.solve() # Rozwiązywanie układu równań

t2 = time.time()

```

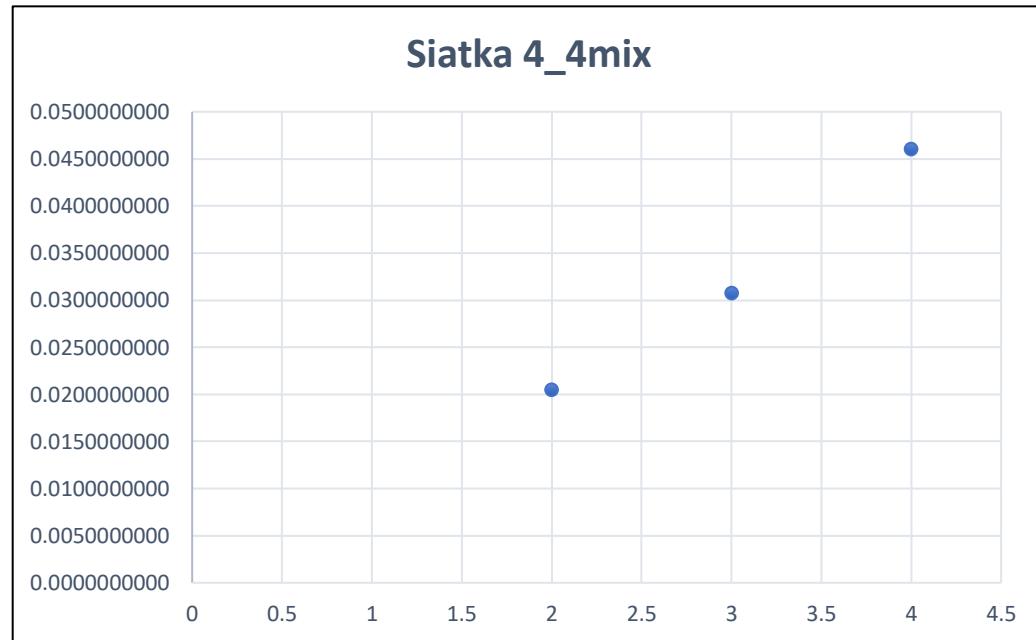
- **Siatka 4_4:**

- **2 pkt**: Czas obliczeń: 0.01986980438232422 s
- **3 pkt**: Czas obliczeń: 0.03090214729309082 s
- **4 pkt**: Czas obliczeń: 0.04640698432922363 s



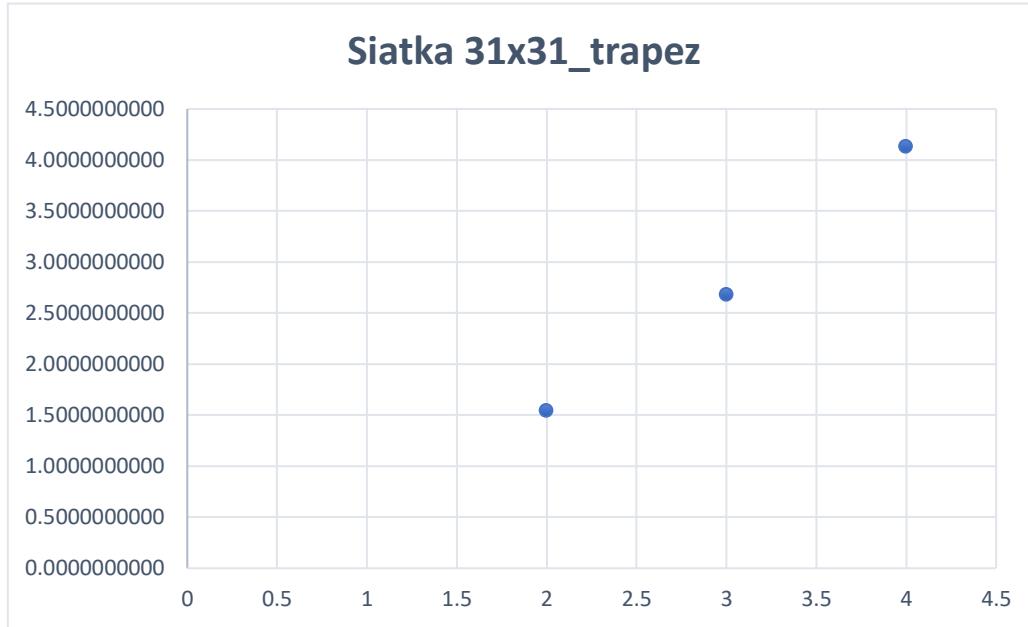
- **Siatka 4_4mix:**

- **2 pkt**: Czas obliczeń: 0.02043008804321289 s
- **3 pkt**: Czas obliczeń: 0.030714988708496094 s
- **4 pkt**: Czas obliczeń: 0.046035051345825195 s



- **Siatka 31x31 trapez:**

- **2 pkt**: Czas obliczeń: 1.5451130867004395 s
- **3 pkt**: Czas obliczeń: 2.6795828342437744 s
- **4 pkt**: Czas obliczeń: 4.135563135147095 s



Wnioski

- W napisanym programie wzrost liczby punktów całkowania powoduje (nieznaczne) zwiększenie dokładności obliczeń, lecz odbywa się to kosztem ich czasu:

```

VectorT // time=50
min: [95.15184899723258] ; max: [374.68633318838175]

VectorT // time=100
min: [147.64441987368297] ; max: [505.9681112788898]

VectorT // time=150
min: [220.16445575098794] ; max: [586.9978492792005]

VectorT // time=200
min: [296.73643835297565] ; max: [647.2855822356539]

VectorT // time=250
min: [370.96827241893254] ; max: [697.3339844638065]

VectorT // time=300
min: [440.5601435998324] ; max: [741.2191101272799]

VectorT // time=350
min: [504.8912021091295] ; max: [781.2095685593976]

VectorT // time=400
min: [564.0015163225324] ; max: [817.3915046240944]

VectorT // time=450
min: [618.1738632590773] ; max: [850.2373167651089]

VectorT // time=500
min: [667.765556911745] ; max: [888.1676019293311]

```

Schemat całkowania 2 pkt

```

VectorT // time=50
min: [95.15907045805812] ; max: [374.6682653076116]

VectorT // time=100
min: [147.65589676737176] ; max: [505.9542555019275]

VectorT // time=150
min: [220.1781130523986] ; max: [586.9894190714291]

VectorT // time=200
min: [296.75086744900926] ; max: [647.2801115566524]

VectorT // time=250
min: [370.9826353113169] ; max: [697.3298658653003]

VectorT // time=300
min: [440.57408633251024] ; max: [741.2156477930183]

VectorT // time=350
min: [504.9043692131344] ; max: [781.2408553797723]

VectorT // time=400
min: [564.0139181296034] ; max: [817.4205105019431]

VectorT // time=450
min: [618.185492013078] ; max: [850.264110038319]

VectorT // time=500
min: [667.776433784155] ; max: [888.1923022343236]

```

Schemat całkowania 4 pkt

Dla siatki 4_4_mix różnice w uzyskanych wynikach, są rzędu 0.01, przy ponad 2 razy dłuższym czasie obliczeń.

Dla większej siatki, różnice w uzyskiwanych wynikach stają się jeszcze mniejsze, a czas wykonania drastycznie (ponad 2.5 razy co jest już zauważalne przez użytkownika).

```
-----_VectorT // time=20-----
min: [100.064319869904] ; max: [341.0846585343215]
```

Schemat całkowania 2 pkt

```
-----_VectorT // time=20-----
min: [100.06431986990401] ; max: [341.08465853432136]
```

Schemat całkowania 4 pkt

W związku z tym, konieczne jest dobranie odpowiedniego schematu, w zależności od potrzeb (dla obliczeń webowych lepiej postawić na szybkość (kosztem dokładności), zaś do obliczeń krytycznych (np. w przemyśle lotniczym) lepiej postawić na optymalniejsze rozwiązanie.

- Uzyskane czasy obliczeń są względnie niewielkie, co zostało osiągnięte, dzięki użyciu biblioteki sparse, która jak już wcześniej wspomniałem optymalizuje czas obliczeń na macierzach rzadkich.

Symulacja (siatka 4_4_mix):

