Metody Elementów Skończonych

Sprawozdanie wykonał Mikita Shmialiou

Temat: Symulacja ustalonych & nieustalonych procesów cieplnych

Symulacja ustalonych procesów cieplnych

Wstęp

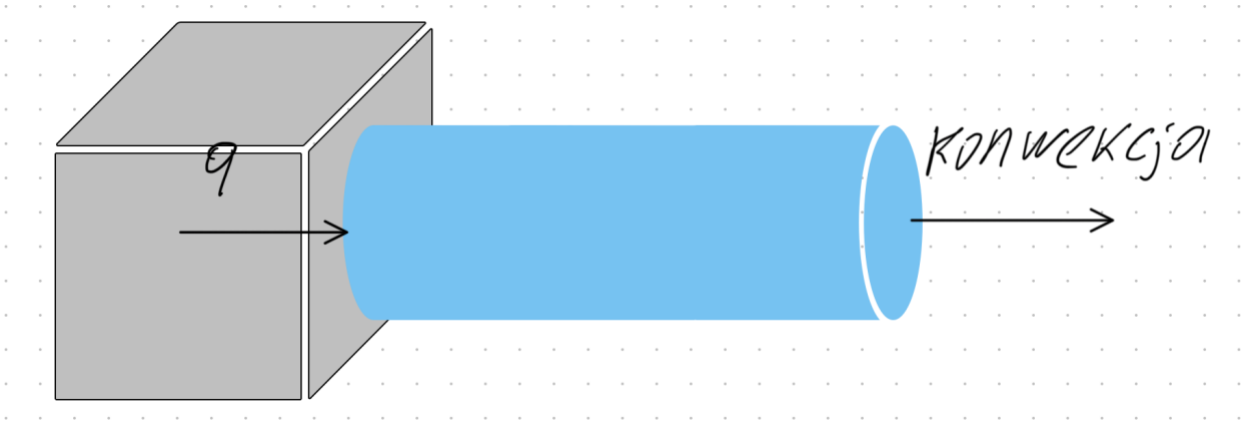
Celem jest obliczenia rozkładu temperatury w jednowymiarowym pręcie. Zadanie będzie rozwiązane na kilka sposobów:

1. Analityczne rozwiązanie układu równań oraz napisanie kodu w Pythonie na podstawie tego rozwiązania
2. przez bezpośrednią minimalizację funkcjonału w programie Excel

Przedstawienie modelu z warunkami brzegowymi

Analiza odywa się na pręcie o długości L przy procesie ustalonego przewodnictwa ciepła. Zakładamy, że wymiana ciepła odbywa się tylko na końcach pręta:

1. na początku pręta mamy strumień ciepła q
2. na końcu mamy konwekcję



Podstawy rozwiązania MES dla problemu optymalizacji bezpośredniej

Rozwiązanie polega na poszukiwaniu minimum funkcjonału energetycznego J. Przy jednowymiarowym, ustalonym przepływie ciepła oraz z warunkami brzegowymi J wygląda:

$$J = \int_V \left[\frac{1}{2} \left(k_x(t) \left(\frac{\partial t}{\partial x} \right)^2 + k_y(t) \left(\frac{\partial t}{\partial y} \right)^2 + k_z(t) \left(\frac{\partial t}{\partial z} \right)^2 \right) - 2Qt \right] dV$$

Dane weściowe:

$k = 50\frac{W}{mK}$; $\alpha = 10\frac{W}{m^2K}$;$S = 2\ m^2$;$L = 5\ m$; $L^{(1)} = L^{(2)} = 2.5\ m$; $q = -150\frac{W}{m^2}$; $t_\infty = 400\ K$

Wyniki MES dla problemu rozwiązywanego Excelem

3-węzlowy element:

$$C^{(1)} = \frac{Sk}{L^{(1)}} = \frac{2m^2 \cdot 50\frac{W}{mK}}{2{,}5m} = 40\frac{W}{K} = C^{(2)};$$

$$\alpha S = 10\frac{W}{m^2K} \cdot 2m^2 = 20\frac{W}{K};$$

$$qS = -150\frac{W}{m^2} \cdot 2m^2 = -300W;$$

$$\alpha S t_\infty = 10\frac{W}{m^2K} \cdot 2m^2 \cdot 40K = 8000W.$$

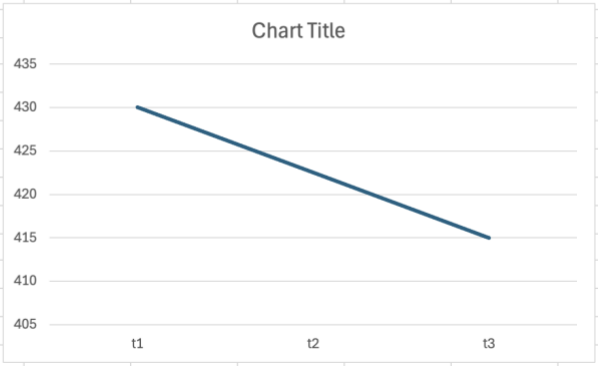
Stąd, otrzymuje się następujący układ równań:

$$\begin{bmatrix} 40 & -40 & 0 \\ -40 & 80 & -40 \\ 0 & -40 & 60 \end{bmatrix} \begin{Bmatrix} t_1 \\ t_2 \\ t_3 \end{Bmatrix} + \begin{Bmatrix} -300 \\ 0 \\ -8000 \end{Bmatrix} = 0$$

Po jego rozwiązaniu względem t uzyskano {t}={430, 422.5, 415}.

$$J = \frac{C^{(1)}}{2}(t_1^2 - 2t_1t_2 + t_2^2) + \frac{C^{(2)}}{2}(t_2^2 - 2t_2t_3 + t_3^2) + qSt_1 + \frac{\alpha S_3}{2}(t_3^2 - 2t_3t_\infty + t_\infty^2)$$

C=	40
Alfa_S=	20
qS=	-300
tsr=	400
J1=	1125.007683
J2=	1124.986764
J3=	-129000
J4=	2250.005208
J=	-124500
t1	429.9999989
t2	422.4999732
t3	415.0000174



Wyniki w EXCELu są bardzo bliskie do tych, co były otrzymane analitycznie, co świadczy o tym, że znalezienie temperatur za pomocą SOLVER'a (minizacja) odbyło się poprawnie.

5-węzlowy element

Dla obliczenia 5 węzłów trzeba delikatnie zmodyfikować plik EXEL'owy. Żeby osiągnąć ten cel trzeba podzielić nasz pręt o długość L na 4 elementy skończony. W rezultacie ilość węzłów będzie wynosiła 5. Trzeba rozszerzyć poprzednie równania do 5 węzłów.

Dane wejściowe, które uległy zmianie:

$L^{(1)} = L^{(2)} = L^{(3)} = L^{(4)} = 1.25\ m;$

$C^{(1)} = \frac{Sk}{L^{(1)}} = \frac{2m^2 \cdot 50 \frac{W}{mK}}{1{,}25m} = 80 \frac{W}{K} = C^{(2)};$

Globalna macierz sztywności dla 5 węzłów (5x5):

$$[H] = \begin{bmatrix} C & -C & 0 & 0 & 0 \\ -C & C+C & -C & 0 & 0 \\ 0 & -C & C+C & -C & 0 \\ 0 & 0 & -C & C+C & -C \\ 0 & 0 & 0 & -C & C \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha S \end{bmatrix}$$

[alfaS w ostatnim węźle to konwekcja]

$$[H] = \begin{bmatrix} 80 & -80 & 0 & 0 & 0 \\ -80 & 160 & -80 & 0 & 0 \\ 0 & -80 & 160 & -80 & 0 \\ 0 & 0 & -80 & 160 & -80 \\ 0 & 0 & 0 & -80 & 100 \end{bmatrix}$$

Wektor obciążeń:

$$\{P\} = \begin{Bmatrix} -300 \\ 0 \\ 0 \\ 0 \\ -8000 \end{Bmatrix}$$

Ostateczny układ równań dla 5 węzłów:

$$\begin{bmatrix} 80 & -80 & 0 & 0 & 0 \\ -80 & 160 & -80 & 0 & 0 \\ 0 & -80 & 160 & -80 & 0 \\ 0 & 0 & -80 & 160 & -80 \\ 0 & 0 & 0 & -80 & 100 \end{bmatrix} \begin{Bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{Bmatrix} + \begin{Bmatrix} -300 \\ 0 \\ 0 \\ 0 \\ -8000 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Z tego równania wynika, że temperatury są równe:

$t_1 = 430$

$t_2 = 426.25$

$t_3 = 422.5$

$t_4 = 418.75$

$t_5 = 415$

Obliczenie w EXCELu

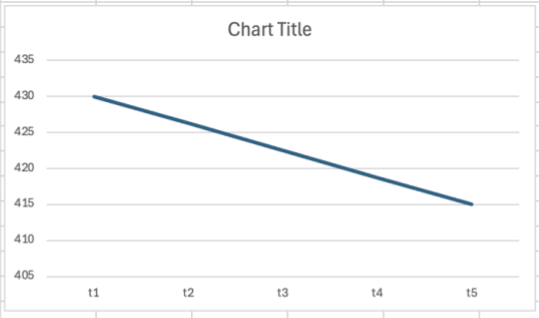
Wzór J dla 5-węzłowego układu:

$$J = \frac{C^{(1)}}{2}(t_1^2 - 2t_1t_2 + t_2^2) + \frac{C^{(2)}}{2}(t_2^2 - 2t_2t_3 + t_3^2) + \frac{C^{(3)}}{2}(t_3^2 - 2t_3t_4 + t_4^2) + \frac{C^{(4)}}{2}(t_4^2 - 2t_4t_5 + t_5^2) + qSt_1 + \frac{\alpha S_5}{2}(t_5^2 - 2t_5t_\infty + t_\infty^2)$$

C=	80
Alfa_S=	20
qS=	-300
tsr=	400

J1=	562.4359114
J2=	562.3076993
J3=	562.1791862
J4=	562.0477306
J5=	-128996.767
J6=	2247.797102
J=	-124499.999

t1	429.9892231
t2	426.2394368
t3	422.4900778
t4	418.7411473
t5	414.9926552



Wyniki w EXCELu są bardzo bliskie do tych, co były otrzymane analitycznie, co świadczy o tym, że znalezienie temperatur za pomocą SOLVER'a (minizacja) odbyło się poprawnie.

Porównanie z kodem

Kod:

main.py 2

```python
import numpy as np
from element import Element

# dane
k = 50 # W/mk
alpha = 10 # W/m2K
S = 2 # m^2
LG = 5 # m - długość
q = -150 # W/m^2
t_sr = 400 # K
ME = 4 # liczba elemetów
MN = ME + 1 # licza węzłów
delta_L = LG / ME
dir = 1 # 1 from left to right; -1 from right to left

# start
final_H = np.zeros(
    (MN, MN)
) # macież o rozmiarze ilość węzłów x ilość węzłów
final_P = np.zeros(
    (MN, 1)
)

elemnts = []

for i in range(ME):
    element: Element  = Element(
        length=delta_L,
        S=S,
        k=50,
        alpha=alpha,
        q=q,
        t_sr=t_sr,
        is_start = (i == 0),
        is_end = (i == ME-1)
    )
    elemnts.append(
        element
    )

for i in range(ME):
    element: Element = elemnts[i]
    matrix_h = element.H
    matrix_P = element.P
    for row_ind, row in enumerate(matrix_h):
        for col_ind, val in enumerate(row):
            final_H[i + row_ind][i + col_ind] += val

    for col_ind, val in enumerate(matrix_P):
        final_P[i + col_ind][0] += val[0] # val should has only 1 element

Element.print_equation(
        final_H,
        final_P
    )

res = Element.solve_equation(
    final_H,
    final_P,
)

print(res)
```

element.py 6

```python
import numpy as np
from typing import Optional

class Element:
    def __init__(
            self, length: float, S: float, k: float,
            alpha: float,
            q: float,
            t_sr: float,
            dir: int = 1,
            is_start: bool = False,
            is_end: bool = False,
    ) -> None:
        self.is_start: bool = is_start
        self.is_end: bool = is_end
        self.length: float = length
        self.q: float = q
        self.t_sr: float = t_sr
        self.S: float = S
        self.k: float = k
        self.alpha: float = alpha
        self.dir: int = dir  # 1 from left to right; -1 from right to left

        if not self.is_start:
            # we use it only in case of the first element
            self.q = 0
        if not self.is_end:
            # we use it only in case of the last element
            self.alpha = 0

        self.addon: Optional[any] = None
        self.C: float = self._calculate_C()
        self.H: np.ndarray = self._calculate_H()
        self.P: np.ndarray = self._calculate_P()

    def _calculate_C(self) -> float:
        return self.S * self.k / self.length

    def _calculate_P(self) -> np.ndarray:
        return np.array(
            [
                [
                    self.q * self.S
                    # != 0 only when element is FIRST
                ],
                [
                    -self.t_sr * self.alpha * self.S
                    # != 0 only when element is LAST
                ]
            ]
        )

    def _calculate_H(self) -> np.ndarray:
        # take a look later
        conv: float = self.alpha * self.S
        el_00: float = self.C + (conv if self.dir == -1 else 0)
        el_11: float = self.C + (conv if self.dir == 1 else 0)
        return np.array(
            [
                [el_00, -self.C],
                [-self.C, el_11]
            ]
        )

    @staticmethod
    def print_equation(H_lines: np.ndarray, P_lines: np.ndarray) -> None:
        H_lines = np.array2string(H_lines, separator=' ').splitlines()
        P_lines = np.array2string(P_lines, separator=' ').splitlines()

        # Zip and print
        t_ind: int = 1
        for H_line, P_line in zip(H_lines, P_lines):
            print(f"{H_line} |t_{t_ind}|   + {P_line}")
            t_ind += 1

    def print(self) -> None:
        H_lines: list[str] = np.array2string(self.H, separator=' ').splitlines()
        P_lines: list[str] = np.array2string(self.P, separator=' ').splitlines()

        # Zip and print
        for H_line, P_line in zip(H_lines, P_lines):
            print(f"{H_line}   {P_line}")

    @staticmethod
    def solve_equation(H: np.ndarray, P: np.ndarray) -> np.ndarray:
        # Ensure that H is invertible (non-singular)
        if np.linalg.det(H) == 0:
            raise ValueError("Matrix H is singular and cannot be inverted.")

        # Solve for t
        t = -np.linalg.inv(H) @ P
        return t
```

(Ważne – zmiana zmiennej dir nie jest poprawnie uwzględniona, czyli wyniki będę złe)

```
➜  lab1 git:(master) ✗ echo "3 węzły"; python main.py
3 węzły
[[ 40. -40.   0.] |t_1|   + [[ -300.]
 [-40.  80. -40.] |t_2|   +  [    0.]
 [  0. -40.  60.]] |t_3|   +  [-8000.]]
[[430. ]
 [422.5]
 [415. ]]
```

Wyniki odpalania programu dla 3 i 5 węzłów

```
➜  lab1 git:(master) ✗ echo "5 węzłów"; python main.py
5 węzłów
[[ 80. -80.   0.   0.   0.] |t_1|   + [[ -300.]
 [-80. 160. -80.   0.   0.] |t_2|   +  [    0.]
 [  0. -80. 160. -80.   0.] |t_3|   +  [    0.]
 [  0.   0. -80. 160. -80.] |t_4|   +  [    0.]
 [  0.   0.   0. -80. 100.]] |t_5|   +  [-8000.]]
[[430.  ]
 [426.25]
 [422.5 ]
 [418.75]
 [415.  ]]
```

Jak widać wyniki są identycznie wynikom otrzymanym analitycznie

Porównanie wyników:

3 węzły

	Kod	Excel	$\Delta t = kod - excel$
t_1	430	429.9999989	1.14895E-06
t_2	422.5	422.4999732	2.67601E-05
t_3	415	415.0000174	-1.7359E-05

5 węzłów

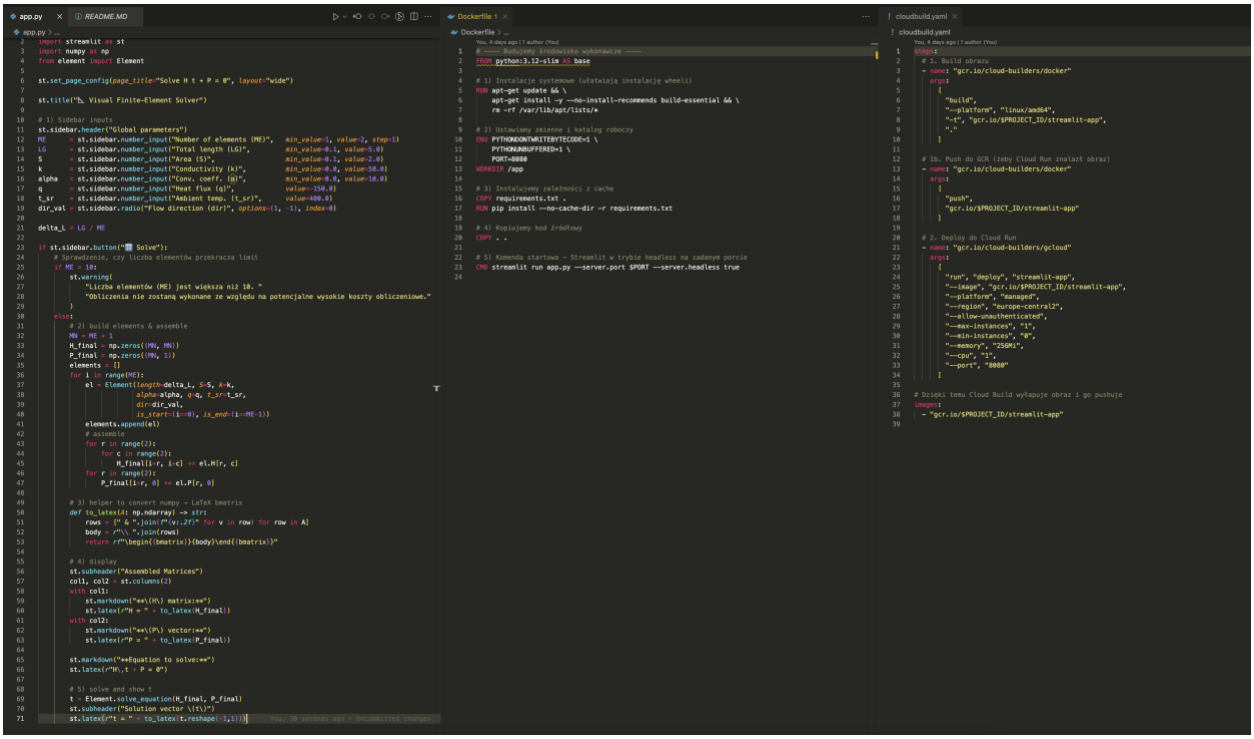
	Kod	Excel	$\Delta t = kod - excel$
t_1	430	429.9892231	1.077688E-02
t_2	426.25	426.2394368	1.056325E-02
t_3	422.5	422.4900778	9.922192E-03
t_4	418.74	418.7411473	-1.147340E-03
t_5	415	414.9926552	7.344792E-03

Różnice między wynikami z kodu i EXCEL'a są niewielkie i wynikają z dokładności narzędzia SOLVER.

Code Addon

Po sprawdzaniu poprawności działania kodu dla obliczenia ustalonego procesu powstał pomysł o zrobieniu wizualizacji do kodu. Wizualizacja polega na korzystaniu z dodatkowych bibliotek, żeby stworzyć graficzny widok do zaprezentowania wyników oraz ułatwienia korzystania z stworzonego narzędzia.

Kod:



Ten kod był wygenerowany za pomocą ChatGPT na podstawie kodu, który był umieszczony wcześniej i służy wyłącznie do wizualizacji.

„Projekt" składa się z 3 plików:

1. app.py – wizualizacja + logika z pliku main.py
2. Dockerfile
3. Cloudbuild.yaml

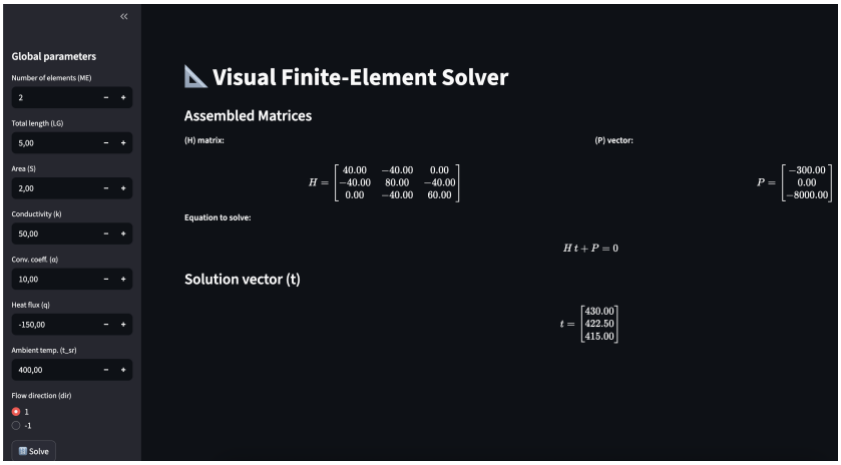
Pliki Dockerfile oraz Cloudbuild.yaml są potrzebne dla możliwości umieszczenia tego projektu w chmurze GCP, żeby dodać możliwość zaprezentowania tego programu wielu ludziom.

Aplikacja będzie dostępna w chmurze pod następnym linkiem (do 29/06/2025):

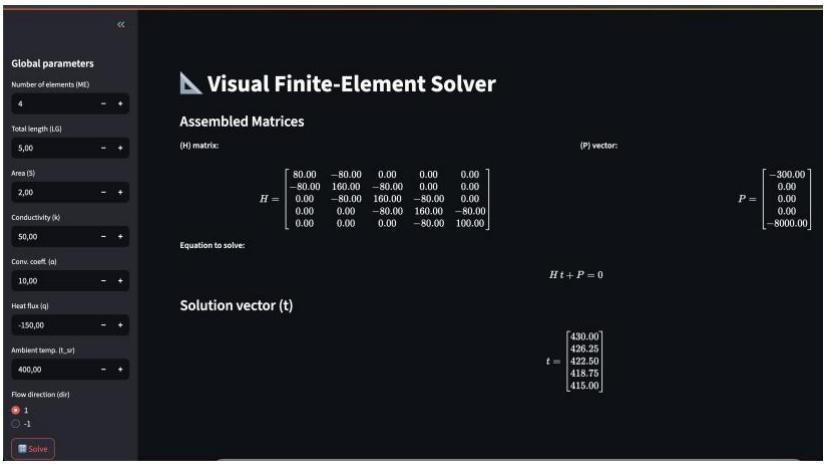
https://streamlit-app-83611701832.europe-central2.run.app/

Demo

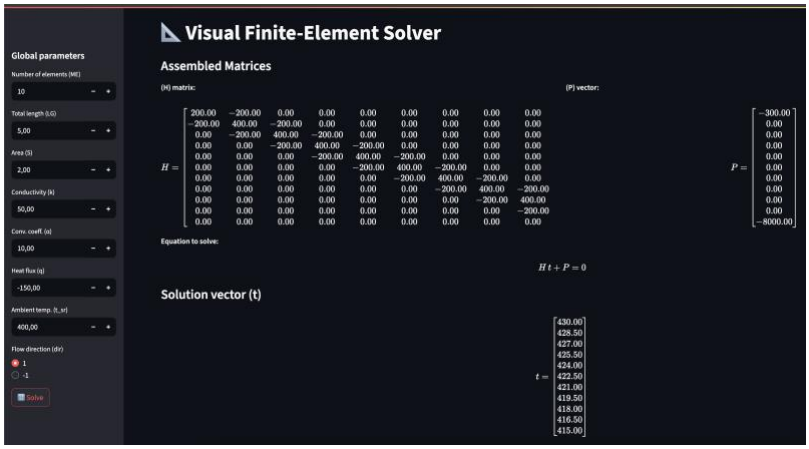
2 elementy



4 elementy



10 elementów



Podsumowanie i wnioski

W ramach sprawozdania była zrobiona symulacja ustalonego przepływu ciepła w pręcie za pomocą MES. Zastosowane rozwiązania:

1. Rozwiązanie analityczne (układ równań)
2. Minimalizacja funkcjonału za pomocą SOLVER'a
3. Rozwiązanie własnym programem w Pythonie

Wszystkie metody dały zbliżone wyniki, co potwierdza poprawność modeli. Zwiększenie liczby węzłów daje nam bardziej szczególny rozkład temperatury, trzymając się przy tym zgodności wartości na brzegach. Stworzenie aplikacji webowej nie było wymagane, ale to stanowi praktyczne rozszerzenie „projektu" oraz ułatwia prezentację oraz analizę wyników.