Metody Elementów Skończonych
Sprawozdanie wykonał **Mikita Shmialiou**
Temat: Symulacja ustalonych & nieustalonych procesów cieplnych

Symulacja ustalonych procesów cieplnych

Wstęp

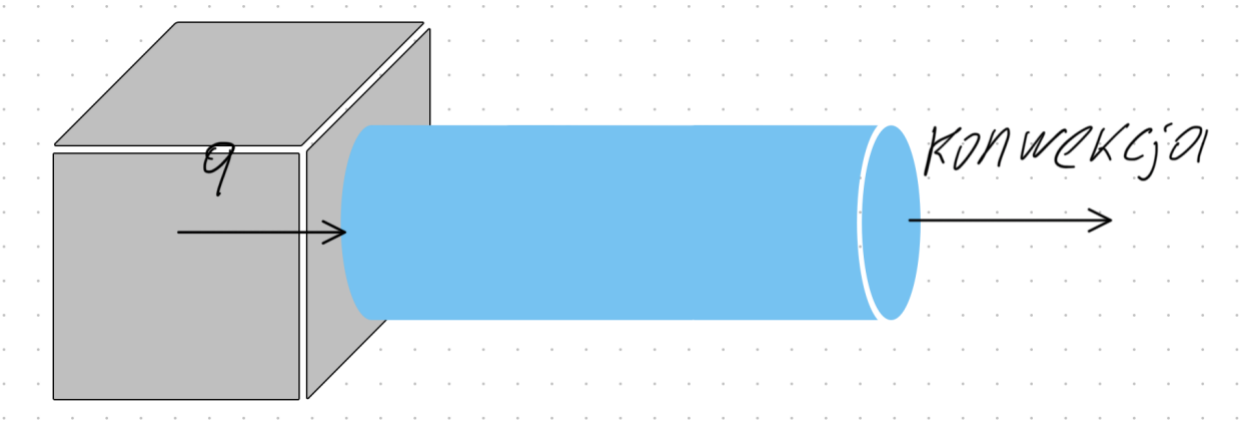
Celem tej części sprawozdania jest obliczenie rozkładu temperatury w jednowymiarowym pręcie. Zadanie będzie rozwiązane na kilka sposobów:

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 odbywa 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 wejściowe:

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

Wyniki MES dla problemu rozwiązywanego Excelem

3-węzlowy element

$$C^{(1)} = \frac{Sk}{L^{(1)}} = \frac{2m^2 \cdot 50\frac{W}{mK}}{2{,}5m} = 40\frac{W}{K} = C^{(2)};$$
$$\alpha S = 10\frac{W}{m^2K} \cdot 2m^2 = 20\frac{W}{K};$$
$$qS = -150\frac{W}{m^2} \cdot 2m^2 = -300W;$$
$$\alpha S t_\infty = 10\frac{W}{m^2K} \cdot 2m^2 \cdot 40K = 8000W.$$

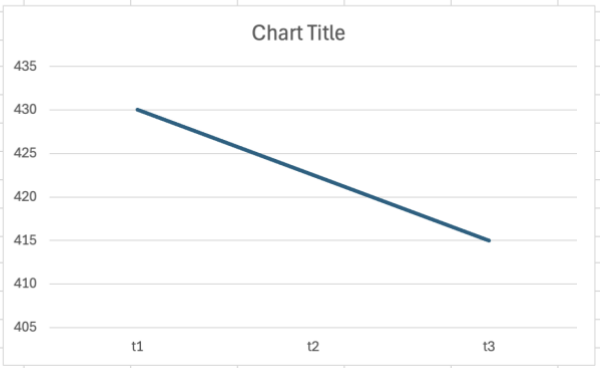
Stąd, otrzymuje się następujący układ równań:

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

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:

$t_1 = 430$
$t_2 = 426.25$
$t_3 = 422.5$
$t_4 = 418.75$
$t_5 = 415$

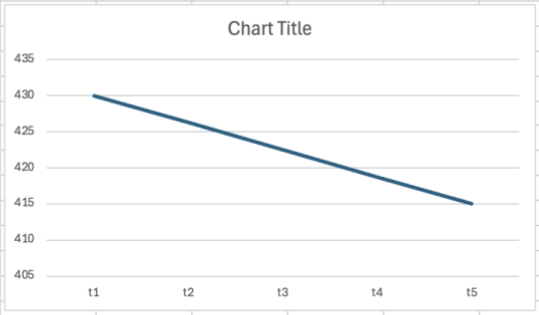
Wzór J dla 5-węzłowego układu:

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

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

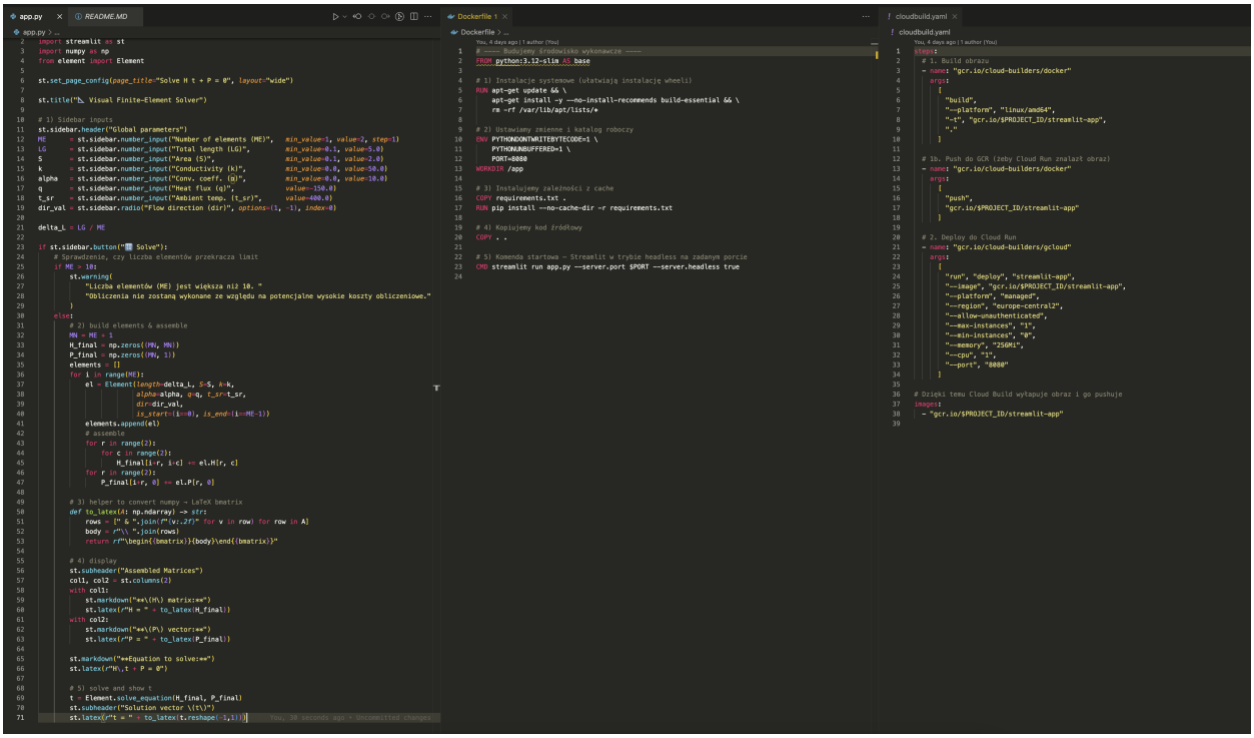
5 węzłów

	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:



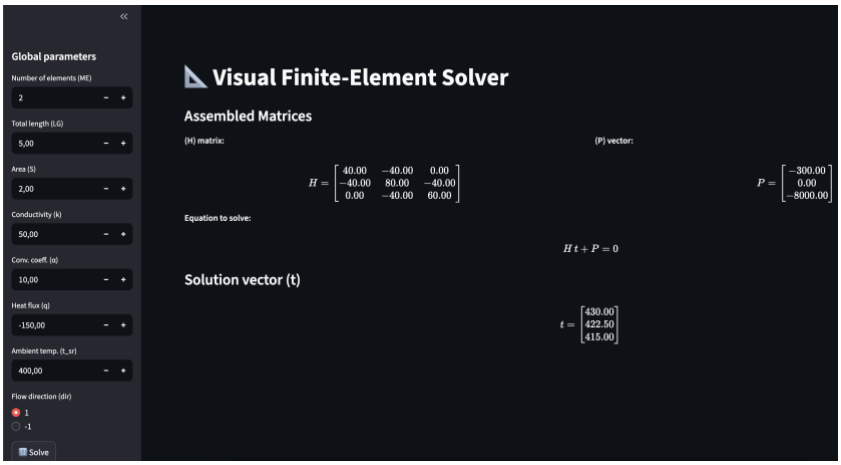
Ten kod był wygenerowany za pomocą ChatGPT na podstawie kodu, który był umieszczony wcześniej i służy wyłącznie do wizualizacji.
„Projekt" składa się z 3 plików:

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

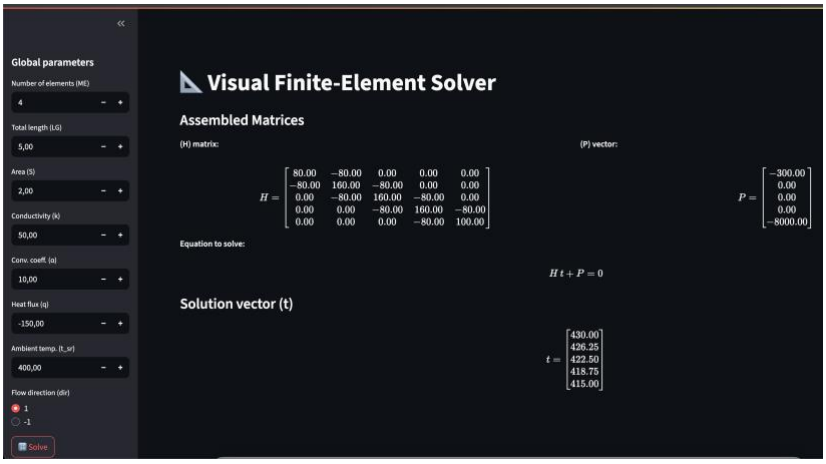
Pliki Dockerfile oraz Cloudbuild.yaml są potrzebne dla możliwości umieszczenia tego projektu w chmurze GCP, żeby dodać możliwość zaprezentowania tego programu wielu ludziom.
Aplikacja będzie dostępna w chmurze pod następnym linkiem (do 29/06/2025):
https://streamlit-app-83611701832.europe-central2.run.app/

Demo

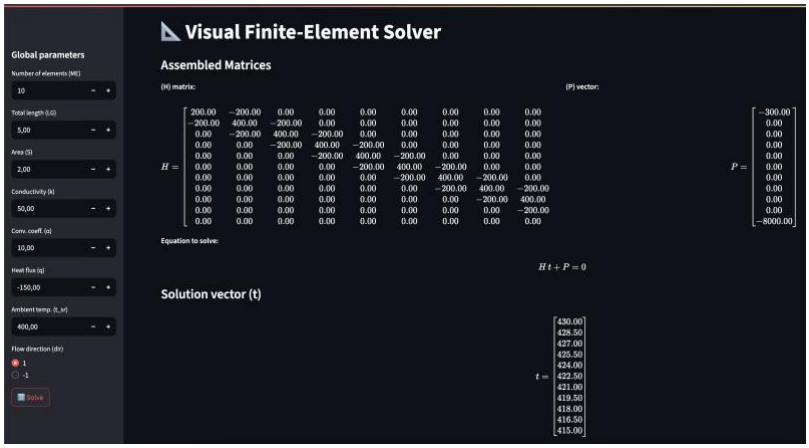
2 elementy



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.

Symulacja nieustalonych procesów cieplnych

Wstęp

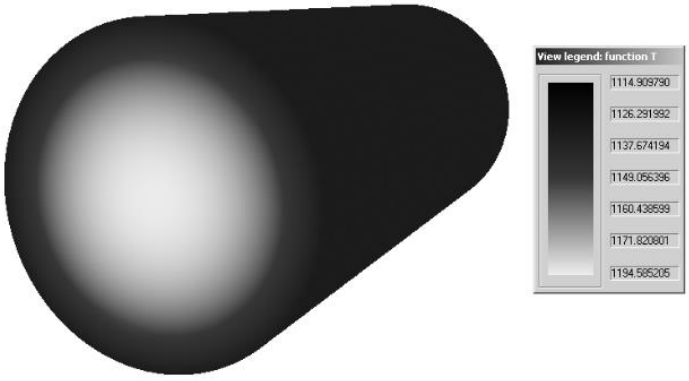
Celem tej części sprawozdania jest analiza niestacjonarnego (zmiennego w czasie) procesu przepływu ciepła we wsadzie o przekroju okrągłym. Analiza będzie zrobiona będzie zrobiona za pomocą programu w Pythonie (który był napisany na podstawie programu w Fortranie), który używa MES do rozwiązania problemu.

Program należy odpalić z różnymi parametrami:

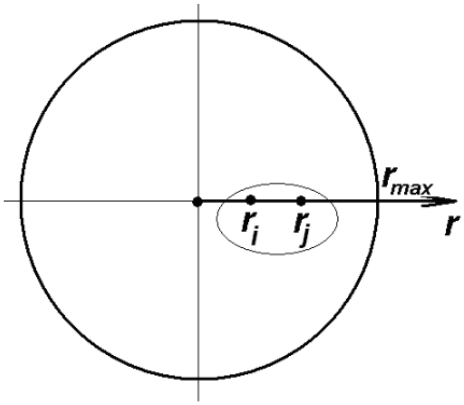
1. Różna ilość elementów bez zmiany innych parametrów i odpowiedzieć na pytanie dla jakiej ilości węzłów rozwiązanie jest stabilne.
2. Różna długość kroku czasowego bez zmiany innych parametrów i wywnioskować na co wpływa długość kroku czasowego.

Przedstawienie modelu z warunkami brzegowymi

Analizowano proces nieustalonego przewodnictwa ciepła we wsadzie o przekroju okrągłym (rys. 6.1). Założono, że wymiana ciepła będzie odbywała się w sposób osiowo-symetryczny (rys. 6.2).



Rys. 6.1. Przykładowy rozkład temperatury we wsadzie o przekroju okrągłym



Rys. 6.2. Schemat obliczeniowy do wyznaczenia nieustalonego rozkładu temperatury

Podstawy rozwiązania MES dla problemu niestacjonarnego

Podstawą analizy jest równanie Fouriera dla procesu niestacjonarnego

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

Zgrubna charakterystyka kodu

Analiza była zrobiona na podstawie kodu w Pythonem, który był napisany na podstawie kodu w Fortranie (TEMP1D). Program używa bibliotekę csv do zapisu wyników do pliku oraz bibliotekę matplotlib do rysowania wykresów. Poniżej umieszczam najbardziej istotną część kodu:

```
while Tau <= TauMax:
    time_history.append(Tau)
    t_center_history.append(vrtxTemp[0])
    t_surface_history.append(vrtxTemp[-1])
    dT_list_history.append(abs(vrtxTemp[0] - vrtxTemp[-1]))

    aC = [0.0] * nh
    aD = [0.0] * nh
    aE = [0.0] * nh
    aB = [0.0] * nh

    TempAir = t1  # stała temperatura otoczenia

    def assemble_element(ie):
        r1 = vrtxCoordX[ie]
        r2 = vrtxCoordX[ie + 1]
        t1e = vrtxTemp[ie]
        t2e = vrtxTemp[ie + 1]
        dR_local = r2 - r1
        Alfa = AlfaAir if ie == ne - 1 else 0.0

        Ke = [[0.0, 0.0], [0.0, 0.0]]
        Fe = [0.0, 0.0]

        for ip in range(Np):
            Rp = N1[ip] * r1 + N2[ip] * r2
            TpTau = N1[ip] * t1e + N2[ip] * t2e

            Ke[0][0] += K * Rp * W[ip] / dR_local + C * Ro * dR_local * Rp * W[ip] * N1[ip] ** 2 / dTau
            Ke[0][1] += -K * Rp * W[ip] / dR_local + C * Ro * dR_local * Rp * W[ip] * N1[ip] * N2[ip] / dTau
            Ke[1][0] = Ke[0][1]
            Ke[1][1] += K * Rp * W[ip] / dR_local + C * Ro * dR_local * Rp * W[ip] * N2[ip] ** 2 / dTau + 2 * Alfa * Rmax

            Fe[0] += C * Ro * dR_local * TpTau * Rp * W[ip] * N1[ip] / dTau
            Fe[1] += C * Ro * dR_local * TpTau * Rp * W[ip] * N2[ip] / dTau + 2 * Alfa * Rmax * TempAir

        return ie, Ke, Fe

    with ThreadPoolExecutor() as executor:
        results = executor.map(assemble_element, range(ne))

    for ie, Ke, Fe in results:
        aD[ie] += Ke[0][0]
        aD[ie + 1] += Ke[1][1]
        aE[ie] += Ke[0][1]
        aC[ie + 1] += Ke[1][0]
        aB[ie] += Fe[0]
        aB[ie + 1] += Fe[1]

    # Rozwiązanie układu równań metodą Thomasa
    for i in range(1, nh):
        m = aC[i] / aD[i - 1]
        aD[i] -= m * aE[i - 1]
        aB[i] -= m * aB[i - 1]

    aB[-1] /= aD[-1]
    for i in range(nh - 2, -1, -1):
        aB[i] = (aB[i] - aE[i] * aB[i + 1]) / aD[i]

    vrtxTemp = aB
    Tau += dTau

end = time.time()-start
print(f"Czas: {end:2f}s")
```

Kod głównie polega na tych wzorach:

$$K_{11} = \frac{k}{\Delta r}\sum_{p=1}^{n_p} r_p w_p + \frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p} N_i^2 r_p w_p \ ,$$

$$K_{12} = -\frac{k}{\Delta r}\sum_{p=1}^{n_p} r_p w_p + \frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p} N_i N_j r_p w_p \ ,$$

$$K_{21} = -\frac{k}{\Delta r}\sum_{p=1}^{n_p} r_p w_p + \frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p} N_i N_j r_p w_p \ ,$$

$$K_{22} = \frac{k}{\Delta r}\sum_{p=1}^{n_p} r_p w_p + \frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p} N_j^2 r_p w_p + 2\alpha r_{\max} \ ,$$

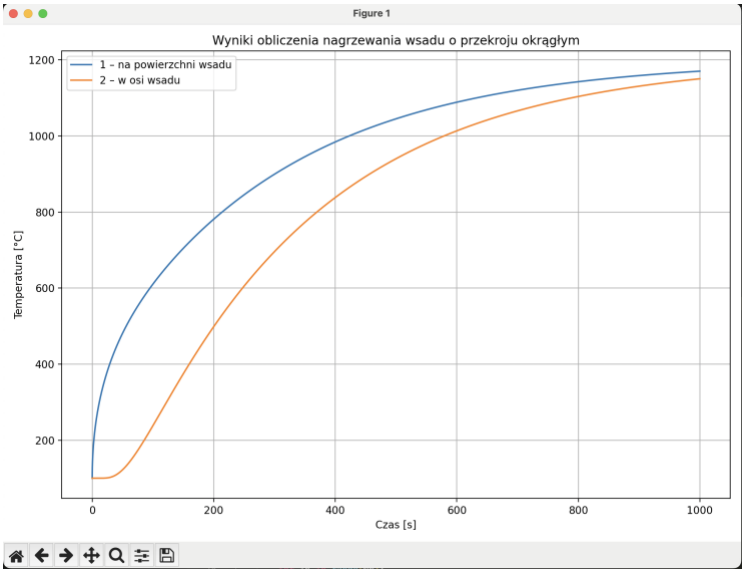
$$F_1 = -\frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p}\left(N_i t_{i0} + N_j t_{j0}\right) N_i r_p w_p \ ,$$

$$F_2 = -\frac{c\rho\Delta r}{\Delta\tau}\sum_{p=1}^{n_p}\left(N_i t_{i0} + N_j t_{j0}\right) N_j r_p w_p - 2\alpha r_{\max} t_\infty .$$

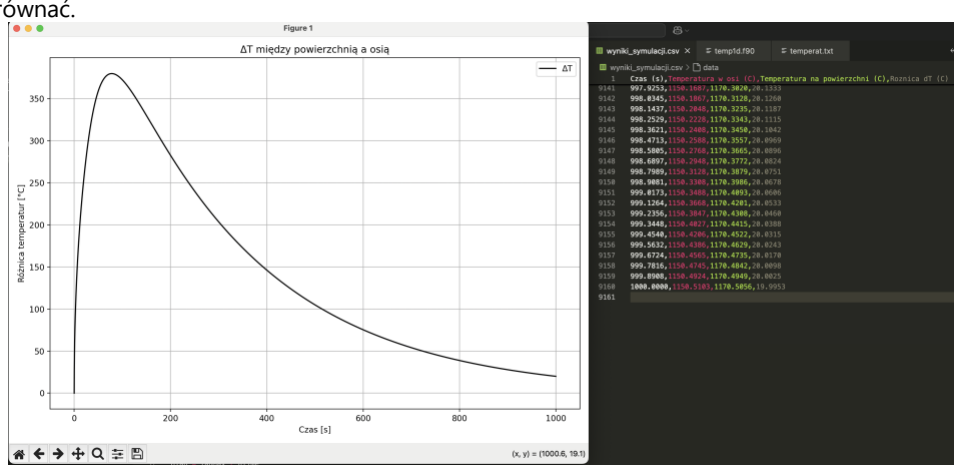
Analiza wpływu wielkości elementu oraz długości kroku czasowego na wyniki obliczeń

1. Parametry standardowe

dt = 0.4367 s
Czas: 2.498515 s



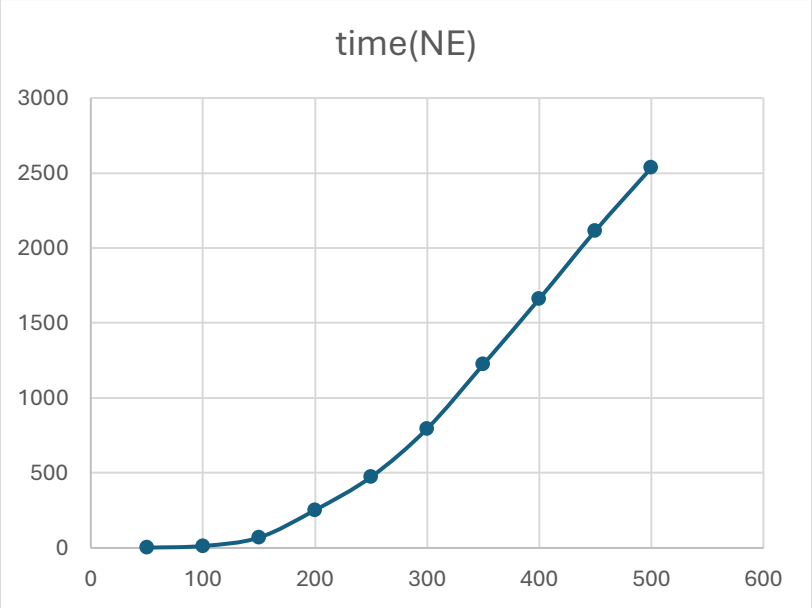
Jak widać temperatury się zbliżają, co jest oczekiwane. Wsad się grzeje przez temperaturę otoczenia i dlatego pomarańczowy wykres jest opóźniony – ma mniejszą temperaturę niż powierzchnia, ale po pewnym czasie temperatury powinny się wyrównać.



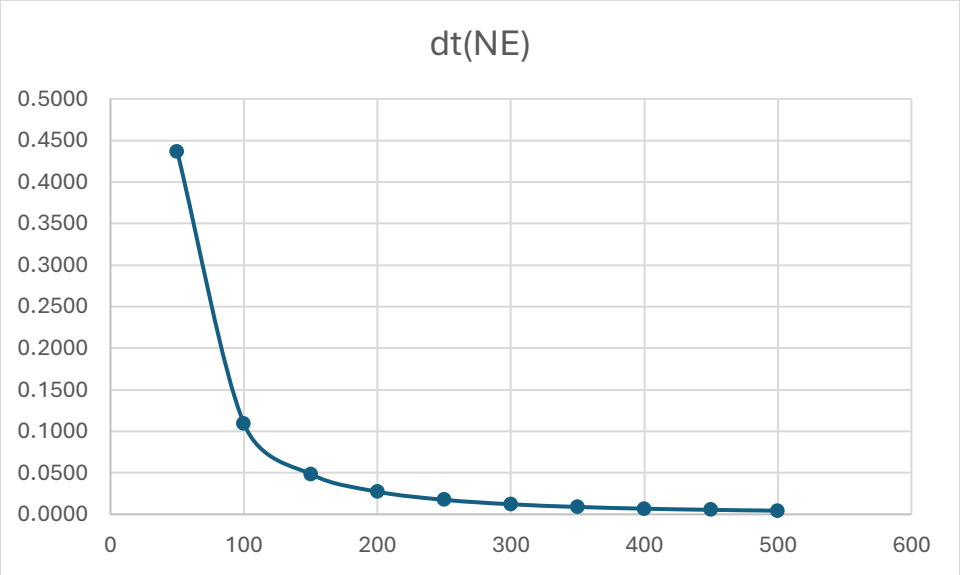
Wyniki oraz dodatkowy wykres różnicy temperatury pokazują, że po 1000 sekundach różnica wynosi ~19 stopni.

2. Zmiana liczy elementów

NE	time, s	dt, s
50	2.17050815	0.4367
100	13.1513889	0.1092
150	69.3653231	0.0485
200	252.294886	0.0273
250	473.519242	0.0175
300	795.241153	0.0121
350	1222.50325	0.0089
400	1660.02613	0.0068
450	2113.89599	0.0054
500	2534.29775	0.0044



Zależność czasu wykonania programu od ilości elementów



Zależność dt od ilości elementów

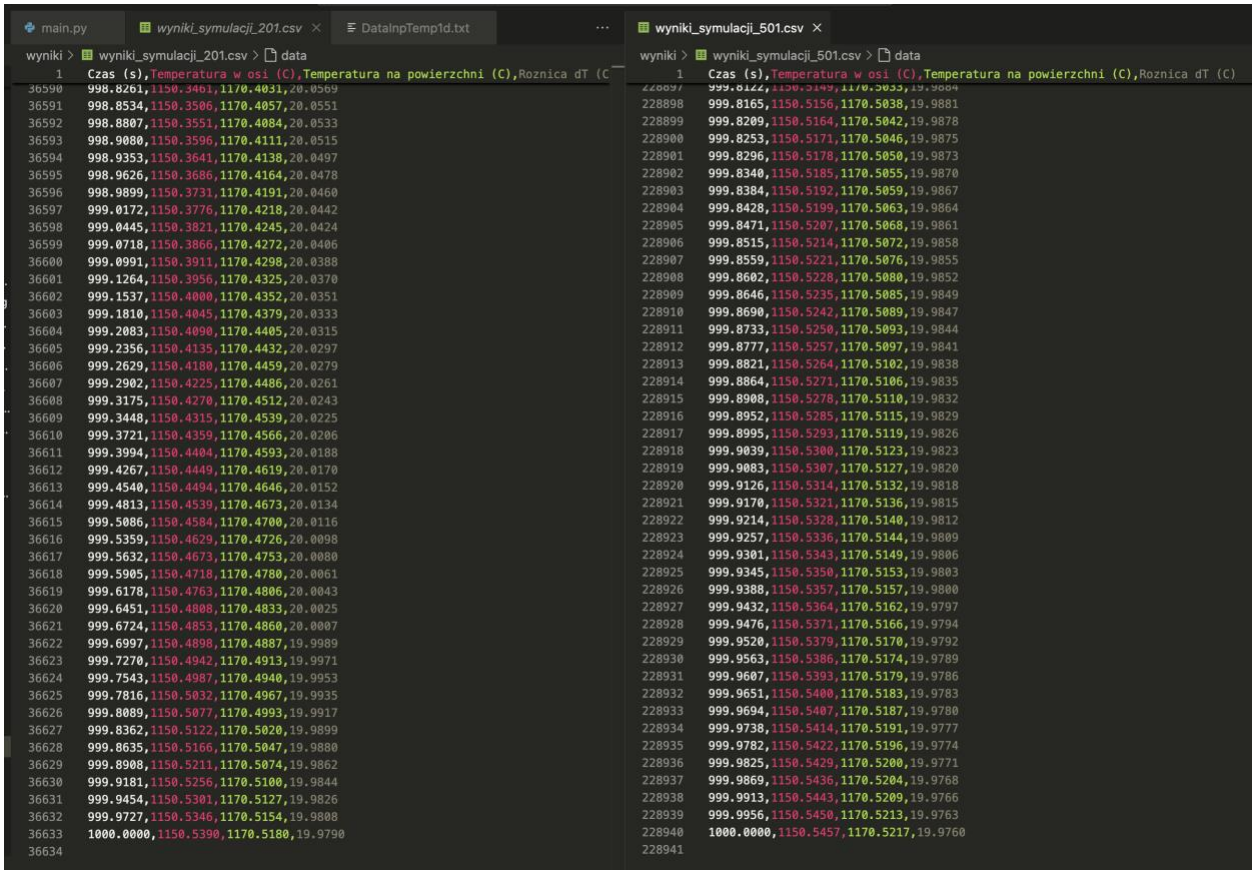
Analiza wyników

Na podstawie danych możemy wywnioskować, że zwiększenie liczby elementów bezpośrednio wpływa na:

1. czas obliczeń
    a. jest większy, bo zwiększa się liczba kroków
    b. więcej obliczeń w każdym kroku
2. długość dt (wzór na obliczenie dt używa liczbę elementów)
3. precyzję obliczeń

Możemy zauważyć, że wykres dt(NE) się wypłaszacza i dalsze zwiększenie liczby elementów niewiele zwiększy precyzję, ale znacznie zwiększy koszty obliczeniowe. Na szybki rzut oka wygląda, że złożoność kodu jest $O(N^3)$ – wyniki czasowe mogą z tym się zgadzać, bo wszystkie obliczenia (dla każdej ilości elementów) było odpalone w tym samym czasie (żeby szybciej policzyć) co mogło spowodować wolniejszą pracę każdego z tych procesów (każde liczba wątków – to osobny proces, który w sobie zawierał wątki).

Również można zauważyć, że wyniki przy 200 i 500 są prawie identyczne, ale liczba wierszy jest o ~6.3 większa. Czyli jeśli rozwiązanie nie wymaga bardzo dużej precyzji, to można się ograniczyć 200 elementami.



main.py | wyniki_symulacji_201.csv | DataInpTemp1d.txt

wyniki > wyniki_symulacji_201.csv > data

	Czas (s)	Temperatura w osi (C)	Temperatura na powierzchni (C)	Roznica dT (C
36590	998.8261	1150.3461	1170.4031	20.0569
36591	998.8534	1150.3506	1170.4057	20.0551
36592	998.8807	1150.3551	1170.4084	20.0533
36593	998.9080	1150.3596	1170.4111	20.0515
36594	998.9353	1150.3641	1170.4138	20.0497
36595	998.9626	1150.3686	1170.4164	20.0478
36596	998.9899	1150.3731	1170.4191	20.0460
36597	999.0172	1150.3776	1170.4218	20.0442
36598	999.0445	1150.3821	1170.4245	20.0424
36599	999.0718	1150.3866	1170.4272	20.0406
36600	999.0991	1150.3911	1170.4298	20.0388
36601	999.1264	1150.3956	1170.4325	20.0370
36602	999.1537	1150.4000	1170.4352	20.0351
36603	999.1810	1150.4045	1170.4379	20.0333
36604	999.2083	1150.4090	1170.4405	20.0315
36605	999.2356	1150.4135	1170.4432	20.0297
36606	999.2629	1150.4180	1170.4459	20.0279
36607	999.2902	1150.4225	1170.4486	20.0261
36608	999.3175	1150.4270	1170.4512	20.0243
36609	999.3448	1150.4315	1170.4539	20.0225
36610	999.3721	1150.4359	1170.4566	20.0206
36611	999.3994	1150.4404	1170.4593	20.0188
36612	999.4267	1150.4449	1170.4619	20.0170
36613	999.4540	1150.4494	1170.4646	20.0152
36614	999.4813	1150.4539	1170.4673	20.0134
36615	999.5086	1150.4584	1170.4700	20.0116
36616	999.5359	1150.4629	1170.4726	20.0098
36617	999.5632	1150.4673	1170.4753	20.0080
36618	999.5905	1150.4718	1170.4780	20.0061
36619	999.6178	1150.4763	1170.4806	20.0043
36620	999.6451	1150.4808	1170.4833	20.0025
36621	999.6724	1150.4853	1170.4860	20.0007
36622	999.6997	1150.4898	1170.4887	19.9989
36623	999.7270	1150.4942	1170.4913	19.9971
36624	999.7543	1150.4987	1170.4940	19.9953
36625	999.7816	1150.5032	1170.4967	19.9935
36626	999.8089	1150.5077	1170.4993	19.9917
36627	999.8362	1150.5122	1170.5020	19.9899
36628	999.8635	1150.5166	1170.5047	19.9880
36629	999.8908	1150.5211	1170.5074	19.9862
36630	999.9181	1150.5256	1170.5100	19.9844
36631	999.9454	1150.5301	1170.5127	19.9826
36632	999.9727	1150.5346	1170.5154	19.9808
36633	1000.0000	1150.5390	1170.5180	19.9790
36634				

wyniki_symulacji_501.csv

wyniki > wyniki_symulacji_501.csv > data

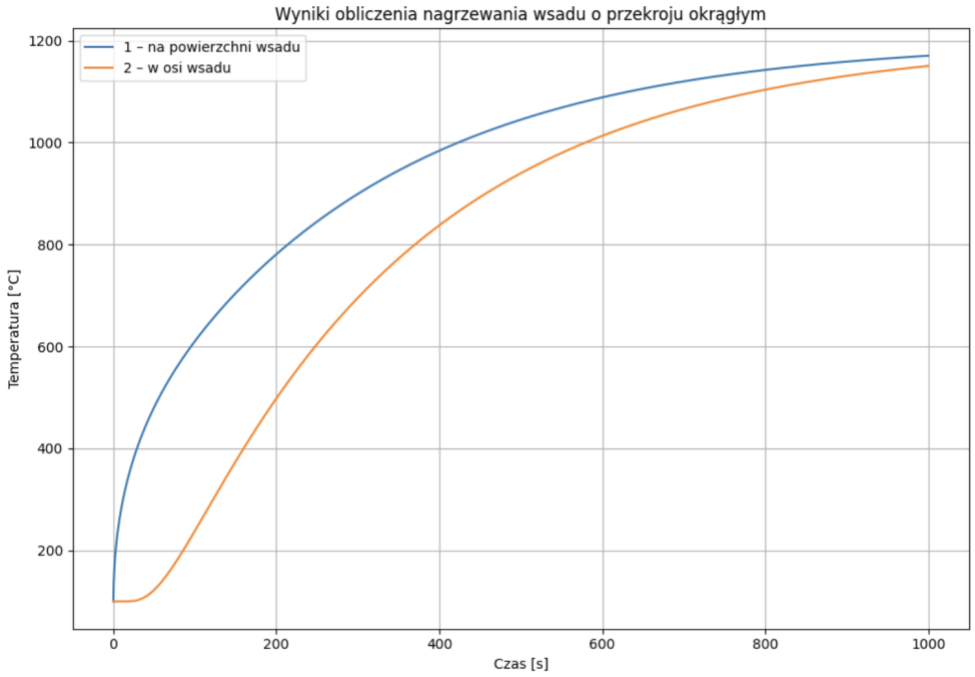
	Czas (s)	Temperatura w osi (C)	Temperatura na powierzchni (C)	Roznica dT (C)
228897	999.8122	1150.5149	1170.5035	19.9884
228898	999.8165	1150.5156	1170.5038	19.9881
228899	999.8209	1150.5164	1170.5042	19.9878
228900	999.8253	1150.5171	1170.5046	19.9875
228901	999.8296	1150.5178	1170.5050	19.9873
228902	999.8340	1150.5185	1170.5055	19.9870
228903	999.8384	1150.5192	1170.5059	19.9867
228904	999.8428	1150.5199	1170.5063	19.9864
228905	999.8471	1150.5207	1170.5068	19.9861
228906	999.8515	1150.5214	1170.5072	19.9858
228907	999.8559	1150.5221	1170.5076	19.9855
228908	999.8602	1150.5228	1170.5080	19.9852
228909	999.8646	1150.5235	1170.5085	19.9849
228910	999.8690	1150.5242	1170.5089	19.9847
228911	999.8733	1150.5250	1170.5093	19.9844
228912	999.8777	1150.5257	1170.5097	19.9841
228913	999.8821	1150.5264	1170.5102	19.9838
228914	999.8864	1150.5271	1170.5106	19.9835
228915	999.8908	1150.5278	1170.5110	19.9832
228916	999.8952	1150.5285	1170.5115	19.9829
228917	999.8995	1150.5293	1170.5119	19.9826
228918	999.9039	1150.5300	1170.5123	19.9823
228919	999.9083	1150.5307	1170.5127	19.9820
228920	999.9126	1150.5314	1170.5132	19.9818
228921	999.9170	1150.5321	1170.5136	19.9815
228922	999.9214	1150.5328	1170.5140	19.9812
228923	999.9257	1150.5336	1170.5144	19.9809
228924	999.9301	1150.5343	1170.5149	19.9806
228925	999.9345	1150.5350	1170.5153	19.9803
228926	999.9388	1150.5357	1170.5157	19.9800
228927	999.9432	1150.5364	1170.5162	19.9797
228928	999.9476	1150.5371	1170.5166	19.9794
228929	999.9520	1150.5379	1170.5170	19.9792
228930	999.9563	1150.5386	1170.5174	19.9789
228931	999.9607	1150.5393	1170.5179	19.9786
228932	999.9651	1150.5400	1170.5183	19.9783
228933	999.9694	1150.5407	1170.5187	19.9780
228934	999.9738	1150.5414	1170.5191	19.9777
228935	999.9782	1150.5422	1170.5196	19.9774
228936	999.9825	1150.5429	1170.5200	19.9771
228937	999.9869	1150.5436	1170.5204	19.9768
228938	999.9913	1150.5443	1170.5209	19.9766
228939	999.9956	1150.5450	1170.5213	19.9763
228940	1000.0000	1150.5457	1170.5217	19.9760
228941				

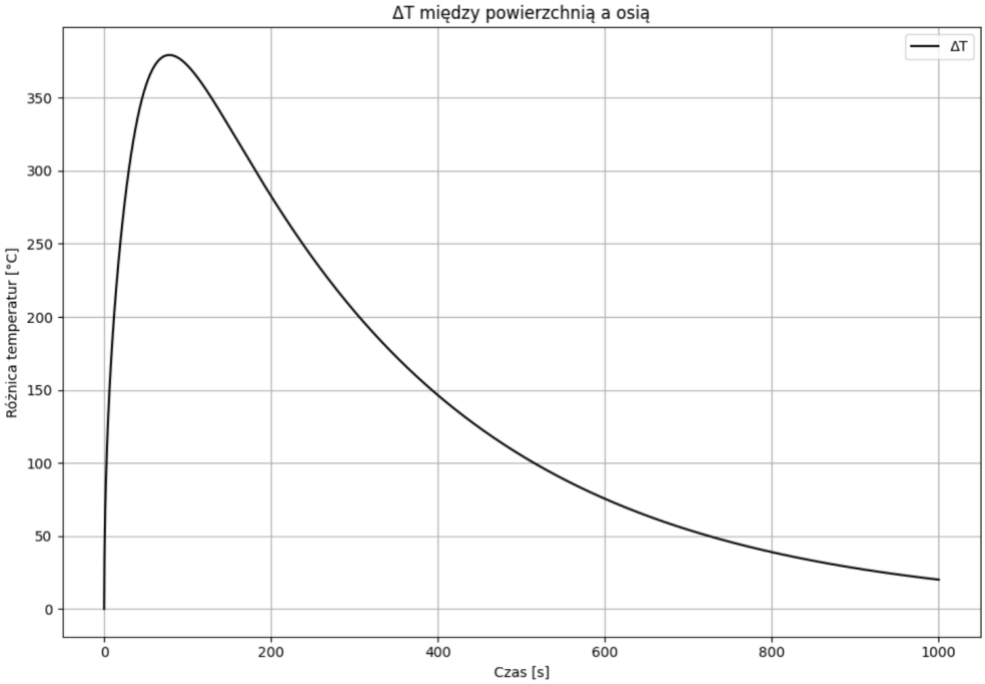
3. Zmiana długości kroku czasowego

Zmiana dTau się odbywa przez modyfikację następnej linijki

```
dTau = dR ** 2 / (0.5 * a)
```

Parametry standardowe:
dt = 0.4367 s
Czas: 7.677455s
dT na końcu obliczeń: 20.0313





	dt, s	time, t	dT, K
dt/8	0.0546	68.983486	19.9893
dt/4	0.1092	37.500213	19.9953
dt	0.4367	7.677455	20.0313
4dt	1.7452	1.769638	20.1752
8dt	3.4843	0.85683	20.3668
100dt	43.4783	0.067629	24.8586

Analiza wyników

Na co wpływa długość kroku czasowego? Długość kroku czasowego wpływa na czas obliczeń (kolumna *time*) oraz powinna wpływać na precyzję obliczeń. Chociaż różnica pomiędzy końcową temperaturą przy $dt/8$; dt; $8dt$ nie jest dużo i wynosi:

$$dT_{dt} - dT_{\frac{dt}{8}} = 20.0313 - \ 19.9893 = 0.042 \text{ K } (0.2\%)$$

$$dT_{dt} - dT_{8dt} = |20.0313 - 20.3668| = 0.3355 \text{ K } (1.67\%)$$

Widoczną różnice widać przy 100dt i ona wynosi 4.8273K. Ale dalej widać czy wyniki końcowe są podobne. Ta długość przede wszystkim wpływa na precyzję obliczeń przy używaniu metody Gausa.