Sprawozdanie
Metoda elementow skończonych

Alicja Miotk
Inżynieria Obliczeniowa
nr albumu: 299 633
Grupa 02

Wstęp

Celem zadania było stworzenie symulacji transferu ciepła za pomocą metody elementów skończonych. Zadanie zrealizowałam przy pomocy języka ***C++*** oraz wykorzystałam IDE od JetBrains: ***Clion***.

Model zakłada siatkę 2D- na płaszczyźnie, składającą się z elementów czterowęzłowych. Wykorzystany został dwupunktowy schemat całkowania Gaussa. Poszczególne punkty zostały opisane na podstawie kwadratur Gaussa-Legendre'a.
Do realizacji oprogramowania MES 2D zagadnienia termicznego został wykorzystany warunek brzegowy konwekcji.

Opis kodu

I etap - stworzenie siatki

Pierwszym etapem było stworzenie siatki na płaszczyźnie.
Utworzyłam poniższe klasy:

GlobalData **-** wczytuje wszystkie potrzebne dane do rozwiązania z pliku mes1.txt/mes2.txt (odpowiednio dwa test casy), tj.:

- temperatura początkowa [C]
- czas symulacji i krok czasowy [s]
- temperatura otoczenia [C]
- alfa (współczynnik wymiany cieplnej) [W/m^2K]
- wysokość i szerokosć siatki [m]
- liczba elementów horyzontalnie i wertykalnie
- ilość węzłów w elemencie
- współczynnik przewodzenia [W/(m*C)]
- ciepło właściwe [J/(kg*C)]
- gęstość materiału [kg/m^3].

Node **-** tworzy węzeł o parametrach:

- id - numer węzła
- x, y – współrzędne węzła
- t – temperatura węzła
- borderCondition – flaga węzła(0 lub 1) w zależności czy węzeł leży na krawędzi siatki.

Element – tworzy element, czyli przechowuje 4 id węzłów, które określają każdy element:

- przyjmuje wektor wskaźników do nodów
- dodatkowo jest metoda używana do wypisania elementów

Grid – tworzy siatkę, czyli „układa" elementy oraz węzły w odpowiednich miejscach – dzieje się to w konstruktorze klasy.

- Zawiera wektor elementów(***elements***) oraz wektor węzłów(***nodes***)
- ***deltax***, ***deltay*** – określają szerokość oraz wysokość każdego elementu, po to aby odpowiednio ułożyć je w siatce
- pierwsze dwie pętle tworzą węzły o odpowiednich ***id***, współrzędnych ***x, y***, ***temperaturze*** = temperaturze początkowej oraz ***borderCondition***, do którego wykorzystuję osobną funkcję - ***checkBorderCondition***, która w zależności od tego czy węzeł leży na krawędzi nadaje mu flagę 1, lub jeśli nie to 0. oraz dodaje te węzły do wektora ***nodes***.
- w kolejnej części przypisuję elementom odpowiednie węzły, wykorzystuję tutaj zmienną *columnElements*, która pozwala stwierdzić czy dany element należy do pierwszej kolumny siatki, czy już do kolejnej. Korzystam z tymczasowego wektora wskaźników do Nodów oraz dodaje je kolejno do wektora ***elements***.

II etap – całkowanie macierzy H lokalnie

Stworzyłam klasę ***UniversalElement***, która w konstruktorze definiuje potrzebne wektory, ustawia 4 punkty całkowania na powierzchni (***integrationPoints***) o odpowiednich współrzędnych z dwupunktowego schematu całkowania Gaussa oraz o odpowiednich wartościach, które określiłam jako wartości globalne ***VALUE_PLUS*** oraz ***VALUE_MINUS***.

W konstruktorze odsyłam do metody ***calculateShapeFunctions***, która liczy pochodne funkcji kształtu po ksi i eta, i funkcje kształtu w każdym punkcie całkowania na powierzchni elementu(***integrationPoints***). Odsyłam również do metody ***pointsOnTheEdges***, która będzie potrzebna przy liczeniu macierzy HBC oraz wektora P.

Całkowanie ***macierzy H*** przebiega w metodzie ***createMatrixHandC***, gdzie również liczę macierz C.

Wszystko dzieję się w pętli po punktach całkowania, bo liczę macierz H w każdym punkcie oraz potem te 4 macierze sumuję.

Na początku następuje policzenie ***macierzy Jacobiego***, ***wyznacznika macierzy Jacobiego*** oraz ***odwrotnej macierzy Jacobiego*** – dzieje się to w metodzie ***calculateJacobiTransformation***.

Macierz Jacobiego ma wymiar 2x2- liczę ją w każdym punkcie całkowania.

Następnie liczę wyznacznik macierzy Jacobiego jak zwykły wyznacznik macierzy 2x2 oraz macierz odwrotną Jacobiego poprzez pomnożenie (1/det) * macierz Jacobiego.

Następnie obliczam pochodne funkcji kształtu po x i y już w metodzie ***createMatrixHandC***, w tym celu odpowiednio mnożę odwrotną macierz Jacobiego i pochodne funkcji kształtu po ksi i eta.

Dzięki przekształceniu Jacobiego mam pochodne funkcji kształtu po x oraz y, wystarczy przemnożyć odpowiednio wektor przez wektor transponowany dla pochodnych funkcji kształtu po x oraz y i je do siebie dodać. Dalej następuje przemnożenie przez współczynnik przewodzenia oraz z racji, że aby obliczyć macierz H liczę całkę po objętości mnożę jeszcze razy wyznacznik obliczony wcześniej w metodzie *calculateJacobiTransformation*. Macierz H liczę w każdym punkcie całkowania, czyli 4 razy.

Po zsumowaniu 4 macierzy otrzymuję ***lokalną macierz H*** dla danego elementu.

Etap III – całkowanie macierzy C lokalnie

Macierz C również jest liczona w metodzie ***createMatrixHandC.*** Wykorzystuję obliczone już w ***calculateShapeFunctions*** funkcje kształtu dla każdego punktu całkowania(***integrationPoints***). Macierz C liczę również dla każdego punktu całkowania, więc 4 razy- dla każdego mnożąc wektor funkcji kształtu przez wektor transponowany oraz dodatkowo przez ciepło właściwe i gęstość. Aby policzyć macierz C, to również mamy całkę po objętości, więc mnożę wynik jeszcze przez wyznacznik obliczony w *calculateJacobiTransformation.*

Sumuję 4 macierze C i otrzymuję ***lokalną macierz C*** dla danego elementu.

Etap IV – całkowanie macierzy HBC lokalnie

Macierz HBC jest częścią macierzy H – jest to obciążenie generowane przez warunek brzegowy na macierz H, jednak liczę ją osobno. Jest to całka po powierzchni.

Wykorzystuję metodę ***pointOnTheEdges*** w klasie ***UniversalElement***, która ustawia punkty całkowania na krawędziach w elemencie (po 2 punkty na każdej krawędzi – w sumie 8 punktów) o zadanych wartościach. Liczę od razu w nich funkcje kształtu. Otrzymuję jedną macierz – 8x4 - ***functionNEdges***.

Liczenie macierzy HBC:
Na początku sprawdzam w metodzie ***checkIfEdge*** klasy ***Grid*** czy dane krawędzie w elemencie są brzegowe(poprzez sprawdzenie czy dwa węzły obok siebie mają flagi borderCondition == 1), jeśli krawędź jest brzegowa, to liczę dla niej w ***edgeLength***(rownież w klasie Grid) jakobian przekształcenia dla układu 1d - *detJ*, czyli stosunek długości boku w układzie globalnym do długości boku w układzie lokalnym.
Następnie dla tej krawędzi liczę ***macierz HBC*** w metodzie ***matrixHBCandVecP*** klasy ***UniversalElement*** i tak dla każdej krawędzi brzegowej w elemencie.

W metodzie ***matrixHBCandVecP*** sprawdzam najpierw dla której dokładnie krawędzi muszę policzyć macierz HBC, a następnie mnożę wektor obliczonych funkcji kształtu w odpowiednim punkcie całkowania na krawędzi przez wektor transponowany oraz dodaję ten sam schemat dla drugiego punktu całkowania z tej samej krawędzi. Całość mnożę jeszcze przez alfę, czyli współczynnik wymiany cieplnej oraz detJ obliczony wcześniej.

Macierz HBC liczę dla każdej krawędzi brzegowej w elemencie- czyli maksymalnie 2 razy.
Dodaje obliczone macierze do siebie i otrzymuję ***lokalną macierz HBC*** dla danego elementu.

Jeżeli żadne krawędzie w elemencie nie są brzegowe, to lokalna macierz HBC jest wypełniona zerami.

Etap V – całkowanie wektora obciążeń P lokalnie

Samo sprawdzanie czy krawędź jest brzegowa przebiega dokładnie tak samo jak przy liczeniu macierzy HBC. Wektor P liczę w tej samej metodzie: ***matrixHBCandVecP.*** Jest to również całka po powierzchni.

Aby obliczyć ***wektor P*** dodaję wektor obliczonych funkcji kształtu w odpowiednim punkcie całkowania na krawędzi do drugiego wektora w drugim punkcie na tej krawędzi i całość mnożę przez temperaturę otoczenia i alfę oraz wcześniej obliczony jakobian przekształcenia detJ.
Wektor P liczę dla każdej krawędzi brzegowej – czyli maksymalnie 2 razy w elemencie.

Dodaję do siebie obliczone wektory i otrzymuję ***lokalny wektor P*** dla danego elementu.
Jeżeli żadne krawędzie w elemencie nie są brzegowe, to lokalny wektor P jest wypełniony zerami.

Etap VI – agregacja, pętla po elementach oraz po czasie

Moja główna metoda znajduję się w klasie ***Grid*** – ***calculate***. Tworzę na początku potrzebne wektory oraz zmienne. Tworzę instancję klasy ***UniversalElement*** oraz wchodzę do pętli po czasie.
Liczba iteracji to czas symulacji/krok czasowy. Zeruję na początku każdej iteracji wektory agregacji, aby na pewno wyniki były poprawne. Wchodzę do pętli po elementach, w której najpierw sprawdzam dla każdego elementu czy krawędzie są brzegowe, jeśli tak to liczę macierz lokalną HBC oraz lokalny wektor P.
Następnie tworzę macierze lokalne H i C.

Kolejne pętle to agregacja macierzy H, C, HBC oraz wektora P, tak aby lokalne rozwiązania przenieść do układu globalnego.

Następnie, już poza pętlą po elementach, obliczam część równania głównego na zagregowanych macierzach, oraz wektorze czyli:

- [H] = [H] + [C] / krok czasowy
- pomocniczy wektor {matrixCxT0} = {matrixCxT0} + ([C] /krok czasowy) * {temperatura początkowa}
- {P} = {P} + {matrixCxT0}

Kolejnym krokiem jest już rozwiązanie równania macierzowego, czyli uzyskanie {T1}.
Do tego wykorzystuję metodę ***solveEquation*** również w klasie ***Grid***.

Rozwiąnie uzyskuję za pomocą metody eliminacji Gaussa dla równań liniowych korzystając z rozkładu macierzy LU.

Następnie wyszukuję minimalne i maksymalne temperatury dla każdej iteracji oraz przypisuję węzłom nowe temperatury.

Na samym końcu wartości wektora T1 przypisuję do wektora temperatur początkowych dla kolejnej iteracji, a T1 zeruję.

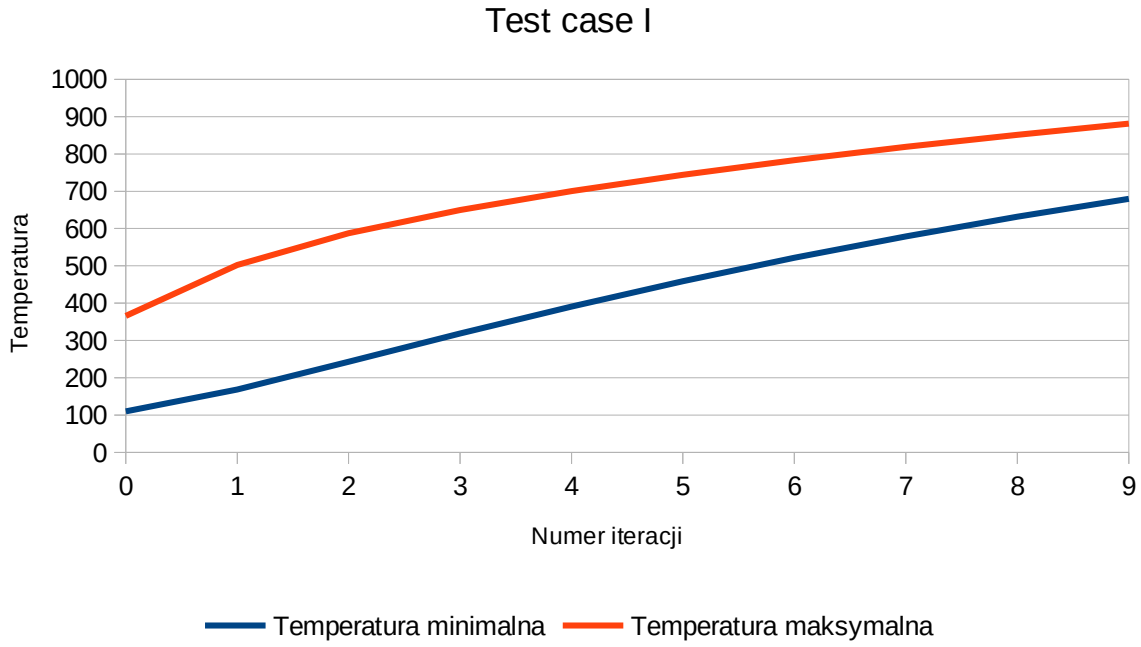
Dodatkowo zamieściłam metody wypisujące poszczególne etapy symulacji w celu potwierdzenia zgodnosci wyników na poszczególnych etapach tworzenia.

Działanie programu

Przedstawiam działanie programu w formie wykresów temperatury minimalnej i maksymalnej w kolejnych iteracjach dla podanych test casów oraz wypisanie tych temperatur w celu pokazania, że program działa poprawnie.

Dodatkowo przedstawiam porównanie w postaci zdjęć ekranu ze strony prowadzącego przedstawiających poprawne wyniki.

Wykres 1.



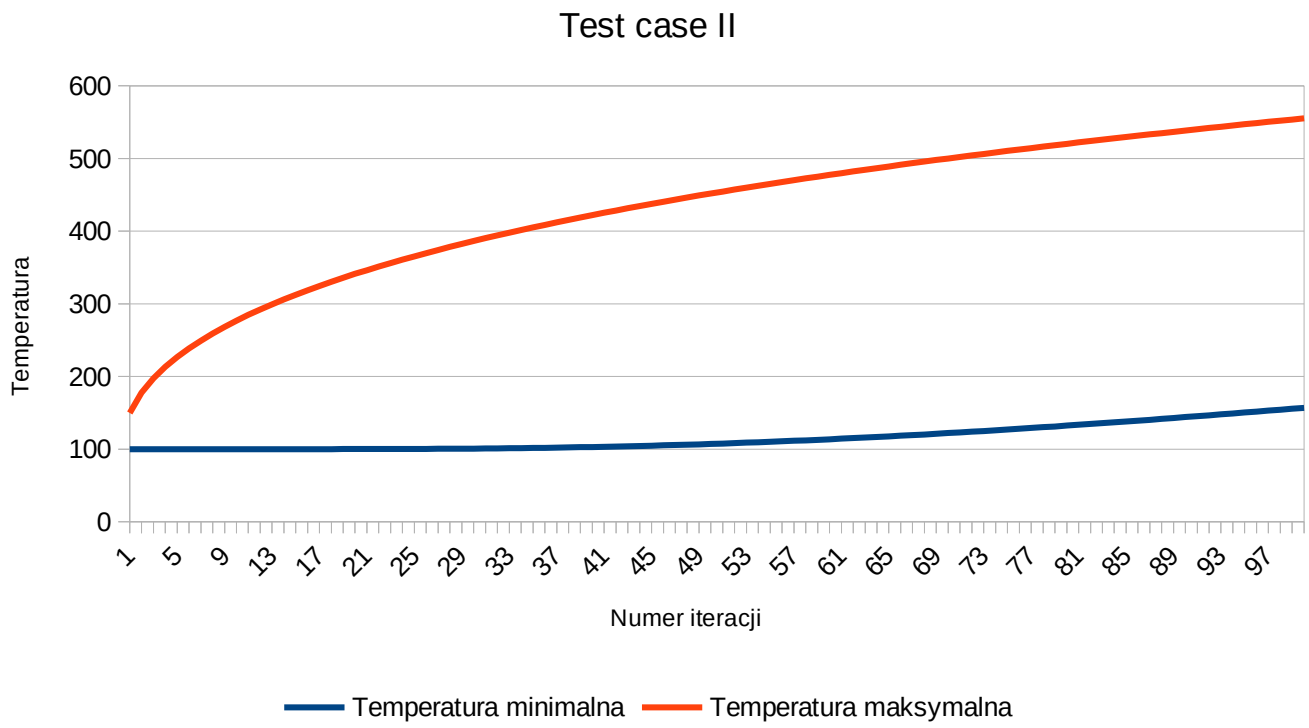
Moje wyniki- test case I

```
/home/ala/Desktop/MES/Mes_Proj/cmake-build-debug/Mes_Proj
ITERACJA: 0
MIN: 110.038  MAX: 365.815

ITERACJA: 1
MIN: 168.837  MAX: 502.592

ITERACJA: 2
MIN: 242.801  MAX: 587.373

ITERACJA: 3
MIN: 318.615  MAX: 649.387

ITERACJA: 4
MIN: 391.256  MAX: 700.068

ITERACJA: 5
MIN: 459.037  MAX: 744.063

ITERACJA: 6
MIN: 521.586  MAX: 783.383

ITERACJA: 7
MIN: 579.034  MAX: 818.992

ITERACJA: 8
MIN: 631.689  MAX: 851.431

ITERACJA: 9
MIN: 679.908  MAX: 881.058


Process finished with exit code 0
```

Wyniki z test case I – ze strony

Time[s]	MinTemp[s]	MaxTemp[s]
50	110.038	365.815
100	168.837	502.592
150	242.801	587.373
200	318.615	649.387
250	391.256	700.068
300	459.037	744.063
350	521.586	783.383
400	579.034	818.992
450	631.689	851.431
500	679.908	881.058

Wykres 2.



Moje wyniki- test case II

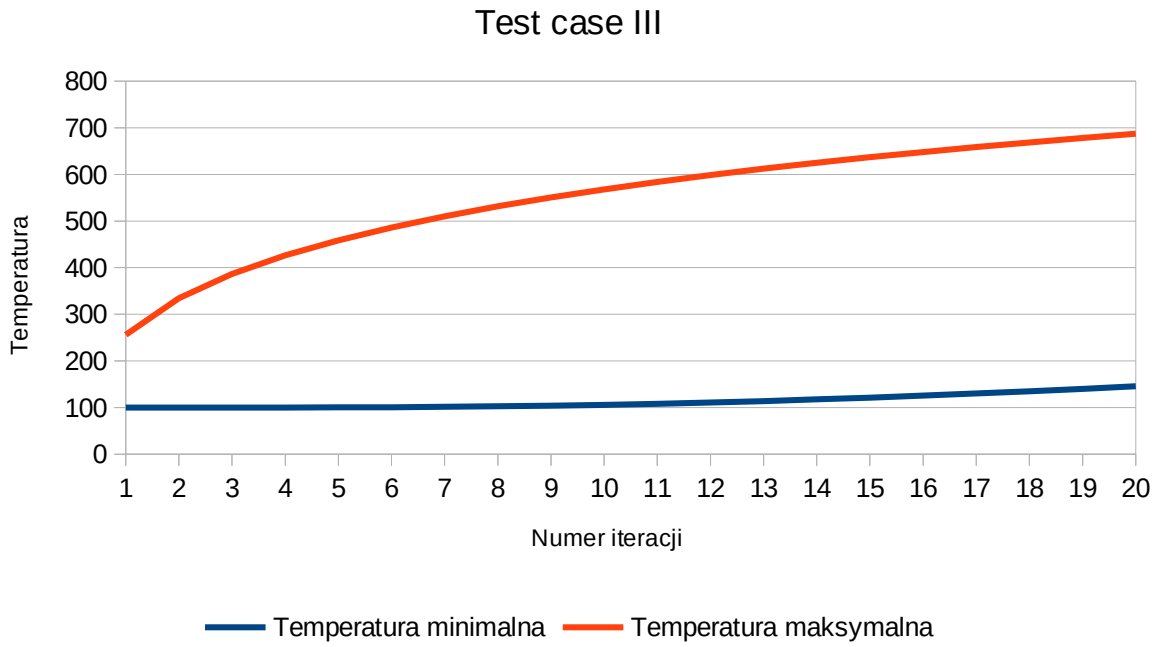
```
/home/ala/Desktop/MES/Mes_Proj/c
ITERACJA: 0
MIN: 100  MAX: 149.557

ITERACJA: 1
MIN: 100  MAX: 177.445

ITERACJA: 2
MIN: 100  MAX: 197.267

ITERACJA: 3
MIN: 100  MAX: 213.153

ITERACJA: 4
MIN: 100  MAX: 226.683

ITERACJA: 5
MIN: 100  MAX: 238.607

ITERACJA: 6
MIN: 100  MAX: 249.347

ITERACJA: 7
MIN: 100  MAX: 259.165

ITERACJA: 8
MIN: 100  MAX: 268.241

ITERACJA: 9
MIN: 100  MAX: 276.701

ITERACJA: 10
MIN: 100.001  MAX: 284.641
```

```
ITERACJA: 11
MIN: 100.002  MAX: 292.134

ITERACJA: 12
MIN: 100.003  MAX: 299.237

ITERACJA: 13
MIN: 100.005  MAX: 305.997

ITERACJA: 14
MIN: 100.009  MAX: 312.451

ITERACJA: 15
MIN: 100.014  MAX: 318.631

ITERACJA: 16
MIN: 100.021  MAX: 324.564

ITERACJA: 17
MIN: 100.032  MAX: 330.271

ITERACJA: 18
MIN: 100.046  MAX: 335.772

ITERACJA: 19
MIN: 100.064  MAX: 341.085

ITERACJA: 20
MIN: 100.088  MAX: 346.223
```

Wyniki z test case II – ze strony

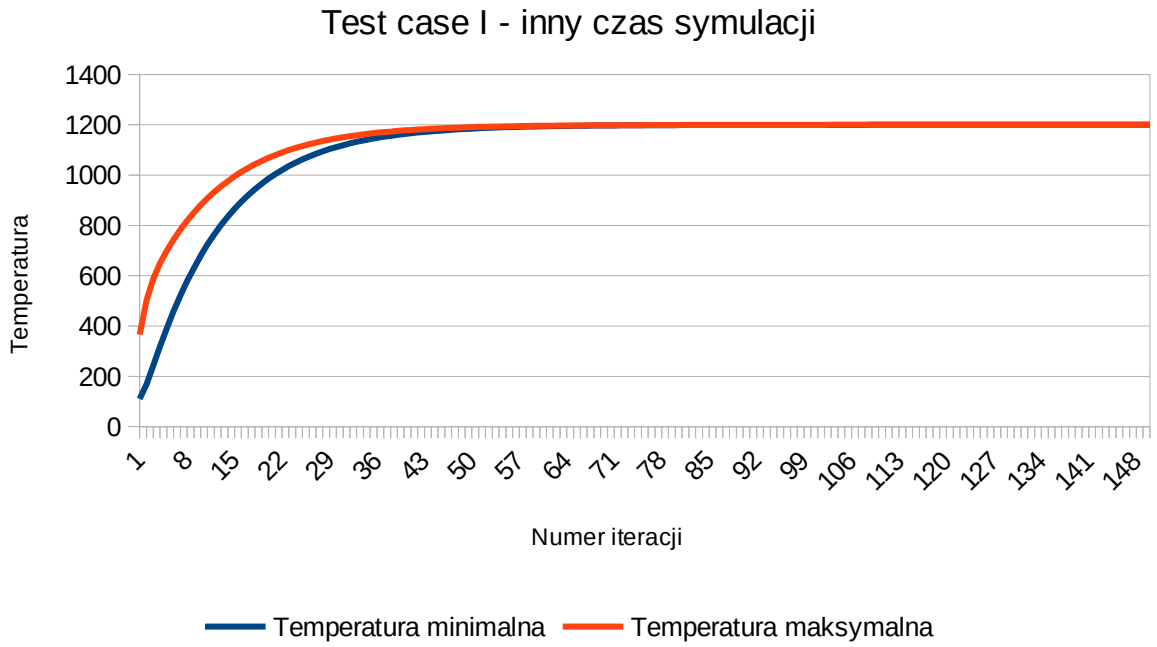
Time[s]	MinTemp	MaxTemp
1	100	149.56
2	100	177.44
3	100	197.27
4	100	213.15
5	100	226.68
6	100	238.61
7	100	249.35
8	100	259.17
9	100	268.24
10	100	276.7
11	100	284.64
12	100	292.13
13	100	299.24
14	100.01	306
15	100.01	312.45
16	100.01	318.63
17	100.02	324.56
18	100.03	330.27
19	100.05	335.77
20	100.06	341.08

Wykres 3.



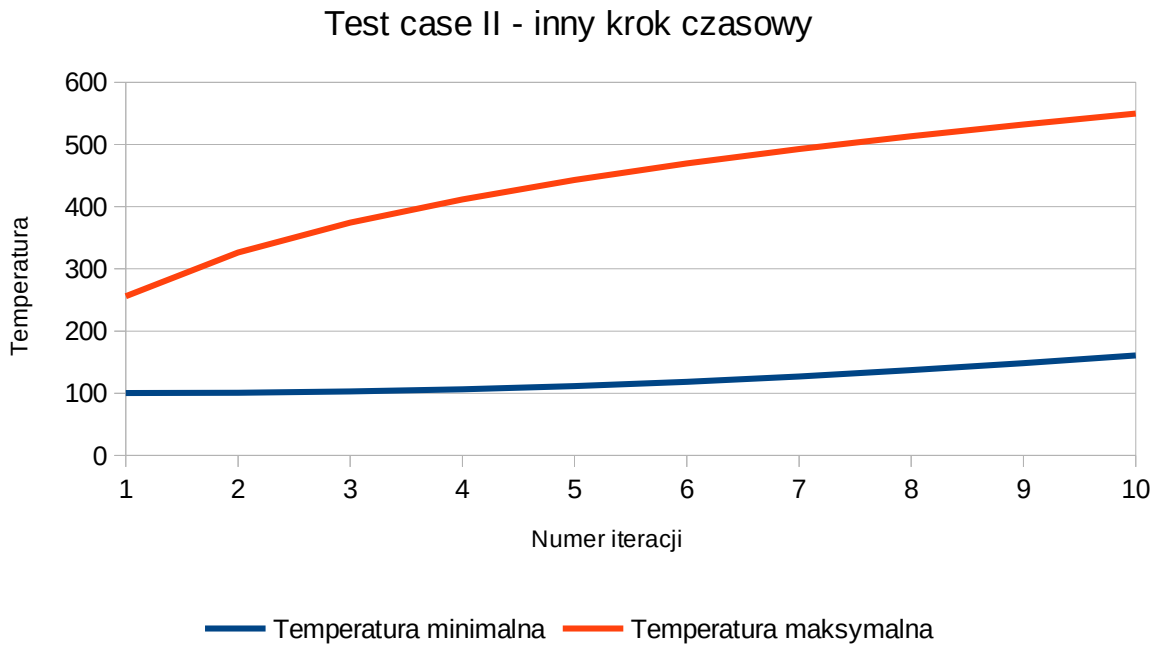
Powyższy wykres przedstawia test case III utworzony przeze mnie(plik „mes3.txt”)
Siatka oraz każdy element jest w tym przypadku prostokątny.

Wykres 4.



Wykres 4. ukazuje test case I ze zmienionym czasem symulacji na 7500 sekund.

Wykres 5.



Wykres 5. ukazuje test case II ze zmienionym krokiem czasowym na 10.

Podsumowanie

- Program wykorzystuje metodę elementów skończonych.
- Klasa Node oraz Element służą do stworzenia obiektów tworzących siatkę.
- Grid jest moją główną klasą, w której tworzę siatkę oraz mam pętle po czasie, elementach, gdzie robie wszystkie agregacje i obliczam rozwiązanie równania macierzowego.
- UniversalElement jest klasą, w której wykonuje wszystkie obliczenia związane z całkowaniem macierzy H, C, HBC oraz wektoraP.

Wnioski

- Na podstawie pierwszych 3 wykresów temperatur z każdego test casu mogę zauważyć, że temperatura zarówno minimalna, jak i maksymalna stale rośnie co świadczy o prawidłowości rozwiązania.
- Wykres 3. pokazuje, że program działa poprawnie również dla prostokątnych elementów.
- Na wykresie 4. mogę zauważyć, że temperatury na początku szybko rosną, a następnie zbliżają się do temperatury otoczenia równej 1200 stopni C oraz na tej temperaturze wyniki się zatrzymują, czyli wszystko działa poprawnie– temperatura w węzłach nie może być wyższa niż temperatura otoczenia.
- Wykres 5. przedstawia test case II ze zmienionym krokiem czasowym, jednak z tym samym czasem symulacji – wykres jest podobny do wykresu przy kroku = 1, jednak większą dokładność zauważam na wykresie 2.
- Między test casem I, a II widać wyraźną różnicę w czasie wykonania – jest to spowodowane dużo większą siatką, przez co dużo większą ilością obliczeń. Z czasem wykonania jest też bardzo powiązane oczywiście samo rozwiązanie równania macierzowego, co również spowalnia program.