Wydział Informatyki i Sztucznej Inteligencji

LAB2a: Budowa modelu bazowego "do przodu" - implementacja algorytmu dla jedno- i dwu-wymiarowego, stacjonarnego przypadku transferu ciepła w materiale niejednorondym

dr inż. Konrad M. Gruszka,*

4 marca 2025

Streszczenie

Ten dokument stanowi instrukcję do wykonania ćwiczenia z zagadnienia transferu ciepła w jednowymiarowym, stacjonarnym przypadku dla materiału niejednorodnego, do przedmiotu Rozwiązywanie Zadań Odwrotnych. Bazując na niniejszym opisie należy zaprojektować i zaimplementować algorytmy MRS w oparciu o kryteria przedstawione w dalszej części tego dokumentu.

1 Metoda różnic skończonych - MRS

Wprowadzenie
Wraz z tym dokumentem otrzymali Państwo również załącznik: opis metody MRS w przypadku jednowymiarowym, niejednorodnym dla stajonarnego problemu transferu ciepła. Należy dokładnie zapoznać się z tym dokumentem, gdyż stanowi on bazę matematyczną, konieczną do rozwiązania postawionego przed Państwem zadania.

Cel ćwiczenia
Celem ćwiczenia jest **zaprojektowanie i implementacja algorytmów rozwiązującego problem jedno- i dwu-wymiarowego stacjonarnego transferu ciepła w materiale niejednorodnym przy użyciu metody różnic skończonych** w języku Python. Efektem końcowym jest skrypt, który pozwoli na analizę rozkładu temperatury wzdłuż niejednorodnego pręta oraz w niejednorodnym materiale 2D.

Środowisko pracy

- Python w wersji > 3
- Visual Studio Code z dodatkiem Jupyter Notebook

Biblioteki

- numpy
- matplotlib
- time

Wymagania funkcjonalne algorytmu 1D

1. Definicja ilości węzłów siatki: Algorytm musi pozwalać użytkownikowi na zdefiniowanie ilości węzłów siatki N.

*Katedra Informatyki, Wydział Informatyki i Sztucznej Inteligencji (kgruszka@icis.pcz.pl)

2. Warunki brzegowe: Użytkownik musi mieć możliwość zdefiniowania temperatur na obu końcach 1-wymiarowego pręta.
3. Użytkownik ma mieć możliwość zdefiniowania rozkładu różnych współczynników k w symulacji.
4. Iteracyjne ustalanie temperatury:
   - Kryterium zakończenia iteracji ma być zbieżność rozwiązań, tj. zmiana temperatur w kolejnych iteracjach musi być poniżej zadanej tolerancji.
   - Użytkownik ma mieć możliwość wyboru kryterium zakończenia iteracji (po całkowitej maksymalnej liczbie iteracji lub po osiągnięciu zbieżności obliczeniowej.
5. Wizualizacja wyników: temperatury w poszczególnych węzłach powinny być wizualizowane na wykresie, na którym oś X reprezentuje numer węzła, a oś Y temperaturę (matplotlib).
6. Skrypt ma umożliwić wywołanie sumulacji z linii poleceń wraz z podaniem parametrów symulacji oraz bezpośrednio z VSCode.

Wydajność obliczeniowa W celu ustalenia metryk wydajności obliczeniowej oraz ustalenia, w jaki sposób dobór parametrów empirycznych symulacji wpływa na całkowity czas wykonania należy zaimplementować:

– Pomiar czasu wykonania algorytmu: należy zmierzyć czas wykonania algorytmu od rozpoczęcia do zakończenia pojedynczej iteracji.

– Pomiar całkowitego czasu wykonania wszystkich iteracji do uzyskania zbieżności lub zakończenia po ustalonej ilości iteracji.

– Dane o wydajności: po zakończeniu iteracji należy wypisać, po ilu iteracjach osiągnięto zbieżność dla zadanej tolerancji.

Wymagania funkcjonalne algorytmu 2D
Większość wymagań pokrywa się z tymi dla symulacji 1D, poniżej wymagadnia dodatkowe:

- Wizualizacja wyników: temperatury w poszczególnych węzłach powinny być wizualizowane na dwuwymiarowym wykresie, na którym kolor reprezentuje otrzymaną temperaturę.
- W przypadku 2D również należy określić rokład parametru k w 2D

Zadania do wykonania oraz do implementacji algorytmu w Pythonie

1. Przygotuj funkcję ‘simulate_heat_transfer_nonhomogeneous1D(N, T0, TN, max_iter, tolerance=None)‘, gdzie:
   – ‘N‘ to liczba węzłów siatki,
   – ‘T0‘, ‘TN‘ to temperatury na końcach pręta,
   – ‘max_iter‘ to maksymalna liczba iteracji po której pętla zakończy działanie,
   – ‘tolerance‘ to tolerancja zbieżności.
   – Funkcja ta powinna zwracać listę temperatur (temperaturę dla każdego węzła).
   – Funkcja musi korzystać z tablicy rozkładu parametru k (generowanej z użyciem np.array())
2. Przygotuj funkcję ‘simulate_heat_transfe_nonhomogeneous2D(Nx, Ny, TU, TD, TL, TR, max_iter, tolerance=None)‘, gdzie:
   – ‘Nx‘ oraz ‘Ny‘ to liczba węzłów siatki, w odpowiednio osi X i osi Y,
   – ‘TU‘, ‘TD‘, ‘TL‘, ‘TR‘ to temperatury na brzegach materiału (TU-Up, TD-Down, TL-Left, TR-Right),
   – ‘max_iter‘ to maksymalna liczba iteracji,
   – ‘tolerance‘ to tolerancja zbieżności.
   – Funkcja musi korzystać z tablicy rozkładu parametru k (generowanej z użyciem np.array())

   - Funkcja ta powinna zwracać listę temperatur (temperaturę dla każdego węzła w 2D).

3. Testowanie i dokumentacja:
   - Przetestuj algorytm dla różnych wartości 'N', 'T_0', 'T_N', 'max_iter', 'tolerance' oraz różnych współczynników k.
   - Dokumentujcie każdy test oraz wyniki.

---

2 Forma i ocena wykonania ćwiczenia

Forma oddania zadania
Zadanie należy oddać w formie pliku źródłowego zawierającego implementację algorytmu oraz wygenerowane wykresy wraz z opisami
Należy przeprowadzić test dla następujących parametrów (1D):

- Liczba węzłów N: 50
- $k_{al} = 237.0$ [W/m·K] (aluminium)
- $k_w = 0.12$ [W/m·K] (drewno)
- $T_0 = 150$
- $T_N = 50$
- $tolerancja = 0.01K$
- $max\_iter = 10000$

Należy przeprowadzić test dla następujących parametrów (2D):

- Liczba węzłów Nx: 30, Ny:30
- $k_{al} = 237.0$ [W/m·K] (aluminium)
- $k_w = 0.12$ [W/m·K] (drewno)
- $T_U = 300$, $T_D = 100$
- $T_L = 200$, $T_R = 0$
- $tolerancja = 0.01K$
- $max\_iter = 10000$

Ponadto w raporcie (np .ipynb, .pdf) należy umieścić następujące informacje:

- Udokumentowanie przeprowadzonego testu dla parametrów podanych powyżej, wyniki oraz odpowiednie komentarze i wyjaśnienia.
- Uzyskane czasy trwania pojedynczej iteracji i całkowitego czasu trwania wszystkich iteracji w odniesieniu do parametrów max_iter oraz dla algorytmu z tolerancją.
- Wykres przedstawiający rokład temperatur w pręcie.
- Wykres przedstawiający rokład temperatur w 2D.

Ocena ćwiczenia będzie bazować na:

1. Poprawności implementacji algorytmu.
2. Kompletności przeprowadzonych testów.
3. Jakości dokumentacji i wyjaśnień teoretycznych.
4. Efektywności i optymalizacji kodu.