Mateusz Bartnicki, grupa nr 3 środa 16:45-18:15

Metody obliczeniowe w nauce i technice, ćwiczenie 6 - 22.05.2024 r.

# Rozwiązywanie układów równań liniowych metodami bezpośrednimi

## 1. Opis ćwiczenia

Ćwiczenie to składa się z 3 zadań, w których rozwiązywane będą układy równań liniowych metodą eliminacji Gaussa oraz z wykorzystaniem algorytmu Thomasa. Zadania te są realizowane w taki sposób, że zadana jest macierz **A**. Początkowo, zakładam wektor **x** jako znany, jest on dowolną n−elementową permutację ze zbioru {1, -1}. Na tej podstawie obliczam wektor **b** (wyliczam to z równania postaci **Ax** = **b**). Gdy mam już wyliczony wektor **b**, „zapominam" o znajomości wektora **x** i próbuję go wyliczyć rozwiązując równanie tej samej postaci. Celem ćwiczenia jest zbadanie różnic pomiędzy z góry wyznaczonym wektorem **x** oraz tym obliczonym. Celem zadania 3 natomiast jest porównanie algorytmów Gaussa i Thomasa.

## 2. Dane techniczne

Program został napisany przy użyciu języka Python (3.10.12) z wykorzystaniem bibliotek numpy oraz matplotlib. Ćwiczenie zostało wykonane na WSL (Windows Subsystem for Linux) - Ubuntu 22.04.3 LTS na procesorze Intel Core i5-11400H 2.70GHz.

## 3. Kryterium pomiaru błędu

Kryterium, którym posłużyłem się aby wyznaczyć wartości błędów dla obliczonych wartości wektora **x** względem zadanego wektora **x** była norma maksimum, która opisana jest wzorem:

$\max_{i=1,\dots,n}\{|x_i - x_i'|\}$, gdzie $x_i$ to i – ta wartość zadanego wektora **x**, natomiast $x_i'$ to i – ta wartość obliczonego wektora **x**.

## 4. Zadanie 1

Elementy macierzy **A** o wymiarze *n* x *n* są określone wzorem:

$$\begin{cases} a_{1j} = 1 \\ a_{ij} = \frac{1}{i+j-1}\ dla\ i \neq 1 \end{cases} \qquad \text{i, j} = 1, 2, \dots, \text{n}$$

W tym zadaniu jako wektor **x** przyjąłem wektor postaci [1, -1, 1, -1, ...], a obliczenia przeprowadziłem dla wartości n z zakresu [2 – 25] odpowiednio dla precyzji float32 i float64.

| N | Wartość błędu w precyzji: | |
|---|---|---|
| | Float32 | Float64 |
| 2 | 0 | 0 |
| 3 | 2.0862e-06 | 0 |
| 4 | 2.6822e-06 | 1.4932e-13 |
| 5 | 0.002 | 6.4275e-12 |
| 6 | 0.0933 | 2.5183e-10 |
| 7 | 0.9195 | 6.8303e-09 |
| 8 | 1.9272 | 5.5975e-08 |
| 9 | 15.2125 | 4.4172e-07 |
| 10 | 7.5025 | 0.0001 |
| 11 | 17.0151 | 0.0078 |
| 12 | 5.7873 | 0.6846 |
| 13 | 3.4012 | 14.0699 |
| 14 | 9.2875 | 15.2886 |
| 15 | 7.4101 | 10.2914 |
| 16 | 25.7688 | 17.2375 |
| 17 | 98.6097 | 17.1994 |
| 18 | 64.1519 | 16.6886 |
| 19 | 22.1518 | 61.6105 |
| 20 | 19.5204 | 24.0810 |
| 21 | 22.4629 | 37.3366 |
| 22 | 29.6309 | 27.1942 |
| 23 | 1658.9425 | 19.1262 |
| 24 | 50.9217 | 22.5087 |
| 25 | 10.9018 | 21.4118 |

Tabela 1 – Wartości błędów dla kolejnych n dla precyzji float32 i float64 dla macierzy z zadania 1

Analizując otrzymane wartości błędów zamieszczone w tabeli 1 widzimy, że dla precyzji float32 początkowe wartości błędów są niewielkie. Dla n = 7 błąd już prawie sięga wartości 1, co oznacza, że wartości obliczonego wektora **x** mogą sporo różnić się od wektora zadanego. Dla kolejnych n wartości błędów wahają się w okolicach 1-20, a dla n > 16 błędy sięgają nawet rzędów setek i tysięcy. Dla precyzji float64 można dostrzec spodziewany efekt – obliczenia są dokładniejsze (wartości błędów są mniejsze), a pierwszy błąd, który sięga okolic liczby 1 otrzymujemy dopiero dla n = 12. Dla kolejnych wartości n, tak samo jak dla precyzji float32, błędy są rzędu $10^2$. Ciekawym spostrzeżeniem jest to, że dla n z zakresu [13, 25] zdarzają się przypadki, że mniejsze błędy uzyskujemy dla precyzji float32 (np. dla n = 13). Nie należy się tym jednak sugerować, gdyż jest to po prostu przypadek, który wynika z arytmetyki komputerowej.

## 5. Zadanie 2

W tym zadaniu postępuję tak samo jak w przypadku zadania 1 (przyjmuję również ten sam wektor **x**), jednak tym razem wzór macierzy **A** wygląda następująco:

$$\begin{cases} a_{ij} = \frac{2i}{j} \; dla \; j \geq i \\ a_{ij} = a_{ji} \; dla \; j < i \end{cases} \quad \text{i, j} = 1, 2, \ldots, \text{n}$$

| N | Wartość błędu dla precyzji: | |
|---|---|---|
| | Float32 | Float64 |
| 2 | 0 | 0 |
| 3 | 5.9605e-08 | 2.2204e-16 |
| 4 | 1.1921e-07 | 1.1102e-16 |
| 5 | 1.1921e-07 | 3.3307e-16 |
| 6 | 1.7881e-07 | 6.6613e-16 |
| 7 | 1.2517e-06 | 6.6613e-16 |
| 8 | 8.9407e-07 | 2.4425e-15 |
| 9 | 8.9407e-07 | 1.1102e-15 |
| 10 | 2.4438e-06 | 1.5543e-15 |
| 11 | 2.6822e-06 | 2.4425e-15 |
| 12 | 5.1260e-06 | 1.1768e-14 |
| 13 | 6.6161e-06 | 1.2212e-14 |
| 14 | 7.2122e-06 | 1.2434e-14 |
| 15 | 7.4506e-06 | 1.6209e-14 |
| 16 | 4.1723e-06 | 2.0206e-14 |
| 17 | 4.6492e-06 | 2.0206e-14 |
| 18 | 4.8876e-06 | 2.0206e-14 |
| 19 | 5.6028e-06 | 2.1760e-14 |
| 20 | 7.3314e-06 | 1.9540e-14 |
| 21 | 9.3579e-06 | 1.9540e-14 |
| 22 | 1.0550e-05 | 2.6201e-14 |
| 23 | 1.1742e-05 | 2.1982e-14 |
| 24 | 1.3173e-05 | 1.7986e-14 |
| 25 | 1.2100e-05 | 1.7319e-14 |
| 50 | 5.54323e-05 | 1.1324e-13 |
| 100 | 0.00029 | 5.95301e-13 |
| 200 | 0.00127 | 5.50338e-12 |
| 500 | 0.03446 | 8.36522e-11 |
| 1000 | 0.231 | 5.02786e-10 |
| 2500 | 2.104 | 7.53647e-09 |

Tabela 2 – Wartości błędów dla kolejnych n dla precyzji float32 i float64 dla macierzy z zadania 2

| N | Wartość błędu dla precyzji: | | | |
|---|---|---|---|---|
| | Float32 | | Float64 | |
| | Macierz nr 1 | Macierz nr 2 | Macierz nr 1 | Macierz nr 2 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 2.0862e-06 | 5.9605e-08 | 0 | 2.2204e-16 |
| 4 | 2.6822e-06 | 1.1921e-07 | 1.4932e-13 | 1.1102e-16 |
| 5 | 0.002 | 1.1921e-07 | 6.4275e-12 | 3.3307e-16 |
| 6 | 0.0933 | 1.7881e-07 | 2.5183e-10 | 6.6613e-16 |
| 7 | 0.9195 | 1.2517e-06 | 6.8303e-09 | 6.6613e-16 |
| 8 | 1.9272 | 8.9407e-07 | 5.5975e-08 | 2.4425e-15 |
| 9 | 15.2125 | 8.9407e-07 | 4.4172e-07 | 1.1102e-15 |
| 10 | 7.5025 | 2.4438e-06 | 0.0001 | 1.5543e-15 |
| 11 | 17.0151 | 2.6822e-06 | 0.0078 | 2.4425e-15 |
| 12 | 5.7873 | 5.1260e-06 | 0.6846 | 1.1768e-14 |
| 13 | 3.4012 | 6.6161e-06 | 14.0699 | 1.2212e-14 |
| 14 | 9.2875 | 7.2122e-06 | 15.2886 | 1.2434e-14 |
| 15 | 7.4101 | 7.4506e-06 | 10.2914 | 1.6209e-14 |
| 16 | 25.7688 | 4.1723e-06 | 17.2375 | 2.0206e-14 |
| 17 | 98.6097 | 4.6492e-06 | 17.1994 | 2.0206e-14 |
| 18 | 64.1519 | 4.8876e-06 | 16.6886 | 2.0206e-14 |
| 19 | 22.1518 | 5.6028e-06 | 61.6105 | 2.1760e-14 |
| 20 | 19.5204 | 7.3314e-06 | 24.0810 | 1.9540e-14 |
| 21 | 22.4629 | 9.3579e-06 | 37.3366 | 1.9540e-14 |
| 22 | 29.6309 | 1.0550e-05 | 27.1942 | 2.6201e-14 |
| 23 | 1658.9425 | 1.1742e-05 | 19.1262 | 2.1982e-14 |
| 24 | 50.9217 | 1.3173e-05 | 22.5087 | 1.7986e-14 |
| 25 | 10.9018 | 1.2100e-05 | 21.4118 | 1.7319e-14 |

Tabela 3 – Zestawienie wartości błędów dla kolejnych n dla precyzji float32 i float64 dla różnych macierzy

W przypadku macierzy z zadania 2, otrzymane błędy są o wiele mniejsze. Dla precyzji float32 dla n <= 25 są one nie większe niż $10^{-5}$, co mówi nam o tym, że otrzymane wyniki są naprawdę zadowalające. Dla wyższych n błędy również nie są duże, w tabeli 2 jedyny błąd większy od 1 jest dla n = 2500. Analizując wartości błędów z tabeli 2, dochodzimy do tego samego wniosku, co w zadaniu 1 – zastosowanie precyzji float64 znacząco zwiększa dokładność obliczeń, a tym samym wartości otrzymanych błędów są dużo mniejsze. W tym przypadku nie przekraczają one liczby $10^{-9}$. W tabeli 3, które jest zestawieniem wartości błędów z macierzy z zadania 1 i zadania 2, dostrzegamy, jak duże znaczenie ma uwarunkowanie macierzy na wyniki.

Aby uzasadnić tak duże różnice w błędach pomiędzy tabelą 1 a tabelą 2, postanowiłem policzyć wskaźnik uwarunkowania obydwóch macierzy, który wyliczony został przy pomocy funkcji *cond* z biblioteki numpy.

| N | Wartość wskaźnika uwarunkowania macierzy z zadania: | |
|---|---|---|
| | Nr 1 | Nr 2 |
| 2 | 12.084 | 1.8866 |
| 3 | 392 | 3.1063 |
| 4 | 11247 | 4.5403 |
| 5 | 283662 | 6.1156 |
| 6 | 1.4638e+07 | 7.7990 |
| 7 | 4.5393e+08 | 9.5698 |
| 8 | 1.3372e+10 | 11.4135 |
| 9 | 5.3898e+11 | 13.3195 |
| 10 | 1.5537e+13 | 15.2795 |
| 11 | 5.4700e+14 | 17.2871 |
| 12 | 2.0354e+16 | 19.3371 |
| 13 | 3.7985e+17 | 21.4251 |
| 14 | 4.5003e+17 | 23.5476 |
| 15 | 5.3574e+17 | 25.7016 |
| 16 | 6.2255e+17 | 27.8844 |
| 17 | 1.0451e+18 | 30.0939 |
| 18 | 5.6631e+18 | 32.3280 |
| 19 | 1.5766e+19 | 34.5850 |
| 20 | 3.0858e+18 | 36.8635 |
| 21 | 4.1564e+18 | 39.1621 |
| 22 | 3.7920e+18 | 41.4795 |
| 23 | 3.1146e+18 | 43.8148 |
| 24 | 2.6921e+18 | 46.1668 |
| 25 | 2.6588e+18 | 48.5348 |

Tabela 4 – Wartość wskaźnika uwarunkowania macierzy zadanej w danym zadaniu

Jak widać na powyższej tabeli, dla macierzy zadanej w zadaniu 2 wartości wskaźnika uwarunkowania macierzy są o wiele niższe niż w przypadku zadania 1, bo dla n = 4 jest to już różnica aż 4 rzędów wielkości, a dla kolejnych n wartość wskaźnika nadal drastycznie rośnie. Oznacza to, że wykonywane obliczenia są o wiele stabilniejsze numerycznie, a więc otrzymywane są mniejsze wartości błędów, co tłumaczy tak duże rozbieżności pomiędzy tabelą 1 i tabelą 2.

## 6. Zadanie 3

(Wszystkie obliczenia w tym zadaniu wykonywane były tylko na precyzji float64).

W zadaniu 3, zadana macierz prezentuje się następująco: (parametry k = 7, m = 4)

$$\begin{cases} a_{i,i} = k \\ a_{i,i+1} = \frac{1}{i+m} \\ a_{i,i-1} = \frac{k}{i+m+1}\ dla\ i > 1 \\ a_{i,j} = 0\ dla\ j < i-1\ oraz\ j > i+1 \end{cases} \qquad \text{i, j} = 1, 2, \ldots, \text{n}$$

W moim programie wykorzystującym algorytm Thomasa macierz tę przedstawiłem jako 3 wektory **a**, **b** i **c**.

Zadanie to polegało na zestawieniu ze sobą algorytmu eliminacji Gaussa oraz algorytmu Thomasa. W tym celu, na samym początku, porównałem ze sobą wartości otrzymanych błędów.

| N | Wartość błędu dla algorytmu: | |
|---|---|---|
| | Gaussa | Thomasa |
| 2 | 0 | 0 |
| 3 | 0 | 0 |
| 4 | 2.2204e-16 | 0 |
| 5 | 2.2204e-16 | 0 |
| 6 | 2.2204e-16 | 1.1102e-16 |
| 7 | 2.2204e-16 | 2.2204e-16 |
| 8 | 2.2204e-16 | 1.1102e-16 |
| 9 | 2.2204e-16 | 1.1102e-16 |
| 10 | 2.2204e-16 | 1.1102e-16 |
| 11 | 2.2204e-16 | 1.1102e-16 |
| 12 | 2.2204e-16 | 1.1102e-16 |
| 13 | 2.2204e-16 | 1.1102e-16 |
| 14 | 2.2204e-16 | 1.1102e-16 |
| 15 | 2.2204e-16 | 1.1102e-16 |
| 16 | 2.2204e-16 | 1.1102e-16 |
| 17 | 2.2204e-16 | 1.1102e-16 |
| 18 | 2.2204e-16 | 1.1102e-16 |
| 19 | 2.2204e-16 | 1.1102e-16 |
| 20 | 2.2204e-16 | 2.2204e-16 |
| 21 | 2.2204e-16 | 1.1102e-16 |
| 22 | 2.2204e-16 | 2.2204e-16 |
| 23 | 2.2204e-16 | 1.1102e-16 |
| 24 | 2.2204e-16 | 1.1102e-16 |
| 25 | 2.2204e-16 | 1.1102e-16 |
| 50 | 2.2204e-16 | 1.1102e-16 |
| 100 | 2.2204e-16 | 1.1102e-16 |
| 250 | 2.2204e-16 | 2.2204e-16 |
| 500 | 2.2204e-16 | 2.2204e-16 |
| 1000 | 2.2204e-16 | 2.2204e-16 |
| 2500 | 2.2204e-16 | 2.2204e-16 |

Tabela 5 – Wartości błędów w normie maksimum dla algorytmów Gaussa i Thomasa

Na powyższej tabeli widać, że wartości tych błędów dla analizowanych n są praktycznie identyczne nawet dla dużych wartości n, a jak występują różnice, to dopiero na 16 miejscu po przecinku, a więc są one pomijalne. Wniosek stąd płynie taki, że otrzymane wyniki są niemalże identyczne. Płynie to stąd, że algorytm Thomasa oparty jest o algorytm Gaussa – jest on tylko jego usprawnieniem, zarówno czasowym, jak i pamięciowym.

Aby sprawdzić, czy algorytm Thomasa jest faktycznie wydajniejszy czasowo, postanowiłem przeprowadzić testy.

| N | Czas [w sekundach] wykonywania algorytmu: | |
|---|---|---|
| | Gaussa | Thomasa |
| 2 | 7.057e-05 | 6.675e-06 |
| 5 | 0.00011 | 6.437e-06 |
| 10 | 0.00023 | 1.311e-05 |
| 50 | 0.0038 | 4.887e-05 |
| 100 | 0.048 | 0.0002 |
| 200 | 0.124 | 0.00036 |
| 500 | 0.489 | 0.001 |
| 1000 | 1.81 | 0.002 |
| 1500 | 4.27 | 0.0036 |
| 2500 | 15.43 | 0.0067 |
| 5000 | 86.94 | 0.016 |

Tabela 6 – Wartość wskaźnika uwarunkowania macierzy zadanej w danym zadaniu

Próbowałem również obliczyć wektor **x** dla n = 10000, jednak udało mi się to tylko z wykorzystaniem algorytmu Thomasa, który wykonał się w 0,02s. Potwierdza się więc hipoteza, że algorytm Thomasa jest wydajniejszy czasowo. Złożoność tego algorytmu wynosi O(n), podczas gdy złożoność czasowa algorytmu Gaussa wynosi $O(n^3)$.

Algorytm Thomasa również przeważa nad algorytmem Gaussa pod względem wykorzystanej pamięci. Algorytm Gaussa wymaga zapisania całej macierzy (a więc *n* x *n* komórek) oraz wektorów **x** i **b**, każdy długości n. Natomiast w implementacji algorytmu Thomasa dla zapisu macierzy wystarczy użyć 3 wektorów długości n (oczywiście również trzeba zapisać wektory **x** i **b**).

## 7. Wnioski

- Na dokładność obliczeń wpływa nie tylko precyzja, ale równie ważne jest zwrócenie uwagi na uwarunkowanie macierzy.
- Metoda Gaussa jest uniwersalna i pozwala rozwiązywać wszystkie układy równań liniowych (które są rozwiązywalne), jednak ma wysoką złożoność czasową i pamięciową.
- Jeśli wiemy, że w rozwiązywanym równaniu **Ax** = **b**, macierz A jest macierzą trójdiagonalną, warto jest zastosować algorytm Thomasa, który jest o wiele szybszy i wymaga użycia mniejszej ilości pamięci.