MOwNiT – Rozwiązywanie układów równań liniowych metodami bezpośrednimi

Przygotował:

Maksymilian Zawiślak

Dany jest układ równań liniowych $Ax = b$. Macierz A składa się z elementów wyznaczanych na podstawie wzoru w każdym z 3 zadań. Za wektor x przyjmowana jest dowolna n-elementowa permutacja liczb ze zbioru $\{-1, 1\}$, a następnie, wykorzystując wektor x, wyznaczany jest wektor b. Następnie metodą eliminacji Gaussa rozwiązywany jest układ równań liniowych $Ax = b$ (przyjmując jako niewiadomą wektor x). Obliczenia powinny być przeprowadzone dla różnych precyzji liczbowych. Sprawdzić należy, jak błędy zaokrągleń zaburzają rozwiązania dla różnych rozmiarów układu.

Pomiar błędu

Jako kryterium pomiaru błędu został przyjęta maksymalna wartość różnicy miedzy i-tym elementem wyznaczonej macierzy x, a startową wartością x'.

$$\max_{i=1,\dots,n} \{|x_i - x_i'|\}$$

Wzór 1

gdzie:

- x_i – i-ta współrzędna otrzymanej macierzy
- x_i' – i-ta współrzędna startowej macierzy

Do przeprowadzania testów zostały wykorzystane dwa typy liczbowe Float32 oraz Float64 z biblioteki NumPy. Wektor x wyznaczany ze zbioru $\{-1, 1\}$, składał się z elementów tego zbioru występujących na przemian np. $x = [-1, 1, -1, 1, -1, \dots]$.

Zadanie 1

Macierz A w zadaniu 1 opisana jest w poniższy sposób. Próba w tym wariancie zadania będzie przeprowadzona na $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$

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

Wzór 2

Wartość n	Wartość błędu dla Float32	Wartość błędu dla Float64
2	0.0000e+00	0.0000e+00
3	2.0862e-06	0.0000e+00
4	6.4373e-05	0.0000e+00
5	2.0037e-03	2.8415e-12
6	6.0793e-02	3.3952e-11
7	**1.3677e+00**	2.0169e-09
8	4.5899e+00	5.3591e-08
9	4.1979e+00	1.8087e-06
10	6.8686e+00	1.0740e-05
11	5.7285e+00	2.9340e-04
12	1.0891e+01	3.7738e-02
13	1.1799e+02	4.0972e-01
14	3.4612e+00	2.4475e-01
15	1.5552e+00	7.2967e-01
16	1.7673e+00	**1.1510e+00**
17	3.1458e+00	1.0965e+00
18	5.2323e+00	1.8758e+00
19	6.8712e+00	1.1860e+01
20	4.3825e+00	4.7742e+01

Tabela 1: Wartości błędów w zadaniu 1

W tabeli 1 można zaobserwować że dla $n = 7$, dla typu Float32 wartość błędu przekracza wartość 1. Dla typu Float64 do takiej wartości błędu dochodzi dopiero dla $n = 16$. Brak błędu dla obu przypadków udało się uzyskać jedynie dla $n = 2$.

Zadanie 2

W zadaniu 2 macierz A opisana jest wzorem poniżej. W tym wariancie do wcześniejszych n zostaną dopisane następujące wartości: $\{30, 50, 100, 150, 200, 300, 400, 500\}$

$$\begin{cases} a_{1,j} = \frac{2i}{j}, \ dla\ j \geq i \\ a_{i,j} = \ a_{i,j}, dla\ j < i \end{cases} \quad \text{gdzie } i, j = 1, \dots, n$$

Wzór 3

Wartość n	Wartość błędu dla Float32	Wartość błędu dla Float64
2	0.0000e+00	0.0000e+00
3	1.1921e-07	2.2204e-16
4	1.1921e-07	2.2204e-16
5	1.1921e-07	2.2204e-16
6	2.3842e-07	2.2204e-16
7	1.4901e-06	3.3307e-16
8	1.2517e-06	1.8874e-15
9	1.2517e-06	1.5543e-15
10	2.9206e-06	4.4409e-15
11	2.9802e-06	6.2172e-15
12	2.7418e-06	6.2172e-15
13	3.3975e-06	4.8850e-15
14	6.1393e-06	7.7716e-15
15	4.8876e-06	8.6597e-15
16	5.9605e-06	7.9936e-15
17	6.2585e-06	7.7716e-15
18	7.5698e-06	7.9936e-15
19	6.3777e-06	7.5495e-15
20	5.7220e-06	7.9936e-15
30	1.2577e-05	2.9532e-14
50	4.3869e-05	8.8152e-14
100	1.9872e-04	5.4934e-13
150	5.4419e-04	1.0950e-12
200	1.3801e-03	1.6424e-12
300	3.1228e-03	4.9534e-12
400	6.0159e-03	7.6616e-12
500	8.4233e-03	1.1454e-11

Tabela 2: Wartości błędów w zadaniu 2

Jak widać w tabeli nawet dla $n = 500$, żaden z typów liczbowych nie przekracza wartość 1. Float64 radzi sobie bardzo dobrze dla tej macierzy A, utrzymując dokładność na poziomie $1e - 11$.

Porównanie wartości błędów dla obu układów

Wartość n	Float32		Flota64	
	Zadanie 1	**Zadanie 2**	**Zadanie 1**	**Zadanie 2**
2	0.0000e+00	0.0000e+00	0.0000e+00	0.0000e+00
3	2.0862e-06	1.1921e-07	0.0000e+00	2.2204e-16
4	6.4373e-05	1.1921e-07	0.0000e+00	2.2204e-16
5	2.0037e-03	1.1921e-07	2.8415e-12	2.2204e-16
6	6.0793e-02	2.3842e-07	3.3952e-11	2.2204e-16
7	**1.3677e+00**	1.4901e-06	2.0169e-09	3.3307e-16
8	4.5899e+00	1.2517e-06	5.3591e-08	1.8874e-15
9	4.1979e+00	1.2517e-06	1.8087e-06	1.5543e-15
10	6.8686e+00	2.9206e-06	1.0740e-05	4.4409e-15
11	5.7285e+00	2.9802e-06	2.9340e-04	6.2172e-15
12	1.0891e+01	2.7418e-06	3.7738e-02	6.2172e-15
13	1.1799e+02	3.3975e-06	4.0972e-01	4.8850e-15
14	3.4612e+00	6.1393e-06	2.4475e-01	7.7716e-15
15	1.5552e+00	4.8876e-06	7.2967e-01	8.6597e-15
16	1.7673e+00	5.9605e-06	**1.1510e+00**	7.9936e-15
17	3.1458e+00	6.2585e-06	1.0965e+00	7.7716e-15
18	5.2323e+00	7.5698e-06	1.8758e+00	7.9936e-15
19	6.8712e+00	6.3777e-06	1.1860e+01	7.5495e-15
20	4.3825e+00	5.7220e-06	4.7742e+01	7.9936e-15
30	1.6163e+01	1.2577e-05	5.6358e+00	2.9532e-14
50	2.6953e+03	4.3869e-05	7.1573e+00	8.8152e-14
100	2.2274e+02	1.9872e-04	2.1625e+01	5.4934e-13
150	4.0610e+01	5.4419e-04	7.4580e+01	1.0950e-12
200	5.7358e+01	1.3801e-03	4.7698e+01	1.6424e-12
300	3.7020e+01	3.1228e-03	6.9383e+01	4.9534e-12
400	7.8775e+01	6.0159e-03	8.0703e+01	7.6616e-12
500	7.8699e+01	8.4233e-03	2.9129e+01	1.1454e-11

Tabela 3: Porównanie wartości błędów miedzy zadaniem 1 oraz zdaniem 2

Jak widać w tabeli 3 dla zadania 1 dla Float32 oraz Float64 od odpowiednio $n = 7$ oraz $n = 16$ wartość błędu jest większa od 1 i wraz ze wzrostem parametru n stale rośnie. Dla zadania 2 w testowanym zakresie n nie dochodzi to takiej sytuacji. Różnica miedzy układami powoduje takie wartości błędów.

Wyznaczanie współczynnika uwarunkowania

Współczynnik uwarunkowania określa, w jakim stopniu błąd reprezentacji danych wejściowych wpływa na błąd wyniku. Jeżeli wskaźnik uwarunkowania dla danego problemu jest duży, wówczas nawet niewielki błąd danych może spowodować otrzymanie nie poprawnego wyniku. Wskaźnik uwarunkowania oblicza się ze wzoru:

$$\kappa(A) = \|A^{-1}\| \cdot \|A\|$$

Wzór 4

Macierz odwrotności była obliczana przy pomocy funkcji np.linalg.inv() z biblioteki NumPy.

Norma jest określona poniższym wzorem:

$$\|A\| = \max_{1 < i < n} \sum_{j=1}^{n} |a_{i,j}|$$

Wzór 5

Współczynnik uwarunkowania został sprawdzony dla macierzy A z zadania 1 oraz 2. Dla $n \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 50, 100, 150, 200, 300, 400, 500\}$. Do obliczeń został wykorzystany typ Float64.

Wartość n	**Zadanie 1**	**Zadanie 2**
2	8.0000e+00	1.0000e+00
3	2.1600e+02	1.4444e+00
4	2.8800e+03	1.8333e+00
5	2.8000e+04	2.2333e+00
6	2.2680e+05	2.6444e+00
7	1.6299e+06	3.0317e+00
8	1.2862e+07	3.4484e+00
9	1.1200e+08	3.8492e+00
10	8.8420e+08	4.2492e+00
11	6.4926e+09	4.6594e+00
12	4.7683e+10	5.0552e+00
13	1.2125e+12	5.4655e+00
14	1.5227e+11	5.8689e+00
15	9.1102e+10	6.2689e+00
16	2.8441e+11	6.6784e+00
17	1.2449e+12	7.0776e+00
18	4.3153e+11	7.4850e+00
19	3.8357e+11	7.8896e+00
20	4.3982e+11	8.2896e+00
30	1.2472e+12	1.2332e+01
50	1.4475e+13	2.0421e+01
100	1.2819e+14	4.0639e+01
150	9.3439e+15	6.0856e+01
200	3.2372e+16	8.1073e+01
300	2.3017e+17	1.2151e+02
400	3.7425e+17	1.6194e+02
500	8.9385e+18	2.0238e+02

Tabela 4: Obliczone współczynniki uwarunkowania

Z tabeli 4 widać że macierz z zadania 1 jest znacznie gorzej uwarunkowana, nawet niewielki błąd danych wejściowych może znacznie wpłynąć na poprawność wyniku.

Wnioski

Wyniki zadania 1 oraz 2 pokazują jak kluczowe dla poprawność rozwiązania układu równań liniowych jest uwarunkowanie problemu. Różnice miedzy wartościami błędów dla obu układów z tabeli 3 dokładnie to pokazują. Ważnym czynnikiem, który może powodować błędy obliczeń jest też arytmetyka komputerowa oraz zastosowanie odpowiedniego typu liczbowego. Metoda Gaussa nie jest idealną metoda rozwiązywania układów równań, lecz jest łatwa, uniwersalna i dla dobrze uwarunkowanego układu oraz przy zastosowaniu odpowiedniej reprezentacji liczb daje zadowalające wyniki.

Zadanie 3

Macierz A jest zadana następującym wzorem. Tutaj została porównana skuteczność obliczeniowa oraz czasowa metody Gaussa oraz metody Thomasa.

$$\begin{cases} a_{1,j} = k \\ a_{i,i+1} = \frac{1}{i+m} \\ a_{i,i-1} = \frac{k}{1+m+i} \text{ dla } i > 1 \\ a_{i,j} = \frac{1}{i+j-1} \text{ dla } j < i-1 \text{ oraz } j > i+1 \end{cases} \quad \text{gdzie } i, j = 1, \dots, n$$

Wzór 6

Parametry do wzoru to $k = 6$ oraz $m = 5$.

Metoda Thomasa

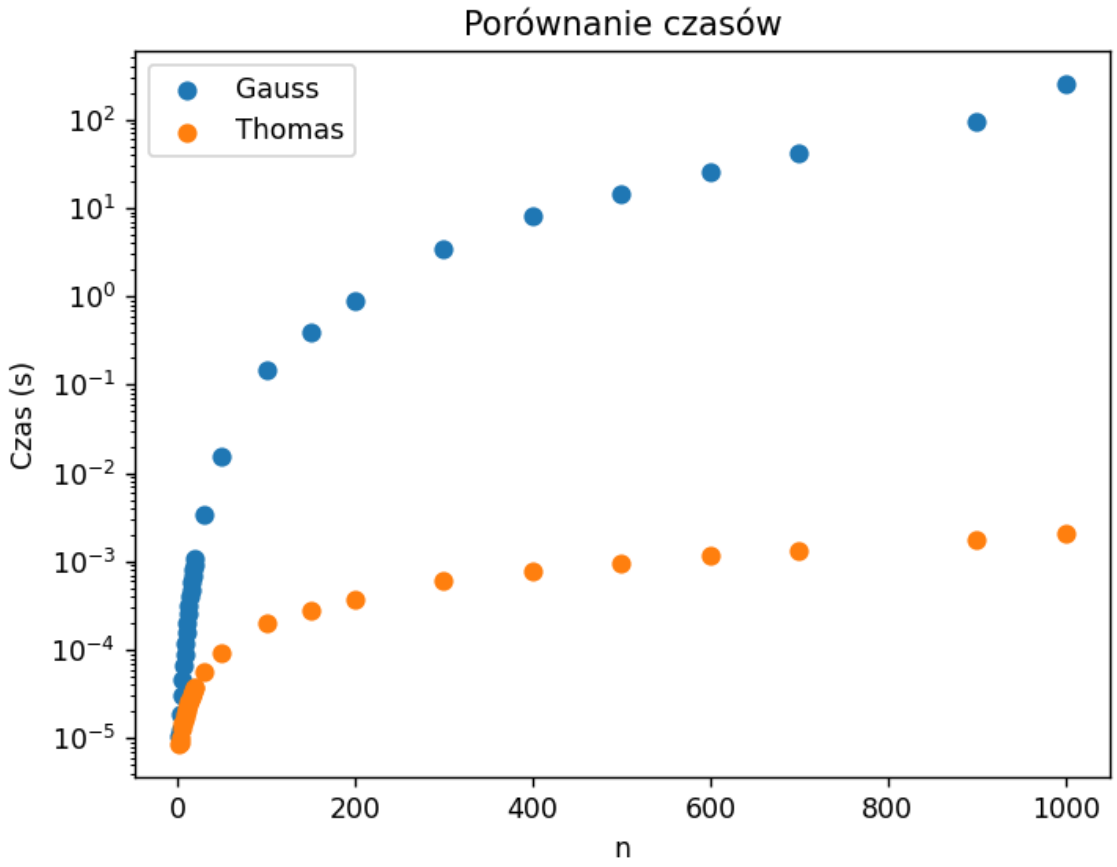
Algorytm Thomasa działa analogicznie do eliminacji Gausa. Metoda wykorzystuje fakt, iż poza trzema głównymi przekątnymi pozostałe wartości w macierzy są zerami. Algorytm operuje jedynie na wartości głównej przekątnej oraz wartości przekątnej pod nią. Powoduje to że algorytm ma złożoność $O(n)$, gdzie eliminacja Gaussa ma złożoność $O(n^3)$. Algorytm w tym rozwiązaniu otrzymał cała macierz A. Operował na dwóch dodatkowych wektorach C oraz D rozmiaru n.

W zadaniu 3 testowane rozmiary macierzy to: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 50, 100, 150, 200, 300, 400, 500, 600, 700, 900, 1000. W następującej tabeli zostały przedstawione wartości błędu oraz czas obliczeń dla metody Gaussa oraz metody Thomasa. Do obliczeń został wykorzystany typ Float64.

Wartość n	Metoda Gaussa		Metoda Thomsona	
	Błąd	**Czas (s)**	**Błąd**	**Czas (s)**
2	0.0000e+00	8.9000e-06	0.0000e+00	8.4000e-06
3	1.1102e-16	1.0600e-05	1.1102e-16	8.7000e-06
4	1.1102e-16	1.6400e-05	1.1102e-16	1.0400e-05
5	1.1102e-16	2.5900e-05	1.1102e-16	1.1100e-05
6	1.1102e-16	3.8200e-05	1.1102e-16	1.2400e-05
7	1.1102e-16	5.5700e-05	1.1102e-16	1.3800e-05
8	1.1102e-16	7.7800e-05	1.1102e-16	1.5700e-05
9	1.1102e-16	1.0540e-04	1.1102e-16	1.7000e-05
10	1.1102e-16	1.3950e-04	1.1102e-16	1.9000e-05
11	1.1102e-16	1.8510e-04	1.1102e-16	2.0100e-05
12	2.2204e-16	2.2390e-04	2.2204e-16	2.1600e-05
13	2.2204e-16	2.7790e-04	2.2204e-16	2.4800e-05
14	2.2204e-16	3.1520e-04	2.2204e-16	2.5200e-05
15	2.2204e-16	3.9440e-04	2.2204e-16	2.6600e-05
16	2.2204e-16	4.8360e-04	2.2204e-16	3.4000e-05
17	2.2204e-16	5.4870e-04	2.2204e-16	2.9800e-05
18	2.2204e-16	7.3370e-04	2.2204e-16	3.1900e-05
19	2.2204e-16	7.6490e-04	2.2204e-16	3.2800e-05
20	2.2204e-16	9.6510e-04	2.2204e-16	3.5000e-05
30	2.2204e-16	4.5664e-03	2.2204e-16	5.0700e-05
50	2.2204e-16	1.4058e-02	2.2204e-16	7.7000e-05
100	2.2204e-16	1.1588e-01	2.2204e-16	1.4640e-04
150	2.2204e-16	3.8370e-01	2.2204e-16	2.5230e-04
200	2.2204e-16	9.0670e-01	2.2204e-16	3.4970e-04
300	2.2204e-16	2.8147e+00	2.2204e-16	5.1620e-04
400	2.2204e-16	7.1578e+00	2.2204e-16	7.0190e-04
500	2.2204e-16	1.4272e+01	2.2204e-16	8.2020e-04
600	2.2204e-16	2.4541e+01	2.2204e-16	1.0731e-03
700	2.2204e-16	3.9568e+01	2.2204e-16	1.4974e-03
900	2.2204e-16	8.3934e+01	2.2204e-16	1.7071e-03
1000	2.2204e-16	1.1609e+02	2.2204e-16	1.6861e-03

Tabela 5: Wyniki dla zadania 3

Obserwując wyniki przedstawione w tabeli 5 widać że dla obu testowanych metod wyniki błędów są identyczne. Patrząc jednak na czas działania metod, algorytm Thomsona działa zdecydowanie szybciej.



Wykres 1: Porównanie czasów wykonywania metod

Wykres 1 przedstawia porównanie czasów przetestowanych rozmiarów macierzy w skali logarytmicznej.

Wnioski

Analizując wyniki można zaobserwować że obie metody dają takie same rozwiązania. Jednak algorytm Thomasa działa zdecydowanie szybciej co doskonale obrazuje wykres 1. Problemem w metodzie Thomsona jest to że można ją stosować jedynie dla układów równań, w których wszystkie wartości poza główną przekątną oraz przekątnymi nad i pod nią są równe 0. Metoda Gaussa jest natomiast uniwersalna.

Do obliczeń został wykorzystany język programowania Python wraz z bibliotekami NumPy, math, pandas oraz matplotlib. Wszystko zostało wykonane pod system Windows 10 na procesorze i5-1135G7 2.40GHz z 16GB pamięci operacyjnej.