Układy równań liniowych – metody bezpośrednie.

Bartłomiej Kozera

1. Informacje techniczne

Obliczenia zostały wykonane na 64 bitowej wersji systemu Windows 10 Pro, z procesorem Ryzen7 3750H oraz z 16 GB pamięci RAM. Biblioteka Pandas odpowiada za wypisywanie danych na standardowe wyjście oraz zapis danych do pliku formatu .xlsx. Biblioteka numpy odpowiada za działania na macierzach, typowania zmiennych w tym języku.

2. Metoda Gaussa

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \begin{matrix} \\ \frac{a_{21}}{a_{11}} \} - \\ \end{matrix} \begin{matrix} \\ \\ \frac{a_{31}}{a_{11}} \} - \end{matrix}$$

Aby to zadziałało $a_{11} \neq 0$.
Po pierwszym etapie mamy:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix}$$

$$a_{22}^{(2)} = a_{22} - \frac{a_{21}}{a_{11}} \cdot a_{12}, a_{23}^{(2)} = \ldots,$$

$$a_{32}^{(2)} = a_{32} - \frac{a_{31}}{a_{11}} \cdot a_{12}, a_{33}^{(2)} = \ldots,$$

$$b_2^{(2)} = b_2 - \frac{a_{21}}{a_{11}} \cdot b_1, \; b_3^{(2)} = b_3 - \frac{a_{31}}{a_{11}} \cdot b_1,$$

W wyniku następnego etapu otrzymamy:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix} \quad \begin{matrix} \\ \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \end{matrix} \Big\}-$$

$$U = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix}, \; c = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix}$$

$$a_{33}^{(3)} = a_{33}^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \cdot a_{23}^{(2)}$$

$$b_3^{(3)} = b_3^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \cdot b_2^{(2)}$$

Ogólnie po k etapach otrzymujemy:
$A^{(k+1)} \cdot x = b^{(k+1)}$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot a_{kj}^{(k)}, i, j > k \quad \boxed{\text{1a}}$$

$$b_i^{(k+1)} = b_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot b_k^{(k)}, i > k \quad \boxed{\text{1b}}$$

przy założeniu, że $a_{kk}^{(k)} \neq 0$
Jeśli $a_{kk}^{(k)} = 0$ należy przestawic wiersze.

W efekcie $A^{(n)} = U$ staje się macierzą trójkątną górną (upper triangular)

Elimination

for i := 1 to n-1 do
    begin
      p :=smallest integer $in[i,\ n] : a_{pi} \neq 0$;
      if no p then no unique solution exist!; STOP;
      if $p \neq i$ then $E_p \leftrightarrow E_i$ /przestawienie/
      for j := i + 1 to n do $(E_j - \frac{a_{ji}}{a_{ii}} E_i \rightarrow E_j)$
    end
if $a_{nn} = 0$ then no unique solution exist!; STOP;

Backward substitution

$x_n = b'_n / a'_{nn}$;

for i := n − 1 downto 1 do $x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} x_j) / a_{ii}$;

Złożoność obliczeniowa tego algorytmu to O(n^3).

3. Zadanie 1

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

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

Przyjmij wektor x jako dowolną n−elementową permutację ze zbioru { 1, -1 } i oblicz wektor b. Następnie metodą eliminacji Gaussa rozwiąż układ równań liniowych Ax=b (przyjmując jako niewiadomą wektor x). Przyjmij różną precyzję dla znanych wartości macierzy A i wektora b. Sprawdź, jak błędy zaokrągleń zaburzają rozwiązanie dla różnych rozmiarów układu (porównaj – zgodnie z wybraną normą – wektory x obliczony z x zadany). Przeprowadź eksperymenty dla różnych rozmiarów układu.

Rozmiary układu które zostały przetestowane: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 30, 50 ,100. Przyjęta precyzja to float32, double (odpowiadająca float64), longdouble (odpowiadająca dloat128) Z biblioteki numpy. Błędy obliczane były jako normy Euklidesowej, która wyraża się wzorem:

$$\|A\| = \left[\sum_i abs(a_i)^2 \right]^{1/2}$$

Tabela 1. Błędy powstałe przy obliczaniu wektora X, dla różnych precyzji

n	float32	float64	longdouble
2	0,00E+00	0,00E+00	0,00E+00
3	1,12E-06	0,00E+00	0,00E+00
4	1,33E-05	3,02E-13	0,00E+00
5	6,08E-04	9,23E-12	4,07E-12
6	5,09E-02	3,64E-10	4,95E-11
7	8,57E-01	1,36E-08	2,91E-09
8	3,64E+00	1,20E-07	8,09E-08
9	2,92E+00	5,40E-07	2,93E-06
10	6,42E+00	1,66E-04	1,69E-05
11	5,75E+00	1,22E-02	4,68E-04
12	1,03E+01	1,21E+00	6,36E-02
13	1,14E+02	2,12E+01	6,92E-01
14	2,07E+00	2,11E+01	4,94E-01
15	1,63E+01	1,50E+01	1,23E+00
20	3,67E+00	8,71E+02	9,97E+01
30	2,79E+01	1,89E+02	1,08E+01
50	6,70E+04	2,46E+02	2,23E+01
100	1,82E+02	3,78E+03	7,61E+01
200	1,54E+02	3,43E+03	2,40E+02

Tabela 2. Czasy wykonania algorytmu w sekundach dla różnych dokładności.

n	czas dla float32	czas dla float64	czas dla longdouble
2	0,00	0,00	0,00
3	0,00	0,00	0,00
4	0,00	0,00	0,00
5	0,00	0,00	0,00
6	0,00	0,00	0,00
7	0,00	0,00	0,00
8	0,00	0,00	0,00
9	0,00	0,00	0,00
10	0,00	0,00	0,00
11	0,00	0,00	0,00
12	0,00	0,00	0,00
13	0,00	0,00	0,00
14	0,00	0,00	0,00
15	0,00	0,00	0,00
20	0,00	0,00	0,00
30	0,01	0,01	0,01
50	0,03	0,04	0,04
100	0,24	0,26	0,25
200	1,92	1,97	2,04

Możemy zauważyć że dla zwiększających się wymiarów macierzy, dokładność obliczeń zmniejsza się. Dokładność obliczeń zmienia się także dla różnej architektury liczby zmiennoprzecinkowej. Im większą dokładność oferuje nam dana architektura liczby, tym większą dokładność otrzymujemy, z wyjątkiem macierzy 100 x 100 oraz 200 x 200, gdzie

float32 uzyskuje lepszą dokładność niż float64. Czasy działania algorytmów praktycznie takie same dla każdej precyzji. Różnice zauważalne dopiero dla n = 50, 100, 200.

4. Zadanie 2

Powtórz eksperyment dla macierzy zadanej wzorem:

$$\begin{cases} a_{ij} = \frac{2i}{j}, j \geq i \\ a_{ij} = a_{ji}, j < i \end{cases} \qquad i, j = 1, \dots, n$$

Porównaj wyniki z tym, co otrzymano w przypadku układu z punktu 1). Spróbuj uzasadnić, skąd biorą się różnice w wynikach. Sprawdź uwarunkowanie obu układów.

Tabela 3. Błędy powstałe przy obliczaniu wektora X, dla różnych precyzji

n	float32	float64	longdouble
2	0,00E+00	0,00E+00	0,00E+00
3	1,27E-07	3,14E-16	3,14E-16
4	6,72E-08	1,11E-16	4,44E-16
5	1,61E-07	4,15E-16	3,14E-16
6	5,12E-07	9,68E-16	2,22E-16
7	1,60E-06	1,82E-15	7,11E-16
8	1,35E-06	4,68E-15	2,53E-15
9	1,77E-06	3,15E-15	1,95E-15
10	4,48E-06	3,25E-15	5,80E-15
11	4,32E-06	4,41E-15	8,43E-15
12	4,19E-06	1,99E-14	9,90E-15
13	6,56E-06	2,20E-14	9,36E-15
14	9,43E-06	2,28E-14	1,26E-14
15	1,07E-05	2,84E-14	1,45E-14
20	1,13E-05	3,79E-14	1,66E-14
30	3,01E-05	9,95E-14	6,28E-14
50	9,87E-05	3,46E-13	1,65E-13
100	6,96E-04	2,29E-12	1,56E-12
200	4,39E-03	2,56E-11	6,70E-12

Tabela 4. Czasy wykonania algorytmu w sekundach dla różnych dokładności.

n	czas dla float32	czas dla float64	czas dla longdouble
2	0,00	0,00	0,00
3	0,00	0,00	0,00
4	0,00	0,00	0,00
5	0,00	0,00	0,00
6	0,00	0,00	0,00
7	0,00	0,00	0,00
8	0,00	0,00	0,00
9	0,00	0,00	0,00
10	0,00	0,00	0,00
11	0,00	0,00	0,00
12	0,00	0,00	0,00
13	0,00	0,00	0,00
14	0,00	0,00	0,00
15	0,00	0,00	0,00
20	0,00	0,00	0,00
30	0,01	0,01	0,01
50	0,03	0,04	0,04
100	0,24	0,25	0,26
200	1,91	2,02	2,09

Tabela 5. Porównanie błędów dla obydwóch zadań dla różnych precyzji.

n	float32		float64		longdouble	
	Zadanie1	Zadanie2	Zadanie1	Zadanie2	Zadanie1	Zadanie2
2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
3	1,12E-06	1,27E-07	0,00E+00	3,14E-16	0,00E+00	3,14E-16
4	1,33E-05	6,72E-08	3,02E-13	1,11E-16	0,00E+00	4,44E-16
5	6,08E-04	1,61E-07	9,23E-12	4,15E-16	4,07E-12	3,14E-16
6	5,09E-02	5,12E-07	3,64E-10	9,68E-16	4,95E-11	2,22E-16
7	8,57E-01	1,60E-06	1,36E-08	1,82E-15	2,91E-09	7,11E-16
8	3,64E+00	1,35E-06	1,20E-07	4,68E-15	8,09E-08	2,53E-15
9	2,92E+00	1,77E-06	5,40E-07	3,15E-15	2,93E-06	1,95E-15
10	6,42E+00	4,48E-06	1,66E-04	3,25E-15	1,69E-05	5,80E-15
11	5,75E+00	4,32E-06	1,22E-02	4,41E-15	4,68E-04	8,43E-15
12	1,03E+01	4,19E-06	1,21E+00	1,99E-14	6,36E-02	9,90E-15
13	1,14E+02	6,56E-06	2,12E+01	2,20E-14	6,92E-01	9,36E-15
14	2,07E+00	9,43E-06	2,11E+01	2,28E-14	4,94E-01	1,26E-14
15	1,63E+01	1,07E-05	1,50E+01	2,84E-14	1,23E+00	1,45E-14
20	3,67E+00	1,13E-05	8,71E+02	3,79E-14	9,97E+01	1,66E-14
30	2,79E+01	3,01E-05	1,89E+02	9,95E-14	1,08E+01	6,28E-14
50	6,70E+04	9,87E-05	2,46E+02	3,46E-13	2,23E+01	1,65E-13
100	1,82E+02	6,96E-04	3,78E+03	2,29E-12	7,61E+01	1,56E-12
200	1,54E+02	4,39E-03	3,43E+03	2,56E-11	2,40E+02	6,70E-12

Dodatkowo dla porównania wyników obydwóch zadań, możemy obliczyć wskaźnik uwarunkowania zadania. Da nam to informację, jak bardzo mały błąd w macierzy b, jest w

stanie wpłynąć na rozwiązanie (macierz x). Im mniejszy jest ten błąd, tym lepiej. Obliczamy go ze wzoru:

$$k = \|A^{-1}\| * \|A\|$$

Norma dla tych macierzy liczona jest jako norma Frobeniusa przedstawiona wzorem:

$$\|A\| = \left[\sum_{i,j} abs(a_{i,j})^2\right]^{1/2}$$

Macierz odwracana jest za pomocą bibliotecznej funkcji np.linalg.inv, która rozwiązuje układ równań A*I, gdzie I jest macierzą identyczności. Układ rozwiązywany jest za pomocą metody faktoryzacji LU. Odwracanie odbyło się na precyzji float64.

Tabela 6. Wskaźniki uwarunkowania dla obydwóch zadań.

n	Wartość dla zadania 1	Wartość dla zadania 2
2	8,00E+00	1,00E+00
3	2,16E+02	1,44E+00
4	2,88E+03	1,83E+00
5	2,80E+04	2,23E+00
6	2,27E+05	2,64E+00
7	1,63E+06	3,03E+00
8	1,29E+07	3,45E+00
9	1,12E+08	3,85E+00
10	8,84E+08	4,25E+00
11	6,47E+09	4,66E+00
12	4,41E+10	5,06E+00
13	1,35E+11	5,47E+00
14	2,46E+11	5,87E+00
15	1,73E+11	6,27E+00
20	4,00E+11	8,29E+00
30	6,48E+11	1,23E+01
50	6,65E+12	2,04E+01
100	7,47E+14	4,06E+01
200	7,51E+16	8,11E+01

5. Wnioski

W zadaniu 2 już na pierwszy rzut oka możemy zauważyć, że błędy rozwiązania zachowują się tak, jak oczekiwalibyśmy od samego początku, w raz z dokładniejszą reprezentacją liczby, błąd obliczeń zmniejsza się. Nawet dla 200 wymiarów wyniki dla longdouble są bardzo dokładne. Wynika to z dużo lepszego uwarunkowania zadania, co możemy zaobserwować w tabeli 6. W zadaniu 1 już dla macierzy 3x3 uzyskujemy wyższy wskaźnik uwarunkowania, niż dla macierzy 200x200 w zadaniu 2. Czasy wykonania algorytmu dla zadania 2, niewiele gorsze niż dla zadania 1, natomiast wynikające z uwarunkowania błędy w obydwóch zadaniach są nieporównywalnie na korzyść macierzy z zadania 2.

6. Metoda Thomasa

Metoda polegająca na rozwiązaniu układu równań z macierzą trójdiagonalną. Taki układ możemy zapisać w postaci

$$a_i * x_{i-1} + b_i * x_i + c_i * x_{i+1} = d_i$$

Gdzie a_1 i c_n są zerami.

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

Złożoność obliczeniowa tego algorytmu to O(n).

Algorytm działa w następujący sposób:

$$x_n = d'_n,$$
$$x_i = d'_i - c'_i x_{i+1}, \quad i = n-1, n-2, \ldots, 1.$$

$$c'_i = \begin{cases} \dfrac{c_i}{b_i}, & i = 1, \\ \dfrac{c_i}{b_i - a_i c'_{i-1}}, & i = 2, 3, \ldots, n-1 \end{cases}$$

$$d'_i = \begin{cases} \dfrac{d_i}{b_i}, & i = 1, \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}}, & i = 2, 3, \ldots, n. \end{cases}$$

7. Zadanie 3

Powtórz eksperyment dla jednej z macierzy zadanej wzorem poniżej (macierz i parametry podane w zadaniu indywidualnym). Następnie rozwiąż układ metodą przeznaczoną do rozwiązywania układów z macierzą trójdiagonalną. Porównaj wyniki otrzymane dwoma metodami (czas, dokładność obliczeń i zajętość pamięci) dla różnych rozmiarów układu. Przy porównywaniu czasów należy pominąć czas tworzenia układu. Opisz, jak w metodzie dla układów z macierzą trójdiagonalną przechowywano i wykorzystywano macierz A.

$$\begin{cases} a_{i,i} = -m * i * k \\ a_{i,i+1} = i \\ a_{i,i-1} = \frac{m}{i},\ i > 1 \\ a_{i,j} = 0,\ j < i-1\ lub\ j > i+1 \end{cases} \qquad i,j = 1,2,\dots,n$$

Tabela 7. Wyniki dla algorytmu Gaussa i algorytmu Thomasa

n	Wynik algorytmu Gaussa	Wynik algorytmu Thomasa
5	0,00E+00	0,00E+00
10	2,48E-16	2,48E-16
15	2,48E-16	2,48E-16
20	4,84E-16	4,84E-16
50	8,16E-16	8,16E-16
100	1,21E-15	1,21E-15
200	1,83E-15	1,83E-15
300	2,18E-15	2,18E-15
500	2,83E-15	2,83E-15
700	3,38E-15	3,38E-15
1000	3,99E-15	3,99E-15

Tabela 8. Porównanie w sekundach czasów wykonania algorytmów

n	Czas dla algorytmu Gaussa[s]	Czas dla algorytmu Thomasa[s]
5	0,00	0,00
10	0,00	0,00
15	0,00	0,00
20	0,01	0,00
50	0,05	0,00
100	0,44	0,00
200	3,36	0,00
300	11,25	0,00
500	53,04	0,00
700	146,45	0,00
1000	434,80	0,01

8. Wnioski

Algorytm Thomasa działa dużo szybciej od algorytmu Gaussa, dając te same wyniki. W tabeli 8 czas dla algorytmu Thomasa jest na tyle mały, że w wynikach przyjęliśmy 0,00. Wynika to ze złożoności obliczeniowej obydwóch algorytmów. Dla algorytmu Gausa jest to O(n^3), natomiast złożoność algorytmu Thomasa jest O(n).