MOwNiT - rozwiązywanie układów równań liniowych metodami bezpośrednimi

Paweł Podedworny

05.06.2024

1 Opis ćwiczenia

Dany jest układ równań liniowych **Ax**=**b**.

1.1 Zadanie 1

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

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

Przyjęcie wektora x jako dowolną n-elementową permutację ze zbioru 1, -1 i obliczenie wektora **b**.
Następnie rozwiązanie metodą eliminacji Gaussa układu równań liniowych **Ax**=**b** (przyjmując jako niewiadomą wektor x). Przyjęcie różnej precyzji dla znanych wartości macierzy **A** i wektora **b**.
Sprawdzenie, jak błędy zaokrągleń zaburzają rozwiązanie dla różnych rozmiarów układu (porównanie zgodnie z normą maksimum wektora x obliczonego z zadanym). Przeprowadenie eksperymentów dla różnych rozmiarów układu.

1.2 Zadanie 2

Powtórzenie eksperymentu dla macierzy zadanej wzorem:

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

Porównanie wyników z tym, co otrzymano w przypadku układu z punktu 1.1. Sprawdzenie uwarunkowania obu układów.

1.3 Zadanie 3

Powtórzenie eksperymentu dla macierzy zadanej wzorem poniżej dla parametrów $m = 3$ oraz $k = 7$. Następnie rozwiązanie układu metodą przeznaczoną do rozwiązywania układów z macierzą trójdiagonalną. Porównanie wyników otrzymanych dwoma metodami (czas, dokładność obliczeń i zajętość pamięci) dla różnych rozmiarów układu.

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

2 Dane techniczne

Komputer z systemem Windows 10 x64
Procesor: AMD Ryzen 5 3600 3.60GHz
Pamięć RAM: 16GB 3200MHz
Środowisko: DataSpell 2023.3.4
Język: Python 3.11 z biblioteką numpy

3 Realizacja ćwiczenia

3.1 Zadanie 1

Do inicjalizacji zadanej macierzy posłużyłem się narzędziami z biblioteki numpy. Następnie przechodząc po wszystkich jej elementach kolejno ją uzupełniam zgodnie z podanym wzorem. Wektor x został zdefiniowany jako ciąg o naprzemiennych wartościach 1 i -1, zaczynając zawsze od 1. Następnie wymnażając macierz z wektorem uzyskałem wektor b, na bazie którego z macierzą A przeprowadzałem eliminacje Gaussa [1]. Metoda ta polega na przekształceniu macierzy do postaci trójkątnej górnej oraz modyfikacji wektora b. Początkowo eliminujemy elementy poniżej głównej przekątnej, zerując je z użyciem odpowiednich współczynników. Następnie za pomocą podstawienia wstecznego obliczamy kolejne wartości wektora x. Takie rozwiązanie ma złożoność czasową $O(n^3)$ oraz pamięciową $O(n^2)$. Do lepszego zobrazowania wyników skorzystałem z trzech dokładności float'ów z biblioteki numpy: float16, float32 oraz float64. Testy przeprowadzane był dla n z zakresu $[3, 30]$, gdzie n to liczba niewiadomych.

3.2 Zadanie 2

Macierz A oraz wektory x i b zostały wyznaczone i wyliczone w taki sam sposób jak w zadaniu 1, korzystając z kolejnego wzoru. Po wyliczeniu nowych wektorów x przystąpiłem do wyliczenia uwarunkowania macierzy A z poprzedniego i tego zadania. W związku z tym skorzystałem ze wzoru [2]:

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

gdzie norma wyliczana była w taki sposób:

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

Normę jak i uwarunkowanie liczyłem dzięki funkcji samemu zaimplementowanym. Jedynie do wyznaczenia macierzy odwrotnej skorzystałem z bibliotecznej funkcji linalg.inv. Do wyznaczenia błędów ponownie posłużyłem sie takim samym zestawem dokładności jak w zadaniu 1., jednakże do policzenia uwarunkowania skorzystałem z float64. Testy przeprowadzane były na tym samym zbiorze n co poprzednio rozszerzonym o wartość 50 oraz kolejne naturalne iloczyny liczby 100 do 1000 włącznie.

3.3 Zadanie 3

Przebieg tworzenia macierzy A oraz wektorów x oraz b przebiegały w tym wypadku tak samo jak poprzednio. Wykorzystywany w tym zadaniu algorytm Thomasa [3] jest modyfikacją standardowej eliminacji Gaussa, zoptymalizowanej pod kątem macierzy trójdiagonalnych. Po inicjalizacji macierzy trójdiagonalnej A oraz wektora b, algorytm przeprowadza eliminację współczynników pod i naddiagonalnych. Następnie stosując podstawienie wsteczne oblicza się wartości wektora rozwiązania x. Dzięki temu złożoność operacji jest $O(n)$. Na potrzeby tego sposobu tworzyłem specjalną macierz 3 x n, gdzie pierwszy wiersz obejmowały kolejne współczynniki poddiagonalne, środkowy współczynniki na przekątnej a dolny współczynniki naddiagonalne. Dzięki temu złożoność pamięciowa wyniosła $O(n)$. Jednakże do części z wykorzystaniem eliminacji Gaussa wykorzystywałem całą macierz. W tym zadaniu do ukazania różnic skorzystałem tylko z dokładności: float32 oraz float64. Testowanymi wartościami n były kolejne naturalne iloczyny liczby 50 do 1000 włącznie. Do policzenia czasu trwania skorzystałem z biblioteki time, gdzie przy badaniu nie brałem pod uwagę tworzenia macierzy.

4 Przeprowadzanie obliczeń

Do wyznaczenia błędów dla pojedynczych testów skorzystałem z normy maksumium. Wyliczonym błędem była maksymalna wartość sposród wartości bezwględnych różnic kolejnych współrzędnych wektorów:

$$\max_{i=1,\ldots,n} \{|x_i - \overline{x}_i|\}$$

gdzie x_i i $\overline{x}_i$ to elementy wektorów x i $\overline{x}$.

4.1 Zadanie 1 - wyniki

n	float16	float32	float64
3	0.00000000	0.00000209	0.00000000
4	0.00000000	0.00000012	0.00000000
5	2.45703125	0.00021541	0.00000000
6	2.85156250	0.02232707	0.00000000
7	12.4375000	0.22568977	0.00000001
8	1.97363281	6.13138866	0.00000008
9	3.89648438	13.6160764	0.00000034
10	2.67187500	6.49310207	0.00010529
11	12.0468750	8.97018051	0.00764796
12	20.2187500	13.9066667	0.71986393
13	8.32812500	76.6610107	12.5322283
14	2.16015625	3.12784433	11.9925847
15	7.09375000	7.34304523	9.06911186
16	15.9296875	9.04860497	15.6372133
17	11.8281250	13.3554182	14.4139281
18	4.98437500	50.2063369	13.5004186
19	8.63281250	27.9617900	52.1563504
20	3.91406250	9.07047939	403.455539
21	1.12890625	29.5723056	35.0990205
22	3.92187500	11.0627937	31.7170829
23	11.6796875	25.9931335	28.2739799
24	9.57031250	81.6652832	56.1457533
25	3.86718750	10.7517070	108.448057
26	2.52343750	16.8567466	44.1860174
27	63.1875000	12.9047937	49.2054693
28	20.8906250	48.2163353	622.804322
29	8.32812500	85.0923843	209.195525
30	279.000000	190.770629	73.0665260

Tabela 1: Wartości błędów w zależności od liczby niewiadomych przy różnych rodzajach dokładności dla zadania pierwszego

Jak możemy zobaczyć w tabeli 1. dla najmniejszej dokładności znaczące błędy dostajemy już dla 5 elementów w macierzy. Dla float32 akceptowalne błędy kończą się na 7 elementach, a dla float64 przy 12. Dodatkowo możemy zauważyć, że pojedyncze wyniki, które w rozwinięciu dziesiętnym mają same zera, tak naprawdę ukazują, że na dalekim miejscu po przecinku wartości różnią się od zera. Daje nam to informacje, że nie dostaliśmy idealnie równych wyników.
Otrzymane rezultaty dają nam informacje o tym, że podany wzór na macierz, jest bardzo niezoptymalizowany i operacje na nim bardzo szybko wywołują błędy.

4.2 Zadanie 2 - wyniki

n	float16	float32	float64
3	0.0009765625	0.0000001192	0.0000000000
4	0.0019531250	0.0000001192	0.0000000000
5	0.0009765625	0.0000001192	0.0000000000
6	0.0029296875	0.0000002980	0.0000000000
7	0.0063476562	0.0000010729	0.0000000000
8	0.0073242188	0.0000011921	0.0000000000
9	0.0117187500	0.0000010729	0.0000000000
10	0.0117187500	0.0000030994	0.0000000000
11	0.0107421875	0.0000027418	0.0000000000
12	0.0136718750	0.0000051260	0.0000000000
13	0.0258789062	0.0000066757	0.0000000000
14	0.0166015625	0.0000073314	0.0000000000
15	0.0190429688	0.0000074506	0.0000000000
16	0.0410156250	0.0000054836	0.0000000000
17	0.0634765625	0.0000059009	0.0000000000
18	0.0576171875	0.0000061989	0.0000000000
19	0.0766601562	0.0000056028	0.0000000000
20	0.0668945312	0.0000090599	0.0000000000
21	0.0668945312	0.0000109076	0.0000000000
22	0.0527343750	0.0000154376	0.0000000000
23	0.0644531250	0.0000179410	0.0000000000
24	0.0693359375	0.0000177026	0.0000000000
25	0.0781250000	0.0000150800	0.0000000000
26	0.0966796875	0.0000185370	0.0000000000
27	0.1054687500	0.0000160933	0.0000000000
28	0.1220703125	0.0000151396	0.0000000000
29	0.1015625000	0.0000249147	0.0000000000
30	0.1044921875	0.0000269413	0.0000000000
50	0.3632812500	0.0000689626	0.0000000000
100	1.4824218750	0.0002392530	0.0000000000
200	18.718750000	0.0013219118	0.0000000000
300	233.62500000	0.0060015917	0.0000000000
400	10.226562500	0.0100346804	0.0000000000
500	45.281250000	0.0344583988	0.0000000001
600	135.37500000	0.0480726957	0.0000000002
700	74.500000000	0.0570264459	0.0000000002
800	87.562500000	0.1133853197	0.0000000002
900	516.00000000	0.1530019045	0.0000000006
1000	142.12500000	0.2310508490	0.0000000006

Tabela 2: Wartości błędów w zależności od liczby niewiadomych przy różnych rodzajach dokładności dla zadania drugiego

Patrząc na tabelę 2., widzimy że w porównaniu do zadania pierwszego dokładność wyników znacząco wzrosła. Dla float64 uzyskaliśmy błąd rzędu 10 cyfry po przecinku dla 1000 niewiadomych. Przy float32 błąd był większy, ale dalej dla takich dużych układów nie przekraczał wartości 1. Dla float16 wyniki w stosunku do pozostałych wyszły gorsze, ale wynikały z niskiej precyzji wybranego typu. Podobnie jak w poprzednim zadaniu, taka reprezentacja zera informuje nas o nie do końca idealnym wyniku.

4.3 Uwarunkowanie macierzy

n	Zadanie 1	Zadanie 2
3	864	9
4	37920	16
5	1442000	27
6	56344680	40
7	2232478706	55
8	81552932522	72
9	2843171843290	92
10	108960945817880	115
11	3951758149661762	140
12	136453553538479040	167
13	2725392318632902656	197
14	2745139521386732544	229
15	10433483999586590720	263
16	2895360864787874304	301
17	13547271121323634688	340
18	2213939234398773182464	382
19	35706711768479363072	426
20	12204247200534620160	473
21	18981325683706511360	522
22	87949760078083948544	573
23	682892607066156728320	627
24	151520162535741562880	684
25	703598371261584752640	742
26	362112977615711371264	804
27	603916387256579260416	867
28	169886791468135743488	934
29	63136801013028020224	1002
30	26127480498444701696	1073
50	2155986395899543683072	3002
100	8674341296343485513728	12070
200	2786033414634803822592	48401
300	16930578285937360371712	108993
400	107283710368993140277248	193847
500	26044640277047910858752	302961
600	36776228838298525630464	436338
700	3838072184578605318144 0	593975
800	1350536486395107296149504	775873
900	467782270159791260172288	982033
1000	204099490468308518436864	1212454

Tabela 3: Wartości uwarunkowania macierzy z zadania 1 oraz 2 w zależności od liczby niewiadomych

Patrząc na tabelę 3. możemy zauważyć, że macierz z zadania 2 jest o wiele lepiej uwarunkowana niż ta z zadania 2. Miało to swoje przełożenie na dokładność obliczeń.

4.4 Zadanie 3 - wyniki

n	Thomas		Gauss	
	float32	float64	float32	float64
50	1.1920e-07	2.2204e-16	1.1920e-07	2.2204e-16
100	1.1920e-07	2.2204e-16	1.1920e-07	2.2204e-16
150	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
200	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
250	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
300	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
350	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
400	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
450	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
500	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
550	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
600	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
650	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
700	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
750	2.3841e-07	3.3306e-16	2.3841e-07	3.3306e-16
800	2.3841e-07	4.4408e-16	2.3841e-07	4.4408e-16
850	2.3841e-07	4.4408e-16	2.3841e-07	4.4408e-16
900	2.3841e-07	4.4408e-16	2.3841e-07	4.4408e-16
950	2.3841e-07	4.4408e-16	2.3841e-07	4.4408e-16
1000	2.3841e-07	4.4408e-16	2.3841e-07	4.4408e-16

Tabela 4: Błędy obliczeniowe dla algorytmu Thomasa i Gaussa w zależności od liczby niewiadomych

n	Thomas		Gauss	
	float32	float64	float32	float64
50	0.0	0.0	0.0039992332	0.0039992332
100	0.0	0.0	0.0150010585	0.0149972438
150	0.0	0.0	0.0335173606	0.0350234508
200	0.0	0.0	0.0605006217	0.0615589618
250	0.0009999275	0.0	0.0963447093	0.1010093688
300	0.0	0.0009973049	0.1432197093	0.1438939571
350	0.0009999275	0.0	0.1926267147	0.1965939998
400	0.0	0.0	0.2523987293	0.2594428062
450	0.0009999275	0.0009989738	0.4426851272	0.4545035362
500	0.0	0.0009996891	0.4220762252	0.4091258049
550	0.0010020732	0.0	0.6572079658	0.8132085800
600	0.0009984970	0.0009994506	0.6629014015	0.6565632820
650	0.0019989013	0.0009992122	0.7917411327	0.7846212387
700	0.0020008087	0.0019989013	0.8476758003	0.8708539009
750	0.0019991397	0.0009996891	0.9897913932	1.0107991695
800	0.0019996166	0.0010015964	1.1839642524	1.2118914127
850	0.0009994506	0.0019989013	1.3224275112	1.3327651023
900	0.0016009807	0.0009989738	1.8561489582	1.5258641242
950	0.0009992122	0.0009987354	1.5611360073	1.6626348495
1000	0.0009996891	0.0019986629	1.8379640579	1.9122438430

Tabela 5: Czasy wykonania algorytmu Thomasa i Gaussa w sekundach w zależności od liczby niewiadomych

Jak możemy zobaczyć z tabel 4 i 5 wartości błędów dla obu metod wyszły takie same. Różnic jednak w tym zadaniu powinniśmy szukać przy czasie wykonania obliczeń. I tak jak można się było spodziewać algorytm Thomasa był o wiele szybszy w stosunku do elimiacji Gaussa. Kiedy dla 1000 elementów Gauss wykonywał się prawie 2 sekundy, Thomas działał w setnych sekundy.

5 Wnioski

W pierwszych dwóch zadaniach zauważyliśmy, że dla źle uwarunkowanej macierzy obliczenia na niej szybko zaczynały wychodzić z błędami, przez co takie dane nie miały dla nas żadnego zastosowania. Kiedy jednak dobraliśmy tak macierz, aby była lepiej uwarunkowana, obliczeń bez straty dokładności mogliśmy dokonywać nawet dla tysięcy zmiennych (jeżeli dobraliśmy dobrą prezycje liczb zmiennoprzecinkowych).
W zadaniu trzecim podczas porównania algorytmu Thomasa i Gaussa łatwo mogliśmy zauważyć, że wyniki dostawaliśmy tak samo dokładne. Różnica w ich działaniu polegała na szybkości wykonywania działań. Z testów wynika, że Thomas działał kilkaset razy szybciej. Wpływ jednak na to ma fakt, że do tego algorytmu wykorzystujemy macierz trójdiagonalną a nie pełną, przez co możemy zaoszczędzić na liczbie operacji.
Po przeprowadzonych testach możemy dojść do wniosku, że dobór macierzy oraz sposobu obliczania układu ma znaczenie. Dodatkowo, mając do czynienia z liczbami o długim rozwinięciu dziesiętnym, wybór odpowiedniego typu ma również wpływ na dokładność obliczeń.

Literatura

[1] **Wikipedia**, *Gaussian elimination*, https://en.wikipedia.org/wiki/Gaussian_elimination

[2] **Wikipedia**, *Wskaźnik uwarunkowania*, https://pl.wikipedia.org/wiki/Wskaźnik_uwarunkowania

[3] **Wikipedia**, *Tridiagonal matrix algorithm*, https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm