Metoda Jacobiego rozwiązywania układów równań liniowych - Raport

Mateusz Karandys

January 7, 2024

1 Wstęp

Zajmiemy się problemem iteracyjnego wyznaczania rozwiązań układu równań $Ax = b$ dla szczególnej postaci macierzy A. Będziemy rozważać macierz postaci:

$$A = \begin{bmatrix} C & S \\ -S & C \end{bmatrix}$$

gdzie A jest wymiaru $2p \times 2p$, $C = diag(c_1, c_2, \ldots, c_p)$ przy czym $detC \neq 0$, $S = diag(s_1, s_2, \ldots, s_p)$ i $\forall i \in \{1, 2, \ldots, p\}$ $c_i^2 + s_i^2 = 1$.

2 Metoda Jacobiego

Zapiszmy macierz A w postaci sumy trzech macierzy L, D, U, gdzie L jest macierzą, której niezerowe elementy znajdują się pod główną diagonalą, D jest macierzą diagonalną, U jest macierzą o niezerowych elementach znajdujących się powyżej głównej diagonali.

$$A = L + D + U =$$
$$\begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ a_{21} & 0 & \cdots & 0 & 0 \\ a_{31} & a_{32} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Po podstawieniu tak zapisanej macierzy A do układu $Ax = b$ otrzymujemy

$$(L + D + U)x = b$$

co można zapisać jako

$$Dx + (L + U)x = b$$
$$Dx = -(L + U)x + b$$
$$x = -D^{-1}(L + U)x + D^{-1}b$$

Po dopisaniu numeru iteracji mamy

$$x^{(k+1)} = -D^{-1}(L + U)x^{(k)} + D^{-1}b, \quad k = 0, 1, \ldots$$

3 Implementacja

Implementację przygotowaliśmy w środowisku MATLAB. Zaimplementowaliśmy funkcję wyznaczającą rozwiązanie zadanego układu przy użyciu metody Jacobiego. Wybraliśmy przybliżenie początkowe $x = [0, \ldots, 0]^T$. Jako warunek stopu wybraliśmy nierówność $||x_k - x_{k-1}||_2 \leq \varepsilon$, gdzie ε jest podawany przez użytkownika. Dodatkowo narzuciliśmy ograniczenie na liczbę iteracji, aby program nie wykonywał się w nieskończoność w przypadku braku zbieżności. Ograniczeniem jest 100000. Wyniki porównywaliśmy z wbudowaną funkcją *linsolve*. Do porównania wyników przygotowaliśmy pomocniczą funkcję zwracającą wyniki w postaci dwóch tabel.

4 Obsługa Skryptu

W skrypcie *przyklady.m* znajdują się badane przypadki metody. W celu wykonania przykładu należy odkomentować linie kodu mu odpowiadające, a następnie uruchomić skrypt. Poniżej znajduje się specyfikacja zaimplementowanych funkcji.

1. *jacobi.m* - $jacobi(c, s, b, stop)$ Wyznacza rozwiązanie układu równań $Ax = b$ metodą Jacobiego. c, s - poziome wektory tej samej długości - wyznaczające macierz układu, b - kolumna wyrazów wolnych układu, *stop* - wartość toleracji błedu. Funkcja zwraca dwie wartości: x - rozwiązanie układu, *cnt* - liczbę wykonanych iteracji (max 100000)

2. *testSolve.m* - $testSolve(c, s, b)$ Wyznacza rozwiązanie układu równań $Ax = b$. Parametry c, s oraz b są takie same jak w funkcji wyżej.

3. *porownaj.m* - $porownaj(c, s, b, d)$ Zwraca dwie tabele, jedną zawierającą zestawienie wektorów c i s, drugą z zestawieniem wyników powyższych funkcji. Parametry c, s, b jak wyżej. d jest warunkiem stopu (odpowiednik *stop* w funkcji *jacobi*).

5 Przykłady

i	c	s	Jacobi	WartoscDokladna
1	0.01	0.99995	Inf	-0.85037
2	0.05125	0.99869	Inf	-0.80518
3	0.0925	0.99571	Inf	-0.32141
4	0.13375	0.99102	Inf	-0.79121
5	0.175	0.98457	Inf	-0.32313
6	0.21625	0.97634	Inf	-0.33631
7	0.2575	0.96628	Inf	-0.079279
8	0.29875	0.95433	Inf	-0.68219
9	0.34	0.94043	Inf	-0.85077
10	0.38125	0.92447	Inf	-0.25073
11	0.4225	0.90636	Inf	-0.50496
12	0.46375	0.88597	Inf	-0.64472
13	0.505	0.86312	Inf	-0.41116
14	0.54625	0.83762	-Inf	0.23923
15	0.5875	0.80922	Inf	-0.16152
16	0.62875	0.77761	Inf	-0.16227
17	0.67	0.74236	-Inf	0.31244
18	0.71125	0.70294	-0.19163	-0.19163
19	0.7525	0.65859	0.3817	0.3817
20	0.79375	0.60824	0.079421	0.079421
21	0.835	0.55025	0.315	0.315
22	0.87625	0.48186	0.45397	0.45397
23	0.9175	0.39774	0.36222	0.36222
24	0.95875	0.28425	0.43201	0.43201
25	1	0	0.56395	0.56395

Przykład 1: Zestawienie wektorów c, s oraz odpowiadających im wyników funkcji *Jacobi* oraz rozwiązania operacją wbudowaną. Ze względu na czytelność zaprezentowane zostały pierwsze połowy wyników funkcji *Jacobi* oraz dokładne wartości (wektory wynikowe są długości 2 razy większej niż długość c). Proszę uwierzyć, że wyniki drugiej połowy są analogiczne do pokazanych - dla pewnych początkowych wartości c funckja *Jacobi* zwraca $\pm\infty$, dla pozostałych dokładne wyniki.

i	c	s	Jacobi	WartoscDokladna
1	0.7	0.71414	Inf	-32.629
2	0.70045	0.7137	-Inf	13.183
3	0.70091	0.71325	-Inf	38.106
4	0.70136	0.7128	Inf	-16.099
5	0.70182	0.71236	Inf	-17.501
6	0.70227	0.71191	Inf	-48.676
7	0.70273	0.71146	-Inf	20.152
8	0.70318	0.71101	Inf	-10.99
9	0.70364	0.71056	-Inf	12.001
10	0.70409	0.71011	-Inf	20.345
11	0.70455	0.70966	-Inf	17.683
12	0.705	0.70921	4.6574×10^{259}	-18.192
13	0.70545	0.70876	7.4025×10^{203}	-14.102
14	0.70591	0.7083	2.431×10^{148}	-24.497
15	0.70636	0.70785	7.14×10^{92}	-41.289
16	0.70682	0.7074	-5.1039×10^{36}	18.384
17	0.70727	0.70694	3.9028	3.9028
18	0.70773	0.70649	11.176	11.176
19	0.70818	0.70603	26.714	26.714
20	0.70864	0.70557	-29.318	-29.318
21	0.70909	0.70512	21.065	21.065
22	0.70955	0.70466	36.876	36.876
23	0.71	0.7042	-2.334	-2.334

Przykład 2: Wartość graniczna zbieżności metody. Dla małych wartości c_i metoda nie jest zbieżna, dla dużych jest. Granicą pomiędzy małymi a dużymi wartościami c_i jest $\frac{\sqrt{2}}{2} \approx 0.707106$. Jeśli każdy element wektora c jest $\geq \frac{\sqrt{2}}{2}$, to macierz A jest diagonalnie dominująca, więc metoda jest zbieżna. Można postawić hipotezę, że jest to również warunek konieczny zbieżności metody.

i	b	Jacobi	WartoscDokladna
1	0.7066×10^6	6.4945×10^5	6.4945×10^5
2	7.0461×10^6	4.6581×10^6	4.6581×10^6
3	6.9846×10^6	5.6726×10^6	5.6726×10^6
4	6.6057×10^6	3.513×10^6	3.513×10^6
5	5.9652×10^6	4.7952×10^6	4.7952×10^6
6	6.1906×10^6	3.72×10^6	3.72×10^6
7	6.4159×10^6	2.0145×10^6	2.0145×10^6
8	1.0285×10^6	-1.559×10^6	-1.559×10^6
9	1.2370×10^6	-1.6386×10^6	-1.6386×10^6
10	9.8860×10^6	4.187×10^6	4.187×10^6
11	1.5222×10^6	6.2×10^5	6.2×10^5
12	4.8879×10^6	4.0508×10^6	4.0508×10^6
13	7.2602×10^6	4.7304×10^6	4.7304×10^6
14	8.7414×10^6	7.6701×10^6	7.6701×10^6
15	1.7486×10^6	1.238×10^6	1.238×10^6
16	2.7612×10^6	3.38×10^5	3.38×10^5
17	2.4845×10^6	1.1169×10^6	1.1169×10^6
18	1.9555×10^6	-1.1115×10^6	-1.1115×10^6
19	9.6078×10^6	8.3133×10^6	8.3133×10^6
20	1.0495×10^6	-2.5291×10^6	-2.5291×10^6

Przykład 3: Zestawienie dla dużych wartości wekotra b dla metody zbieżnej ($0.8 \leq c_i \leq 1$ dla każdego i). Wartości wektora b nie psują zbieżności.

Rozmiar c	Rozmiar A	Zbieżność
100	200×200	TAK
500	1000×1000	TAK
1000	2000×2000	TAK

Przykład 4: Zbadaliśmy również wpływ rozmiaru danych na zbieżność metody. Dla wektora c długości 100, 500, 1000 metoda była zbieżna z ograniczeniem z przykładu 2. Wniosek: Zbieżność metody nie zależy od rozmiaru danych wejściowych. Ze względu na czytelność nie umieszczamy tu tabeli potwierdzających te obserwacje.

Próba Rozmiar A	1	2	3	4	5	6	7	8	9	10	Średnia
10×10	67	59	51	74	54	68	59	57	66	73	62.8
20×20	80	79	75	80	78	75	77	67	80	81	77.2
40×40	78	73	82	80	78	77	69	74	80	83	77.4
100×100	77	83	81	82	80	80	79	80	81	78	80.1
200×200	79	82	82	84	81	84	81	81	82	82	81.8
1000×1000	85	86	85	85	86	85	85	85	85	86	85.3

Przykład 5: Liczba iteracji w zależności od rozmiaru macierzy A. Analizę przeprowadzono dla wartości $c_i \geq 0.8$. Wraz ze zwiększaniem rozmiaru danych liczba iteracji potrzebna do uzyskania wyniku nieznacznie rośnie.

Próba Zakres c_i	1	2	3	4	5	6	7	8	9	10	Średnia
$[0.95, 1]$	22	22	22	22	22	22	22	22	22	22	22
$[0.90, 1]$	33	33	33	34	33	32	33	33	34	33	33.1
$[0.85, 1]$	49	51	49	49	51	49	46	50	48	43	48.5
$[0.80, 1]$	81	78	78	77	82	79	81	79	80	80	79.5

Przykład 6: Liczba iteracji potrzebnych do otrzymania wyniku w zależności od zakresu wartości elementów wektora c. Zestawienie przygotowano dla macierzy A rozmiaru 100×100 (c ma tu długość 50). Im wartości c_i są bliższe 1 tym szybsza jest metoda.

6 Zastosowania

1. Iteracyjny algorytm znajdowania wartości własnych macierzy (Eigenvalue Problem)
2. Jądrowe maszyny wektorów nośnych (Kernelized Support Vector Machines)

Źródła

1. Kicaid A. "Analiza numeryczna"
2. Notatki do wykładu Wróbel I. "Metody numeryczne", MiNI 2023, semestr zimowy
3. https://wazniak.mimuw.edu.pl/index.php?title=MN08