28 października 2019

Paweł Golik
298868
G1

Metoda Jacobiego dla układu Ax = b z macierzą rozrzedzoną.

Projekt nr 36

1 Wstęp

Celem projektu jest zgłębienie problemu rozwiązywania dużych równań liniowych $Ax = b^*$, w większości posiadających zerowe współczynniki liczbowe (macierz rozrzedzona), przy wykorzystaniu metody iteracyjnej Jacobiego z zastosowaniem modyfikacji algorytmu oraz alternatywnych sposobów reprezentowania danych, pozwalających na obsługę macierzy rozrzedzonych o dużych rozmiarach (np. $10^6 \times 10^6$) z optymalną złożonością czasową oraz pamięciową.

W projekcie uwzględniono zagadnienie warunku zbieżności metody oraz problematykę generowania odpowiednich macierzy spełniających wymagania zadania, a także rezultat programu dla niewłaściwych danych.

Algorytm przetestowany został na kilu przypadkach macierzy rozrzedzonej, przekątniowo dominującej, o rożnym rozmiarze oraz współczynniku wypełnienia (liczbie niezerowych elementów macierzy w porównaniu do jej rozmiarów), badając czas działania programu oraz liczbę iteracji metody.

Program zabezpieczony został także przed podaniem niewłaściwych danych – wprowadzenia macierzy osobliwej lub niespełniającej warunków zbieżności metody Jacobiego. Każdy z przypadków został porównany z rezultatem funkcji wbudowanej programu MATLAB.

* A – macierz rozrzedzona o wymiarach $n \times n$; x – wektor pionowy rozwiązań o rozmiarze n; b – wektor pionowy współczynników wolnych o rozmiarze n

2 Opis użytych metod

Algorytm iteracyjny - Metoda Jacobiego:

Biorąc $M = diag(A)$, gdzie $diag(A)$ jest macierzą diagonalną składającą się z wyrazów stojących na głównej przekątnej macierzy A oraz $Z = A - M$, układ $Ax = b$ jest równoważny układowi: $Mx = Zx + b$, a stąd (o ile na przekątnej macierzy nie mamy zera) otrzymujemy metodę iteracyjną: $x_k = Bx_{k-1} + c$, gdzie $B = M^{-1}Z$ i $c = M^{-1}b$, zwaną metodą Jacobiego.

Użyty algorytm zwraca poprawne wyniki jedynie dla macierzy, spełniających kryterium zbieżności tej metody.

Twierdzenie o zbieżności algorytmu Jacobiego:
W metodzie Jacobiego warunek dostateczny zbieżności

$$\| B \| < 1$$

jest spełniony np. wtedy, gdy macierz ma dominującą przekątną, tzn. gdy

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}| \qquad \forall i = 1, ..., N$$

Metoda reprezentacji macierzy rozrzedzonych:

W celu optymalnego działania programu i małej złożoności pamięciowej w projekcie skorzystano z wbudowanej funkcji programu MATLAB – *sparse* (oraz jej modyfikacji) w celu utworzenia macierzy rozrzedzonej o rozmiarze $n \times n$, zapisanej jako trzy wektory o długości n, przedstawiające: „i-te" współczynniki macierzy, „j-te" współczynniki macierzy oraz wartość współczynnika $a_{i,j}$. Funkcja umożliwia działania na macierzach typu *sparse* zgodnie ze składnią MATLAB dla zwykłych macierzy. Złożoność pamięciowa algorytmu jest rzędu $O(n)$.

Generowanie przykładów:

Dla pewności wykonalności programu wygenerujemy macierze spełniające twierdzenie o zbieżności metody Jacobiego.

Utworzenie macierzy dominujących diagonalnie zostało zrealizowane przez wygenerowanie macierzy jednostkowej w formacie *sparse* za pomocą funkcji *speye* oraz przemnożenie jej przez dużą liczbę, a następnie dodanie do niej macierzy rozrzedzonej o małych współczynnikach przy wykorzystaniu funkcji *sprandn* – przyjmującą wielkość żądanej macierzy oraz wsp. wypełnienia.

Program zabezpieczono przed wykonaniem dla niewłaściwych danych, czyli macierzy osobliwych oraz niespełniających twierdzenia o zbieżności.

Utworzenie macierzy niedominujących odbyło się analogicznie, z zamianą rzędu wielkości współczynników na diagonali macierzy i tych poza nią.

Utworzenie macierzy osobliwych to utworzenie macierzy rozproszonej, w której ostatni wiersz jest sumą wcześniejszych.

Wektor wyrazów wolnych został wygenerowany dla odpowiedniego rozmiaru n.

Sprawdzenie poprawności:

Poprawność działania algorytmu zrealizowano przez wbudowaną funkcję: $x = A/b$.

Inne:

Stałe wykorzystane w zadaniu (*toll* – dokładność, max_{iter} – maksym. liczba iteracji, *seed* – wsp. wypełnienia macierzy) zostały wyznaczone empirycznie.

Czas wykonania poszczególnych części kodu zmierzono przy użyciu funkcji *tic* and *toc*.

Odwracanie macierzy diagonalnej odbyło się poprzez odwrócenie wszystkich jej współczynników zgodnie jej własnością

$$diag(d_1, d_2, ..., d_n)^{-1} = diag(d_1^{-1}, d_2^{-1}, ..., d_n^{-1})$$

za pomocą funkcji *inv*.

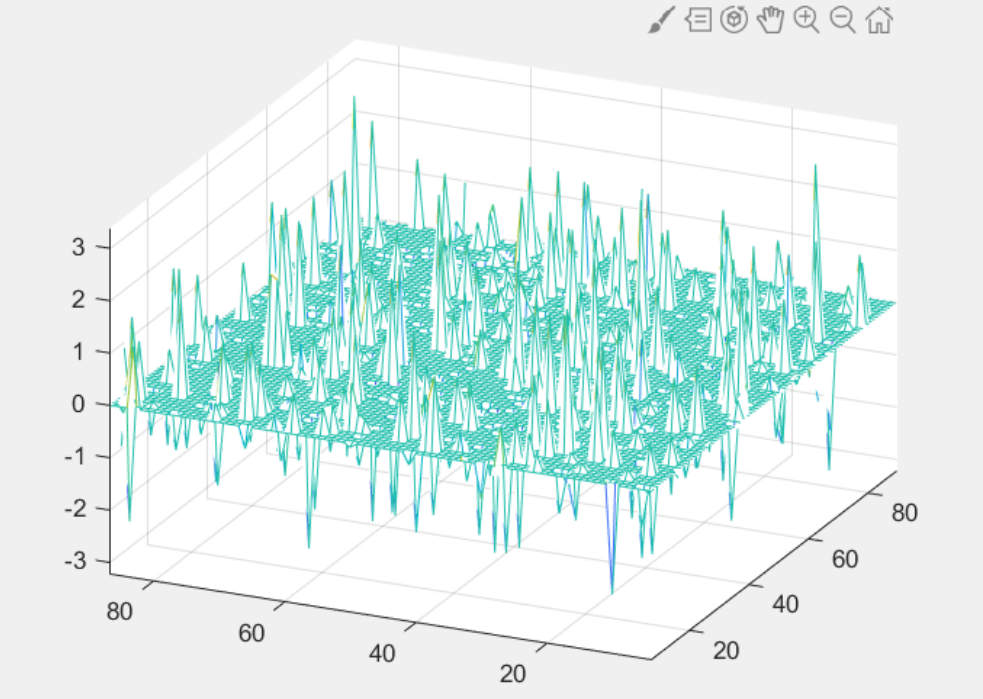
3 Prezentacja przykładów oraz wyników

Działanie algorytmu dla poprawnych macierzy rozrzedzonych o wielkości 100×100****:**

Początkowo program generuje macierz w sposób opisany powyżej dla rozmiaru $n = 100$ dla trzech różnych wartości współczynników wypełnienia macierzy***. Pozwala to na zmierzenie wpływu liczby niezerowych elementów na działanie metody.

Przykładowe dane wejściowe:

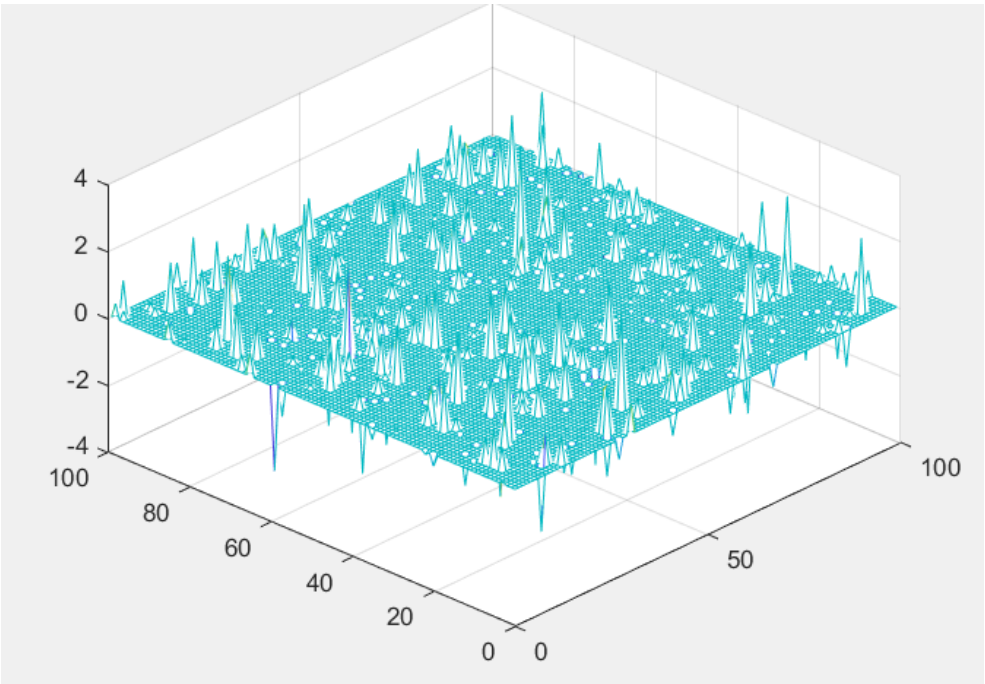
Macierz A1: 100×100 ze wsp. wypełnienia: 0.1
Macierz A2: 100×100 ze wsp. wypełenienia: 0.05
Macierz A3: 100×100 ze wsp. wypełnienia: 0.01



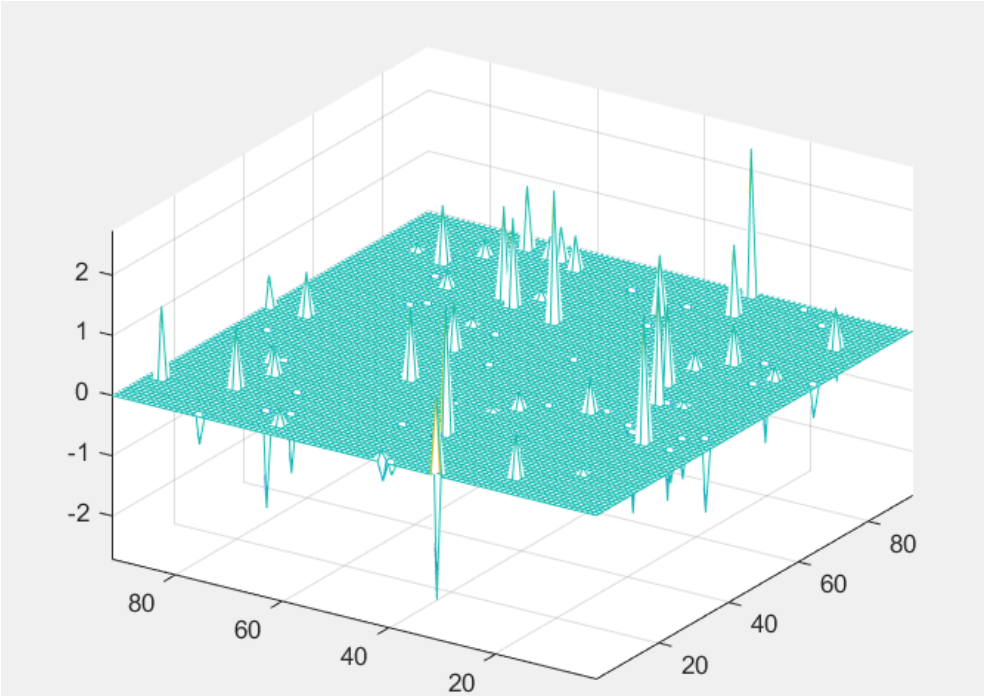
Rysunek 1: Macierz A1

** rozmiar 100×100 wybrano ze względu na nieczytelność grafik prezentujących macierze dla większego n, dla $n = 10^4, 10^5, 10^6$ zaprezentowano czas trwania algorytmu i liczbę iteracji w tabelce.

*** funkcja *sprandn* generująca macierz rozrzedzoną jako wsp. wypełnienia przyjmuje argument $size \times size \times density$, zatem dla macierzy o różnych wymiarach dany współczynnik nie gwarantuje podobnej gęstości macierzy.



Rysunek 2: Macierz A2



Rysunek 3: Macierz A3

Wykonanie programu:

Po dodaniu powyższych macierzy do macierzy diagonalnych o dużych współczynnikach oraz połączeniu z wektorem wyrazów wolnych, algorytm rozpoczął iteracje w celu uzyskania rozwiązań układu.

Warunki STOPu:

$$|x_{k+1} - x_k| < toll$$

oraz liczba iteracji nie przekracza max_{iter}.

Funkcja tic i toc mierzyła czas wykonania pętli oraz czas sprawdzenia poprawności przez funkcję $x = A/b$.

Wynik algorytmu oraz sprawdzenie:

Tabela 1: Wyniki algorytmu dla macierzy 100×100

Macierz	A1	A2	A3
Wsp. wypełnienia	0.1	0.05	0.01
Czas trwania algorytmu	0.0026	0.0028	0.0024
Czas sprawdzania	0.0122	0.0007301	0.0001969
Przybliżenie ostatniej niewiadomej (wg. alg)	0.094132541659185	0.034301877470114	0.087909295989998
Przybliżenie ostatniej niewiadomej (wg. spr)	0.094132541738859	0.034301877529077	0.087909296011243
Liczba iteracji	6	6	5

Algorytm powtórzono dla innych rozmiarów macierzy ($10^4 \times 10^4$, $10^5 \times 10^5$ *oraz* $10^6 \times 10^6$) przy zachowaniu analogii do powyższego przykładu.

Tabela 2: Wyniki algorytmu dla macierzy $10^4 \times 10^4$

Macierz	B1	B2	B3
Wsp. wypełnienia	0.001	0.0005	0.0001
Czas trwania algorytmu	0.0198	0.0834	0.00345
Czas sprawdzania	0.0134	0.0009872	0.0002498
Przybliżenie ostatniej niewiadomej (wg. alg)	0.042837527839237	0.083905325790356	0.024876543258384
Przybliżenie ostatniej niewiadomej (wg. spr)	0.042837527839023	0.083905325790278	0.024876543258984
Liczba iteracji	6	6	5

Tabela 3: Wyniki algorytmu dla macierzy $10^5 \times 10^5$

Macierz	C1	C2	C3
Wsp. wypełnienia	0.00001	0.000005	0.000001
Czas trwania algorytmu	0.6234566	0.034987	0.0043235
Czas sprawdzania	0.094347	0.00618	0.000543
Przybliżenie ostatniej niewiadomej (wg. alg)	0.034347896543547	0.065839575734923	0.068542459273822
Przybliżenie ostatniej niewiadomej (wg. spr)	0.034347896543543	0.065839575734922	0.068542459273829
Liczba iteracji	7	6	5

Tabela 4: Wyniki algorytmu dla macierzy $10^6 \times 10^6$

Macierz	D1	D2	D3
Wsp. wypełnienia	0.000001	0.0000005	0.0000001
Czas trwania algorytmu	1.6277938	1.0143663	0.3574204
Czas sprawdzania	1.2427691	0.4075918	0.2650865
Przybliżenie ostatniej niewiadomej (wg. alg)	0.093092262067797	0.080139632197360	0.068439609854172
Przybliżenie ostatniej niewiadomej (wg. spr)	0.093092262067770	0.080139632197360	0.068439609854172
Liczba iteracji	9	7	6

Interpretacja

Metoda Jacobiego dobrze spisuje się w rozwiązywaniu układów równań z macierzą rozrzedzoną, uzyskanie dobrej dokładności nie kosztuje nas zbyt wielu iteracji a czas wykonania iteracji jest zbliżony do czasu trwania funkcji wbudowanej MATLAB dla małych n.

Dla dużych n metoda Jacobiego odznacza się gorszym wynikiem czasowym, jednak nie jest to wynik znacząco gorszy od czasu dla funkcji wbudowanej. Im mniejszy wsp. wypełnienia, czyli mniej niezerowych elementów macierzy, tym szybszy jest czas wykonania i potrzeba mniej iteracji.

Podanie macierzy niespełniającej warunku zbieżności:

Jeżeli podana macierz nie będzie spełaniała warunku zbieżności to maksymalna liczba iteracji zostanie przekroczona, a wyswietlony zostanie odpowiedni komunikat.

Podanie macierzy osobliwej:

Macierz o wyznaczniku równym zero jest generowana w sposób opisany powyżej, następnie program przed przystąpieniem do wykonywania algorytmu iteracyjnego sprawdza wyznacznik macierzy A i informuje o jej osobliwości. Program zostaje przerwany.

4 Wnioski

Powyżej opisany sposób przechowywania macierzy pozwala na szybkie rozwiązywanie układów równań liniowych przy zachowaniu intuicyjnej składni MATLAB dla operacji na macierzy. Kod działający dla macierzy typu *sparse* nadaje się do rozwiązania układu równań przy wykorzystaniu zwykłych macierzy, z zastrzeżeniem ich mniejszego rozmiaru (inaczej nie dałoby się ich przechowywać).

Uniwersalność kodu to istotna zaleta tego rozwiązania przy jednoczesnej optymalizacji pamięciowej.

Zastosowanie metody Jacobiego w formie wzoru bazującego na trzech macierzach (diagonalnej, dolnej i górnej trójkątnej) to ułatwienie w czytelności programu, z uwagi na kompatybilność macierzy *sparse* z działaniem funkcji macierzowych.

Nie ma konieczności „ręcznego" działania na poszczególnych indeksach wartości niezerowych, co ułatwia zrozumienie algorytmu oraz jego poprawki. Metoda Jacobiego dla dobrze dobranych danych prezentuje dużą efektywność rozwiązywania układów. Pamiętać należy jednak o jej ograniczeniach.

Literatura

[1] https://www.mathworks.com/help/matlab/ref/sparse.html?s_tid=srchtitle

[2] http://www.mini.pw.edu.pl/ iwrobel/MN1_informatyka_lato_2018-19/Materialy

[3] http://wazniak.mimuw.edu.pl/index.php?title=MN08