Sprawozdanie z Zadania Numerycznego 6

Paweł Nykiel

Grudnień 2024

1 Polecenie

Zadana jest macierz:

$$M = \begin{pmatrix} 9 & 2 & 0 & 0 \\ 2 & 4 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}.$$

(a) Stosując metodę potęgową znajdź największą co do modułu wartość własną macierzy M oraz odpowiadający jej wektor własny. Na wykresie w skali logarytmicznej zilustruj zbieżność metody w funkcji ilości wykonanych iteracji.

(b) Stosując algorytm QR bez przesunięć, opisany w zadaniu nr 6, znajdź wszystkie wartości własne macierzy M. Sprawdź, czy macierze A_i upodabniają się do macierzy trójkątnej górnej w kolejnych iteracjach. Przeanalizuj i przedstaw na odpowiednim wykresie, jak elementy diagonalne macierzy A_i ewoluują w funkcji indeksu i.

(c) Zastanów się, czy zbieżność algorytmu z pkt. (a) i (b) jest zadowalająca. Jak można usprawnić te algorytmy?

Wyniki sprawdź używając wybranego pakietu algebry komputerowej lub biblioteki numerycznej.

2 Wprowadzenie

W niniejszym zadaniu analizujemy macierz M przy wykorzystaniu dwóch metod numerycznych: potęgowej oraz QR. Pierwsza metoda służy do wyznaczenia największej wartości własnej macierzy i odpowiadającego jej wektora własnego. Druga umożliwia określenie wszystkich wartości własnych, a także obserwacje ich ewolucji podczas kolejnych iteracji. Celem jest porównanie skuteczności oraz efektywności obu podejść.

3 Część teoretyczna

3.1 Zadana macierz

Rozważana macierz ma następującą postać:

$$M = \begin{pmatrix} 9 & 2 & 0 & 0 \\ 2 & 4 & 1 & 0 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}.$$

Jest to macierz symetryczna, a jej struktura pozwalala na efektywne zastosowanie metod iteracyjnych i numerycznych. Obliczenia przeprowadzono w oparciu o program napisany w Pythonie.

3.2 Metoda potęgowa

Metoda ta pozwala na znalezienie największej co do modułu wartości własnej poprzez iteracyjne mnożenie macierzy przez wektor. Po każdej iteracji wynik normalizujemy, a przybliżenia wartości własnej wyznaczamy na podstawie skalarnych iloczynów. Proces kontynuujemy do momentu, gdy różnica pomiędzy kolejnymi wynikami spełniała założenie tolerancji (zbieżności).

Liczenie zaczynamy od wyboru początkowego punktu x. Później wkraczamy w iteracyjną pętle i liczymy tymczasowy wektor:

$$\boldsymbol{y = Mx_i}$$

Teraz przybliżamy największą wartość własną macierzy M:

$$\boldsymbol{\lambda_i = \frac{y^p x_i}{x_i^p x_i}}$$

Ostatecznie normalizujemy wektor y otrzymując kolejny wektor:

$$\boldsymbol{x_{i+1} = \frac{y}{\|y\|}}$$

Iterujemy w ten sposób dopóki nie osiągniemy ustanowionej wcześniej granicy tolerancji. W naszym przypadku wynosi ona $\boldsymbol{e^{-10}}$

3.3 Algorytm QR

Ta metoda opiera się na rozkładzie QR macierzy M, co pozwala na iteracyjne przybliżanie wszystkich jej wartości własnych. Po każdym kroku obliczeniowym powstaje nowa macierz, która stopniowo przybiera postać macierzy trójkątnie górnej. Kryterium zakończenia iteracji jest zbieżność macierzy lub osiągnięcie ustalonej maksymalnej liczby kroków.

W tym algorytmie nie wybieramy żadnego punktu początkowego, jednak tworzymy nową macierz:

$$\boldsymbol{M_1 = M}$$

Oraz stosujemy na niej dekompozycję QR:

$$\boldsymbol{M_1 = Q_1 R_1}$$

Iteracyjnie powtarzamy schemat:

$$\boldsymbol{M_{i+1} = Q_{i+1} R_{i+1} = Q_i R_i}$$

Używając tego algorymu kolejne macierze M nie zmieniają swoich wartości własnych, a metoda działa szybciej z każdą iteracją. Jest to skutek tego, że macierz coraz bardziej przypomina macierz trójkątną górną. Program kończy liczenie, gdy osiągniemy granicę tolerancji $\boldsymbol{e^{-10}}$ lub po 1234 iteracjach.

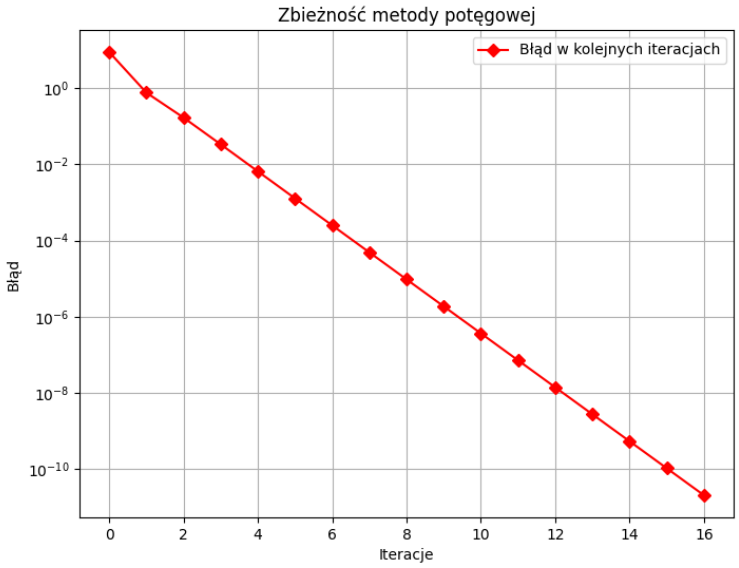
4 Wyniki

4.1 Metoda potęgowa

Wyznaczona wartość własna wynosi $\lambda \approx 9.71854825$.
Towarzyszący jej wektor własny:

$$\vec{\boldsymbol{v}} = \begin{bmatrix} \mathbf{0.93984702} \\ \mathbf{0.33766423} \\ \mathbf{0.05124805} \\ \mathbf{0.00664006} \end{bmatrix}$$

jest zgodny z wynikami uzyskanymi przy użyciu biblioteki `numpy`. Poniższy wykres ilustruje szybkie tempo zbieżności.



Rysunek 1: Wartość błędu dla kolejnych iteracji

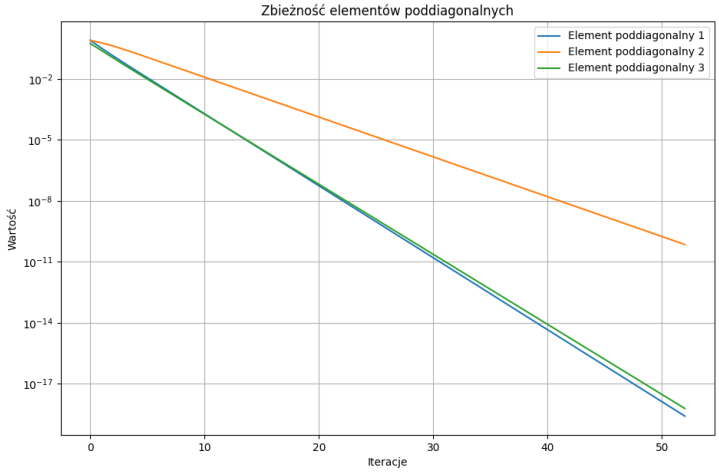
4.2 Algorytm QR

Wartości własne macierzy, uzyskane tą metodą, wynoszą odpowiednio:

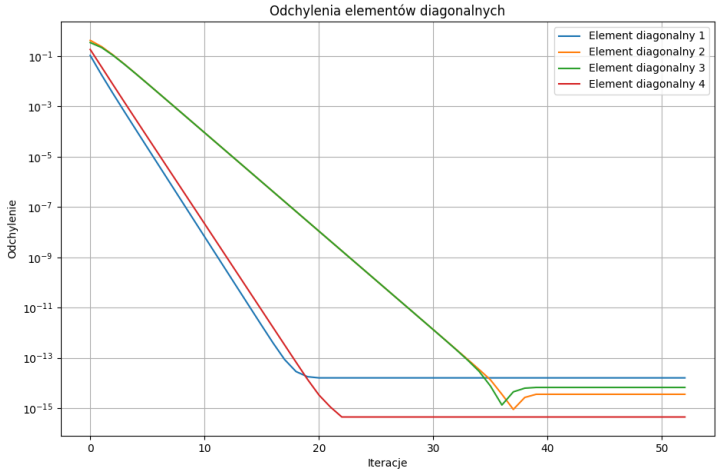
- 9.718548254119618
- 4.301704905012378
- 2.740194113151937
- 1.2395527277160612

Co jest zgodne z wynikami uzyskanymi przy użyciu biblioteki `numpy`:

- 9.71854825
- 4.30170491
- 2.74019411
- 1.23955273



Rysunek 2: Wartość elementu pod diagonalą w kolejnych iteracjach



Rysunek 3: Różnica pomiędzy dokładną wartością, a wartością własną

5 Wnioski

5.1 Metoda potęgowa

Z analizy wyników wynika, że metoda potęgowa wykazuje dobrą zgodność z wynikami uzyskanymi za pomocą biblioteki `numpy`, różniąc się jedynie w dalszych miejscach po przecinku. Na podstawie wykresu można zauważyć, że każda kolejna iteracja przybliża wynik do wartości oczekiwanej, co potwierdza zbieżność tej metody.

5.2 Algorytm QR

Uzyskane wartości własne przy zastosowaniu algorytmu QR są w pełni zgodne z wynikami obliczeń numerycznych z wykorzystaniem biblioteki numpy. W procesie iteracyjnym

zauważono, że macierz stopniowo nabiera struktury macierzy trójkątnej górnej, co można zaobserwować na drugim wykresie. Elementy poniżej diagonali maleją do zera, co potwierdza poprawność algorytmu.

6 Potencjalne ulepszenia

6.1 Metoda potęgowa

Metodę potęgową można usprawnić, wprowadzając przesunięcia. Taka modyfikacja polega na dodaniu lub odjęciu określonej wartości od elementów macierzy, co przyspiesza zbieżność do dominującej wartości własnej. Trzeba jednak zachować ostrożność, ponieważ nieodpowiednie dobranie przesunięcia może wydłużyć proces iteracji.

6.2 Algorytm QR

W przypadku algorytmu QR, zastosowanie przesunięć również może poprawić efektywność i skrócić czas obliczeń. Jeśli macierz początkowa nie jest w postaci Hessenberga, warto najpierw dokonać odpowiedniej transformacji. Macierz w postaci Hessenberga, która posiada zera poniżej pierwszej podprzekątnej, znacząco ułatwia dalsze obliczenia. W tym zadaniu taka transformacja nie była konieczna, co uprościło implementację.