Metody numeryczne - N8

DAMIAN PORADYŁO

N8 *Zadanie numeryczne*

Znaleźć wartości własne macierzy z dokładnością 10^{-8}

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & -1 \end{pmatrix}, \tag{8}$$

korzystając z metody potęgowej, Rayleigha i metody iteracyjnej QR:

$$\begin{aligned} B^{(0)} &= A, \\ Q^{(n)}R^{(n)} &= B^{(n)}, \\ B^{(n+1)} &:= R^{(n)}Q^{(n)}. \end{aligned} \tag{9}$$

1 OPIS METODY

Uzyskane wyniki:

Metoda:	Wartości własne:
QR	**8.54851285, -4.57408722, 0.02557437**
Potęgowa	**8.548512847334699, -4.57408722900602, 0.02557437263427499**
Rayleigh'a	**8.548512853222787, -4.574087225857106, 0.025574372634318356**

1.1 METODA QR

Najbardziej przyjemna w implementacji metoda. Wykonujemy rozkład QR dopóki nie uda nam się uzyskać żądanej precyzji.

W tym rozkładzie $A = QR$, gdzie Q jest macierza ortogonalna ($QTQ = 1$), a R jest macierzą trójkątna górna (czyli taka gdzie wszystkie elementy poniżej diagonali są równe zero).

Nasze wartości własne, zawarte są na diagonali macierzy wynikowej.

1.2 METODA POTĘGOWA

W każdej z naszych iteracji wykonujemy poniższą operacje:

$$b_{i+1} = \frac{Ab_i}{\|Ab_i\|}$$

tzn., musimy pomnożyć wektor przez macierz a następnie to normalizujemy i dzielimy obie wartości. Dzięki temu, otrzymamy wektor własny który jest sprzężony z naszą największą wartością własną.

Na początku musimy jednak sami wybrać dowolny wektor startowy.

Generalnie, metoda potęgowa daje nam największą wartość własną danej macierzy. Aby uzyskać wszystkie wartości własne, należy po każdej iteracji odpowiednio modyfikować macierz bazową.

Na początku szukamy odpowiednio wektora własnego q_1 oraz wartości własnej λ_1. Następnie właśnie modyfikujemy naszą macierz tj, $A_2 = A - \lambda_1 q_1 q_1^T$. Powtarzamy nasz algorytm metody potęgowej tym razem na macierzy A_2 aby znaleźć λ_2 oraz q_2. Robimy to analogicznie dla każdej kolejnej wartości własnej. Jest to tzw. *deflacja Hotellinga.*

1.3 METODA RAYLEIGHA

Metoda ta, można powiedzieć, że jest swego rodzaju zmodyfikowana odwrotną metodą potęgową.

Najpierw musimy policzyć początkowe przybliżenie wartości własnej (dla początkowego wektora b):
$\lambda_{new} = b^T A b$

Kolejnym krokiem jest modyfikacja początkowego wektora b: $b_{i+1} = \frac{(A-\lambda_i \mathbb{I})^{-1} b_i}{\|(A-\lambda_i \mathbb{I})^{-1} b_i\|}$

Następnie obliczamy kolejne przybliżenia wartości własnej w postaci: $\lambda_{new} = b_i^T A b_i$

Obliczenia wykonujemy do momentu uzyskania dokładności rzędu 10^{-8}

Dzięki dobraniu kolejno wektorów początkowych b :

1. $b_1 = [1, 0, 0]$
2. $b_2 = [0, 1, 0]$
3. $b_3 = [0, 0, 1]$

Jesteśmy w stanie obliczyć wszystkie wartości własne.

Generalnie, główną wadą tej metody jest to, że podczas każdej iteracji musimy rozwiązywać układ równań z inna macierzą.

2 KOD PROGRAMU

2.1 METODA QR

```
import numpy as np

e = 1e-8

def qr_method(A):
    A_copy = A
    while True:
        x = A_copy.item((0, 0))
        Q, R = np.linalg.qr(A_copy)
        A_copy = R * Q
        if abs(x - A_copy.item(0, 0)) < e:
            break
    return A_copy.diagonal()


A = np.matrix([[1.0, 2.0, 3.0], [2.0, 4.0, 5.0], [3.0, 5.0, -1.0]])
print(qr_method(A))
```

2.2 METODA POTĘGOWA

```
import numpy as np

e = 1e-8

def power_method(A):
    eigenvalues = []
    A_copy = np.array(A)

    for _ in range(3):
        v = np.array([1, 1, 1])
        Av = A_copy.dot(v)
        ev = v.dot(Av)
        while True:
            Av = A_copy.dot(v)
            v_new = Av / np.linalg.norm(Av)
            Av = A_copy.dot(v_new)
            ev_new = v_new.dot(Av)
            if np.abs(ev - ev_new) < e:
                break
            v = v_new
            ev = ev_new
        largest_eigenvalue = ev
        eigenvalues.append(largest_eigenvalue)
        eigenvector = np.array([v_new])
        A_copy -= largest_eigenvalue*(eigenvector * eigenvector.T)

    return eigenvalues


A = np.array([[1.0, 2.0, 3.0], [2.0, 4.0, 5.0], [3.0, 5.0, -1.0]])
print(power_method(A))
```

2.3 METODA RAYLEIGHA

```
import numpy as np

e = 1e-8


def rayleigh(A):
    vectors_b = [np.array([1.0, 0.0, 0.0]),
                 np.array([0.0, 1.0, 0.0]),
                 np.array([0.0, 0.0, 1.0])]

    eigenvalues = []
    for b in vectors_b:
        l_new = np.dot(b.T, np.dot(A, b))
        while True:
            l_old = l_new
            w = np.linalg.solve(A - l_old * np.eye(3), b)
            b = w / np.linalg.norm(w)
            l_new = np.dot(b.T, np.dot(A, b))
            err = np.linalg.norm(w - l_new * b) / np.linalg.norm(w)
            if abs(1.0 - err) < e:
                break
        eigenvalues.append(l_new)
    return eigenvalues


A = np.array([[1.0, 2.0, 3.0], [2.0, 4.0, 5.0], [3.0, 5.0, -1.0]])
print(rayleigh(A))
```

3 ZAWARTOŚĆ

W skład zestawu wchodzą:

- Plik programów:
    - qr_method.py
    - power_method.py
    - raylegih_method.py
- Opracowanie