

# Metody Numeryczne – Zadanie III

Mateusz Kamiński

Listopad 2025

## 1 Wstęp

Celem zadania jest wyznaczenie dwóch największych wartości własnych zadanej macierzy  $\mathbf{A} \in \mathbb{R}^{6 \times 6}$  oraz odpowiadających im unormowanych wektorów własnych przy użyciu metody potęgowej.

W pierwszej części wykorzystano klasyczną metodę potęgową do aproksymacji największej wartości własnej  $\lambda_1$  i wektora własnego  $\mathbf{e}_1$ . Następnie zastosowano zmodyfikowaną iterację z ortogonalizacją względem  $\mathbf{e}_1$ , w celu wyznaczenia drugiej wartości własnej  $\lambda_2$  oraz wektora własnego  $\mathbf{e}_2$ .

## 2 Macierz zadania

Rozważana w zadaniu macierz  $A$  ma postać:

$$A = \begin{pmatrix} \frac{19}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & -\frac{17}{12} \\ \frac{13}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & -\frac{11}{12} & \frac{13}{12} \\ \frac{5}{6} & \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{13}{12} & -\frac{11}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{13}{12} \\ -\frac{17}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{19}{12} \end{pmatrix}$$

Macierz jest rzeczywista i symetryczna ( $A^\top = A$ ). Z teorii wynika, że wszystkie wartości własne są rzeczywiste, a istnieje baza ortonormalnych wektorów własnych  $\mathbf{e}_1, \dots, \mathbf{e}_6$  spełniających:

$$\mathbf{A}\mathbf{e}_i = \lambda_i \mathbf{e}_i, \quad \mathbf{e}_i^\top \mathbf{e}_j = \delta_{ij}.$$

Szukamy więc dwa największe wektory własne, przy okazji dowiadując się o dwóch największych (co do modułu) wartościach własnych.

## 3 Metoda potęgowa

Niech  $\lambda_1$  będzie największą wartością własną macierzy  $\mathbf{A}$ , a  $\mathbf{e}_1$  – odpowiadającym jej unormowanym wektorem własnym. Startujemy z losowo wygenerowanego **wektora**  $\mathbf{y}_1$  o normie 1:

$$\|\mathbf{y}_1\| = 1.$$

Iteracja ma postać:

$$\mathbf{z}_k = \mathbf{A}\mathbf{y}_k, \quad \mathbf{y}_{k+1} = \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|}.$$

Dla dużego  $k$  zachodzi:

$$\mathbf{y}_k \longrightarrow \mathbf{e}_1, \quad \mathbf{z}_k = A\mathbf{y}_k \longrightarrow \lambda_1 \mathbf{e}_1,$$

a więc:

$$\lambda_1 \approx \|\mathbf{z}_k\|.$$

## 4 Druga wartość własna – iteracja z ortogonalizacją

Wektor startowy  $\mathbf{y}_1$  można rozwinąć w bazie ortonormalnych wektorów własnych macierzy  $A$ :

$$\mathbf{y}_1 = \beta_1 \mathbf{e}_1 + \beta_2 \mathbf{e}_2 + \cdots + \beta_6 \mathbf{e}_6.$$

Ponieważ macierz  $A$  jest diagonalizowalna, działanie  $A$  na taki wektor można przedstawić jako:

$$A\mathbf{y}_1 = \beta_1 \lambda_1 \mathbf{e}_1 + \beta_2 \lambda_2 \mathbf{e}_2 + \cdots + \beta_6 \lambda_6 \mathbf{e}_6.$$

W zwykłej metodzie potęgowej każdy kolejny krok mnoży wektor przez macierz  $A$ , co prowadzi do:

$$A^k \mathbf{y}_1 = \beta_1 \lambda_1^k \mathbf{e}_1 + \beta_2 \lambda_2^k \mathbf{e}_2 + \cdots + \beta_6 \lambda_6^k \mathbf{e}_6.$$

Jeżeli wartości własne spełniają

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_6|,$$

to po znormalizowaniu wektora otrzymujemy

$$\mathbf{y}_k \longrightarrow \mathbf{e}_1,$$

ponieważ składnik przy  $\lambda_1^k$  dominuje nad pozostałymi. W praktyce oznacza to, że *zwykła metoda potęgowa zawsze zbiega do  $\mathbf{e}_1$  bez względu na wartości współczynników  $\beta_i$ .*

Aby otrzymać drugi wektor własny  $\mathbf{e}_2$ , należy usunąć składową w kierunku  $\mathbf{e}_1$ , tak aby wymusić  $\beta_1 = 0$  w każdej iteracji. Innymi słowy, iterowane wektory muszą pozostać w podprzestrzeni prostopadłej do  $\mathbf{e}_1$ :

$$\mathbf{e}_1^\top \mathbf{y}_k = 0.$$

Dlatego po każdym mnożeniu wykonujemy ortogonalizację:

$$\mathbf{z}_k = A\mathbf{y}_k, \quad \mathbf{z}_k \leftarrow \mathbf{z}_k - \mathbf{e}_1(\mathbf{e}_1^\top \mathbf{z}_k),$$

czyli odejmujemy rzut wektora  $\mathbf{z}_k$  na kierunek  $\mathbf{e}_1$ . Zapewnia to, że nowy wektor

$$\mathbf{y}_{k+1} = \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|}$$

pozostaje w przestrzeni  $\{\mathbf{e}_1\}^\perp$ .

W arytmetyce dokładnej wystarczyłoby wybrać **wektor startowy** ortogonalny do  $\mathbf{e}_1$ , jednak ze względu na kumulację błędów zaokrągleń dodatkowa ortogonalizacja jest niezbędna. W przeciwnym razie niewielka składowa w kierunku  $\mathbf{e}_1$  zostałaby wzmocniona przez kolejne mnożenia przez  $A$  i dominowałaby nad składową odpowiadającą  $\lambda_2$ .

Losowo wygenerowany **wektor startowy** jest więc najpierw rzutowany na przestrzeń prostopadłą do  $\mathbf{e}_1$ :

$$\mathbf{y}_1 \leftarrow \mathbf{y}_1 - \mathbf{e}_1(\mathbf{e}_1^\top \mathbf{y}_1), \quad \mathbf{y}_1 \leftarrow \frac{\mathbf{y}_1}{\|\mathbf{y}_1\|},$$

a dopiero potem rozpoczyna się właściwa iteracja.

## 5 Implementacja

```
import numpy as np
from numpy.typing import NDArray

vector = NDArray[np.float64]
matrix = NDArray[np.float64]

def power_iteration(A: matrix, iterations: int) -> tuple[vector, np.float64]:
    y = np.random.rand(A.shape[0])
    y = y / np.linalg.norm(y)

    for _ in range(iterations):
        # obliczenie wektora Ay
        z = A @ y

        # obliczenia normy
        z_norm = np.linalg.norm(z)

        # normalizacja wektora
        y = z / z_norm

    lam = float(np.linalg.norm(z))

    return y, lam

def second_eigenpar(A: matrix, e1: vector, iterations: int) -> tuple[vector, np.float64]:
    y = np.random.rand(A.shape[0])
    y -= e1 * (e1 @ y) # rzutowanie
    y = y / np.linalg.norm(y)

    for i in range(iterations):
        z = A @ y

        # ortogonalizacja względem e1
        z = z - e1 * (e1 @ z)

        y = z / np.linalg.norm(z)

    lam = float(np.linalg.norm(z))

    return y, lam

def main():
    A: vector = np.array([[19/12, 13/12, 5/6, 5/6, 13/12, -17/12],
                          [13/12, 13/12, 5/6, 5/6, -11/12, 13/12],
                          [5/6, 5/6, 5/6, -1/6, 5/6, 5/6],
                          [5/6, 5/6, -1/6, 5/6, 5/6, 5/6],
                          [13/12, -11/12, 5/6, 5/6, 13/12, 13/12],
                          [-17/12, 13/12, 5/6, 5/6, 13/12, 19/12]], dtype=np.float64)

    e1, lam1 = power_iteration(A, 1000)
    e2, lam2 = second_eigenpar(A, e1, 1000)
    print(f"-----")
    print(f"lambda: {lam1}")
    print(f"e1: {e1}")
    print(f"lambda: {lam2}")
    print(f"e2: {e2}")

if __name__ == "__main__":
```

## 6 Wyniki

Dla rozważanej macierzy otrzymano:

```
lambda: 4.0
e1: [0.40824829 0.40824829 0.40824829 0.40824829 0.40824829 0.40824829]
lambda: 3.0000000000000004
e2: [-0.70710678 0. 0. 0. 0. 0.70710678]
```

$$\lambda_1 \approx 4.0, \quad \mathbf{e}_1 \approx \frac{1}{\sqrt{6}}(1, 1, 1, 1, 1, 1),$$

$$\lambda_2 \approx 3.0, \quad \mathbf{e}_2 \approx \frac{1}{\sqrt{2}}(-1, 0, 0, 0, 0, 1).$$

## 7 Wnioski

- Metoda potęgowa zbiega do największej wartości własnej macierzy oraz odpowiadającego jej unormowanego wektora własnego.
- Druga wartość własna wymaga iteracji z ortogonalizacją względem  $\mathbf{e}_1$ .
- Uzyskano dwie największe wartości własne:  $\lambda_1 = 4$  oraz  $\lambda_2 = 3$ .