Zadanie Numeryczne 06

Mateusz Wojtyna

1. Wstęp

Należało znaleźć przybliżony wektor własny do wartości własnej $\lambda \approx 0.38197$ macierzy

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 1 \\ -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 & -1 \\ 1 & 0 & 0 & -1 & 2 \end{bmatrix} \tag{1}$$

2. Opis

2.1. Rezolwenta

Jeżeli λ jest wartością własną macierzy $\mathbf{A}$ której wektory własne stanowią bazę w $\mathbb{R}^n$ i $\tau \approx \lambda$ nie należy do widma $\mathbf{A}$ (np. przybliżenie numeryczne λ), to macierz $(\mathbf{A} - \tau\mathbb{1})^{-1}$ jest zwana *rezolwentą*. Wtedy zaczynając od dowolnego $\mathbf{y}_1$ (z warunkiem $\|\mathbf{y}_1\| = 1$) mamy iterację

$$\begin{aligned} (\mathbf{A} - \tau\mathbb{1})^{-1}\mathbf{y}_k &= \mathbf{z}_k \\ \mathbf{y}_{k+1} &= \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|} \end{aligned} \tag{2}$$

zbiegającą się do wektora własnego macierzy $\mathbf{A}$ odpowiadającego $\lambda \approx \tau$. Pierwszą równość można przekształcić do postaci $(\mathbf{A} - \tau\mathbb{1})\mathbf{z}_k = \mathbf{y}_k$. Wtedy macierz układu jest łatwa do policzenia i można zrobić jej faktoryzację dla szybszego rozwiązania układu w każdej iteracji. Iterację zatrzymujemy jeżeli $|1 - |\mathbf{y}_{k+1} \cdot \mathbf{y}_k|\ | \leq \varepsilon$ lub jeżeli przekroczono limit kroków[1].

2.2. Algorytm Shermana-Morrisona

Widzimy że macierz $\mathbf{A} - \tau\mathbb{1}$ jest prawie trójdiagonalna, przeszkadzają jedynie wartości w rogach macierzy. Można więc skorzystać z algorytmu Shermana-Morrisona.

[1] Z powodu oscylacji znaków składowych $\mathbf{y}_k$ nie używamy $\|\mathbf{y}_{k+1} - \mathbf{y}_k\| \leq \varepsilon$.

Definiując $\tilde{\mathbf{A}} = (\mathbf{A} - \tau\mathbb{1}) - \mathbf{u}\mathbf{v}^T$ oraz biorąc $\mathbf{u} = \mathbf{v} = [1, 0, 0, 0, 1]$, widać że $\tilde{\mathbf{A}}$ jest macierzą trójdiagonalną. Teraz iteracja (2) wygląda następująco:

$$\begin{aligned} \tilde{\mathbf{A}}\mathbf{p} &= \mathbf{y}_k \\ \tilde{\mathbf{A}}\mathbf{q} &= \mathbf{u} \\ \mathbf{z}_k &= \mathbf{p} - \frac{\mathbf{v}^T\mathbf{p}}{1 + \mathbf{v}^T\mathbf{q}}\mathbf{q} \\ \mathbf{y}_{k+1} &= \frac{\mathbf{z}_k}{\|\mathbf{z}_k\|} \end{aligned} \tag{3}$$

Macierz $\tilde{\mathbf{A}}$ jest symetryczna, ale nie jest dodatnio określona, więc nie możemy zastosować faktoryzacji Cholesky'ego, zostaje nam więc faktoryzacja LU. Ponieważ macierz jest trójdiagonalna, faktoryzacja oraz każde rozwiązanie układu z innymi wyrazami wolnymi zajmie $\mathcal{O}(n)$ czasu.

3. Kod

Program napisano w Pythonie z użyciem pakietów *NumPy* oraz *Matplotlib*.

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

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



def LU_factor_tridiagonal(
    underdiag: array, diag: array, overdiag: array
) -> tuple[vector, vector]:
    n = len(diag)
    u = np.zeros(n, dtype=np.float64)
    l = np.zeros(n - 1, dtype=np.float64)

    u[0] = diag[0]

    for i in range(1, n):
        l[i - 1] = underdiag[i - 1] / u[i - 1]
        u[i] = diag[i] - l[i - 1] * overdiag[i - 1]

    return l, u



def LU_solve_tridiagonal(overdiag: array, l: vector, u: vector, b: vector) -> vector:
    n = len(b)
```

```
    # Forward substitution (Ly = b)
    y = np.zeros(n, dtype=float)
    y[0] = b[0]
    for i in range(1, n):
        y[i] = b[i] - l[i - 1] * y[i - 1]

    # Backsubstitution (Ux = y)
    x = np.zeros(n, dtype=float)
    x[-1] = y[-1] / u[-1]
    for i in range(n - 2, -1, -1):
        x[i] = (y[i] - overdiag[i] * x[i + 1]) / u[i]

    return x


def sherman_morrison(z: vector, v: vector, q: vector) -> vector:
    return z - ((np.dot(v, z)) / (1.0 + np.dot(v, q))) * q


def calc_eigenvector(
    diag: array,
    underdiag: array,
    overdiag: array,
    u_sm: vector,
    v_sm: vector,
    limit: int,
    eps: float,
):
    N = len(diag)
    y = np.ones(N, dtype=np.float64)
    y /= np.linalg.norm(y)

    l, u = LU_factor_tridiagonal(underdiag, diag, overdiag)

    for i in range(limit):
        p = LU_solve_tridiagonal(overdiag, l, u, y)
        q = LU_solve_tridiagonal(overdiag, l, u, u_sm)
        z_k = sherman_morrison(p, v_sm, q)
        y_new = z_k / np.linalg.norm(z_k)

        if abs(1 - abs(y_new @ y)) <= eps:
            return (y_new, i + 1)

        y = y_new

    return (y, limit)
```

```
def main():
    np.set_printoptions(suppress=True)

    # A - tau*I - u*v^T
    tau = 0.38197
    underdiag = np.array([-1, 1, 1, -1], dtype=np.float64)
    diag = np.array([1, 2, 1, 2, 1], dtype=np.float64)
    diag -= np.ones(len(diag), dtype=np.float64) * tau
    overdiag = np.array([-1, 1, 1, -1], dtype=np.float64)
    u = np.array([1, 0, 0, 0, 1], dtype=np.float64)
    v = np.array([1, 0, 0, 0, 1], dtype=np.float64)

    eigenvector, steps = calc_eigenvector(diag, underdiag, overdiag, u, v, 1000, 1e-12)
    print("Obliczono", eigenvector, "po", steps, "krokach")


if __name__ == "__main__":
    main()
```

4. Wyniki

Otrzymano zbieżność do wektora własnego

$$\mathbf{y} \approx \begin{bmatrix} 0.60150 \\ 0.37175 \\ 0.00000 \\ -0.37175 \\ -0.60150 \end{bmatrix} \tag{4}$$

po 6 krokach.