Metody Numeryczne - Zadanie V

Mateusz Kamiński

Grudzień 2025

1 Wprowadzenie

Celem ćwiczenia było wyznaczenie wartości i wektorów własnych zadanej macierzy hermitowskiej H poprzez jej rozszerzenie do rzeczywistej macierzy symetrycznej H_f.

2 Opis

2.1 Rozszerzenie Macierzy Hermitowskiej

Rozważano macierz hermitowską $H \in \mathbb{C}^{n \times n}$:

$$H = \begin{bmatrix} 0 & 1 & 0 & -i \\ 1 & 0 & -i & 0 \\ 0 & i & 0 & 1 \\ i & 0 & 1 & 0 \end{bmatrix} = A + iB,$$

gdzie $A = \mathrm{Re}(H)$ i $B = \mathrm{Im}(H)$ są macierzami rzeczywistymi. Problem wartości własnych $H\mathbf{v} = \lambda\mathbf{v}$ został przekształcony do problemu macierzy rzeczywistej $H_f \in \mathbb{R}^{2n \times 2n}$:

$$H_f = \begin{bmatrix} A & -B \\ B & A \end{bmatrix}.$$

Wartości własne macierzy H_f są podwojone, $\lambda(H_f) = \{\lambda_1, \lambda_1, \lambda_2, \lambda_2, \dots\}$, a wektory własne $\mathbf{v}_f \in \mathbb{R}^{2n}$ pozwalają odzyskać wektory zespolone $\mathbf{v} \in \mathbb{C}^n$ poprzez

$$\mathbf{v} = \mathrm{Re}(\mathbf{v}_f) + i \cdot \mathrm{Im}(\mathbf{v}_f).$$

2.2 Trójdiagonalizacja metodą Householdera

Macierz H_f została zredukowana do macierzy trójdiagonalnej T za pomocą transformacji Householdera:

$$T = Q^T H_f Q,$$

gdzie $Q = P_1 P_2 \dots P_{n-2}$ jest macierzą ortogonalną, będącą iloczynem macierzy Householdera $P_k = I - 2\mathbf{u}_k \mathbf{u}_k^T$. Implementacja funkcji `tridiagonalize` obliczała jednocześnie macierz T oraz macierz transformacji Q, co zapewnia poprawność numeryczną dla małych wymiarów.

2.3 Obliczenie wartości i wektorów własnych

Wartości ($\boldsymbol{\lambda}_T$) i wektory ($\mathbf{Y}$) własne macierzy trójdiagonalnej T wyznaczono funkcją `scipy.linalg.eigh_tridiagonal`. Wektory własne pierwotnej macierzy H_f odzyskano przez:

$$\mathbf{W} = Q\mathbf{Y}.$$

Ostatecznie wektory własne macierzy H uzyskano przez konwersję $\mathbf{W}$ do postaci zespolonej.

3 Implementacja

Aby uniknąć problemów numerycznych, pasy macierzy T oczyszczono z błędów rzędu 10^{-12}.

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

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

def house_holder(x: vector) -> vector:
    sigma = 1.0 if x[0] >= 0 else -1.0
    v = x.copy()
    x_norm = np.linalg.norm(x)

    if x_norm == 0.0:
        return np.zeros_like(x)

    v[0] += sigma * x_norm
    v_norm = np.linalg.norm(v)

    if v_norm == 0.0:
        return np.zeros_like(x)
    u = v / v_norm
    return u

def tridiagonalize(A: matrix):
    A = A.copy().astype(np.float64)
    n = A.shape[0]
    Q = np.eye(n, dtype=np.float64)
```

```python
    householder_vectors = []
    for k in range(n - 2):
        x = A[k + 1:, k]
        x_norm = np.linalg.norm(x)
        if x_norm < 1e-14:
            continue
        u = house_holder(x)

        u_full = np.zeros(n)
        u_full[k + 1:] = u
        householder_vectors.append(u_full)

        A[k + 1:, k:] -= 2 * np.outer(u, u @ A[k + 1:, k:])
        A[:, k + 1:] -= 2 * np.outer(A[:, k + 1:] @ u, u)

    return A, householder_vectors

def real_to_complex(w: vector) -> NDArray[np.complex128]:
    n2 = len(w)
    n = n2 // 2
    x = w[:n]
    y = w[n:]
    u = x + 1j * y
    u_norm = np.sqrt(np.vdot(u, u))
    if u_norm == 0.0:
        return u
    return u / u_norm

def apply_householders(y: matrix, householder_vectors: list[vector]) ->
    matrix:
    # Wektory własne Hf: w = P_{n-2} * ... * P_1 * y
    w = y.copy()

    # Stosujemy wektory Householdera w ODWRÓCONEJ KOLEJNOŚCI
    # do macierzy wektorów własnych y.
    for u_full in reversed(householder_vectors):
        w -= 2 * np.outer(u_full, u_full @ w)

    return w

def main():
    H = np.array([
        [0,   1,   0, -1j],
        [1,   0, -1j,   0],
        [0,  1j,   0,   1],
        [1j,  0,   1,   0]
    ], dtype=np.complex128)

    A = np.real(H)
```

```
    B = np.imag(H)

    Hf: matrix = np.block([
        [A, -B],
        [B, A]
    ]).astype(np.float64)

    T, householder_vectors = tridiagonalize(Hf)

    np.set_printoptions(precision=4, suppress=True, linewidth=np.inf)

    diagonal = np.diag(T)

    subdiagonal_raw = np.diag(T, k=-1)

    subdiagonal = np.where(np.abs(subdiagonal_raw) < 1e-12, 0.0,
        subdiagonal_raw)

    v, y = scipy.linalg.eigh_tridiagonal(diagonal, subdiagonal)

    # Przekształcenie wektorów własnych Hf: w = Q @ y
    w = apply_householders(y, householder_vectors)

    print("## Wartości Własne Hf (rzeczywiste):")
    print(v)

    print("---")
    print("## Wektory Własne H (kolumny):")

    # Konwersja wektorów własnych Hf do H
    for i in [0, 2, 4, 6]:
        eigvec_h = real_to_complex(w[:, i])
        print(f"  ż ={v[i]:.4f}: {eigvec_h}")

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

4 Wyniki

4.1 Wartości własne

Wartości własne macierzy H:

$$\boldsymbol{\lambda}_H \approx \begin{bmatrix} -2.000 \\ 0.0000 \\ 0.0000 \\ 2.0000 \end{bmatrix}$$

4.2 Wektory własne macierzy H

$$\mathbf{v}_{\lambda=-2} = \begin{bmatrix} 0.0 + 0.5\,i \\ 0.0 - 0.5\,i \\ -0.5 + 0.0\,i \\ 0.5 - 0.0\,i \end{bmatrix}, \quad \mathbf{v}_{\lambda=0} = \begin{bmatrix} 0.0 - 0.0\,i \\ 0.0 - 0.7071\,i \\ 0.0 - 0.0\,i \\ -0.7071 - 0.0\,i \end{bmatrix}$$

$$\mathbf{v}_{\lambda=0} = \begin{bmatrix} 0.0 + 0.0\,i \\ -0.7071 - 0.0\,i \\ 0.0 + 0.0\,i \\ 0.0 + 0.7071\,i \end{bmatrix}, \quad \mathbf{v}_{\lambda=2} = \begin{bmatrix} 0.5 + 0.0\,i \\ 0.5 + 0.0\,i \\ 0.0 + 0.5\,i \\ 0.0 + 0.5\,i \end{bmatrix}$$

5 Wnioski

Zaimplementowana metoda trójdiagonalizacji Householdera oraz wykorzystanie zoptymalizowanego algorytmu `eigh_tridiagonal` umożliwiły skuteczne wyznaczenie wartości i wektorów własnych macierzy hermitowskiej H. Kluczowe było przekształcenie wektorów własnych macierzy trójdiagonalnej $\mathbf{Y}$ z powrotem do przestrzeni pierwotnej za pomocą Q, tj. $\mathbf{W} = Q\mathbf{Y}$. Złożoność obliczeniowa procesu wynosi $O(n^3)$, głównie ze względu na konieczność wymnożenia macierzy $\boldsymbol{QY}$.