Metody numeryczne

Mateusz Kwolek

4. Sprowadź macierz z zadania 3 do postaci trójdiagonalnej, a następnie znajdź jej wszystkie wartości własne.

3. Dana jest macierz

$$\mathbf{A} = \begin{bmatrix} \frac{19}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & -\frac{17}{12} \\ \frac{13}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & -\frac{11}{12} & \frac{13}{12} \\ \frac{5}{6} & \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{5}{6} & \frac{5}{6} & -\frac{1}{6} & \frac{5}{6} & \frac{5}{6} & \frac{5}{6} \\ \frac{13}{12} & -\frac{11}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{13}{12} \\ -\frac{17}{12} & \frac{13}{12} & \frac{5}{6} & \frac{5}{6} & \frac{13}{12} & \frac{19}{12} \end{bmatrix}. \tag{3}$$

1) Wybór algorytmu

Algorytm Householdera jest powszechnie używany do przekształcenia danej macierzy do postaci trójdiagonalnej*. Trójdiagonalizacja jest krokiem wstępnym do zastosowania algorytmu QR dzięki, któremu będziemy w stanie znaleźć wartości własne macierzy.

**macierzy, która ma niezerowe elementy tylko na głównej przekątnej oraz na pierwszej przekątnej powyżej i poniżej głównej przekątnej*

2) Kod źródłowy

Do stworzenia programu użyłem języka Python ze względu na jego prostotę, bogactwo bibliotek oraz częste zastosowanie do wykonywania obliczeń matematycznych.

```python
import numpy as np
import copy
np.set_printoptions(suppress=True, linewidth=1000)

A = np.array([[19, 13, 10, 10, 13, -17],
              [13, 13, 10, 10, -11, 13],
              [10, 10, 10, -2, 10, 10],
              [10, 10, -2, 10, 10, 10],
              [13, -11, 10, 10, 13, 13],
              [-17, 13, 10, 10, 13, 19]]) / 12

# Trójdiagonalizacja macierzy metodą Householdera
def householder(A):
    size = len(A)
    R = copy.deepcopy(A)
    for i in range(size - 1):
        alpha = -np.sign(R[i + 1][i]) * np.linalg.norm(R[i + 1:, i])
        r = np.sqrt((alpha * alpha - R[i + 1, i] * alpha) / 2)
        v = np.zeros_like(R[i:, i])
        v[1] = (R[i + 1, i] - alpha) / (2 * r)
        v[2:] = (R[i + 2:, i] / (2 * r))
        P = np.eye(len(v)) - 2 * np.outer(v, v)
        R[i:, i:] = np.dot(np.dot(P, R[i:, i:]), P)
    return R

# Poszukiwanie wartości własnych macierzy trójdiagonalnej metodą rozkładu QR
def qr_algorithm(T, num_iter=1000, tol=1e-9):
    n = T.shape[0]
    A_k = np.copy(T)

    for _ in range(num_iter):
        Q, R = np.linalg.qr(A_k)
        A_k = R @ Q

        if np.allclose(A_k - np.diag(np.diag(A_k)), 0, atol=tol):
            break

    eigenvalues = np.diag(A_k)
    return eigenvalues

# Sprowadzenie macierzy A do postaci trójdiagonalnej
T = householder(A)

# Znalezienie wszystkich wartości własnych macierzy trójdiagonalnej
eigenvalues = qr_algorithm(T)

# Wyświetlenie wyników
print("Matrix A:\n", A)
print()
print("Tridiagonal matrix:\n", T)
print()
print("Eigenvalues:\n", eigenvalues)
```

3) Wynik

```
Matrix A:
 [[ 1.58333333  1.08333333  0.83333333  0.83333333  1.08333333 -1.41666667]
 [ 1.08333333  1.08333333  0.83333333  0.83333333 -0.91666667  1.08333333]
 [ 0.83333333  0.83333333  0.83333333 -0.16666667  0.83333333  0.83333333]
 [ 0.83333333  0.83333333 -0.16666667  0.83333333  0.83333333  0.83333333]
 [ 1.08333333 -0.91666667  0.83333333  0.83333333  1.08333333  1.08333333]
 [-1.41666667  1.08333333  0.83333333  0.83333333  1.08333333  1.58333333]]

Tridiagonal matrix:
 [[ 1.58333333 -2.39646731 -0.         -0.         -0.          0.        ]
 [-2.39646731 -0.01259573  0.93475928 -0.         -0.         -0.        ]
 [-0.          0.93475928  2.36902143 -2.07886321 -0.         -0.        ]
 [-0.         -0.         -2.07886321  0.06024096 -0.         -0.        ]
 [-0.         -0.         -0.         -0.          1.06335437 -0.24359925]
 [ 0.         -0.         -0.          0.         -0.24359925  1.93664563]]

Eigenvalues:
 [ 4.  3. -2.  2. -1.  1.]
```