

# Metody Numeryczne - Zadanie IV

Mateusz Kamiński

Listopad 2025

## 1 Wstęp

Celem zadania jest sprowadzenie macierzy  $\mathbf{A}$  do postaci trójdagonalnej, wykorzystamy do tego transformację Householdera, która nie zmienia widma macierzy  $\mathbf{A}$ . Następnie należało znaleźć jej wektory własne, wykorzystamy do tego fakt, że dowolna macierz podobna  $\mathbf{B} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ , gdzie  $\mathbf{P}$  to dowolna macierz odwracalna, ma identyczne widmo z macierzą  $\mathbf{A}$ . Do sprowadzenia macierzy do postaci diagonalnej użyjemy obrotów Givensa, aby wyeliminować elementy pod diagonalą oraz żeby w kolejnym kroku móc odczytać wartości własne.

## 2 Opis

### 2.1 Trójdagonalizacja

Rozważana w zadaniu macierz  $A$  ma postać:

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

Na początku zadania sprowadzamy macierz  $\mathbf{A}$  do postaci trójdagonalnej, wykorzystując odbicia Householdera. Transformacja Householdera pozwala wyzerować wybrane elementy w kolumnach macierzy. W praktyce, dla każdej kolumny  $k$  (poza dwoma ostatnimi) bierzemy wektor

$$\mathbf{x} = \mathbf{A}_{k+1:n, k}$$

czyli fragment kolumny rozpoczynający się poniżej diagonalnego elementu. Następnie obliczamy jego normę  $\|\mathbf{x}\|$  i na tej podstawie tworzymy wektor  $\mathbf{v}$ .

$$\mathbf{v} = \mathbf{x} + \sigma\|\mathbf{x}\|\mathbf{e}_1, \quad \sigma = \text{sign}(x_1)$$

gdzie  $\mathbf{e}_1 = (1, 0, 0, 0, 0, 0)^T$ .

Następnie normalizujemy wektor  $\mathbf{v}$ , otrzymując wektor Householdera:

$$\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}.$$

Na tej podstawie budujemy macierz odbicia

$$\mathbf{H}_k = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T,$$

która zeruje wszystkie elementy poniżej pierwszego w  $k$ -tej kolumnie. Macierz transformacji przyjmuje postać:

$$\mathbf{Q}_k = \begin{pmatrix} \mathbf{I}_k & 0 \\ 0 & \mathbf{H}_k \end{pmatrix},$$

dzięki czemu przekształca tylko odpowiedni fragment macierzy  $\mathbf{A}$ . W każdym kroku wykonujemy transformację podobieństwa

$$\mathbf{A} \leftarrow \mathbf{Q}_k^T \mathbf{A} \mathbf{Q}_k,$$

co zachowuje widmo macierzy, a jednocześnie zeruje kolejne elementy macierzy  $\mathbf{A}$ . Po wykonaniu  $n - 2$  takich kroków otrzymujemy macierz trójdziagonalną.

Uzyskana w ten sposób macierz ma postać

$$\mathbf{T} = \mathbf{Q}^T \mathbf{A} \mathbf{Q},$$

gdzie  $\mathbf{T}$  jest trójdziagonalna, a  $\mathbf{Q}$  jest iloczynem wszystkich odbić Householdera. Macierz  $\mathbf{T}$  zachowuje wszystkie wartości własne macierzy  $\mathbf{A}$ , co pozwala w dalszym etapie na efektywne zastosowanie algorytmu QR z obrotami Givensa do sprowadzenia jej do postaci diagonalnej. Otrzymana macierz  $\mathbf{T}$  wygląda:

$$\mathbf{T} = \begin{pmatrix} 1.5833 & -2.3965 & \bullet & \bullet & \bullet & \bullet \\ -2.3965 & -0.0126 & 0.9348 & \bullet & \bullet & \bullet \\ \bullet & 0.9348 & 2.3690 & -2.0789 & \bullet & \bullet \\ \bullet & \bullet & -2.0789 & 0.0602 & 0.0000 & \bullet \\ \bullet & \bullet & \bullet & 0.0000 & 1.7078 & -0.4548 \\ \bullet & \bullet & \bullet & \bullet & -0.4548 & 1.2922 \end{pmatrix}.$$

## 2.2 Szukanie wartości własnych – diagonalizacja

Gdy mamy już macierz  $\mathbf{T}$  przystępujemy do wyznaczania jej wartości własnych. W tym celu stosujemy iteracyjny algorytm QR oparty na obrotach Givensa, który w przypadku macierzy trójdziagonalnych działa wyjątkowo efektywnie (jeden obrót w czasie  $\mathcal{O}(1)$ , sumarycznie  $\mathcal{O}(n)$ ).

W każdej iteracji zerujemy kolejno elementy pod diagonalą przy pomocy rotacji Givensa skonstruowanej tak, aby wyzerować element  $\mathbf{T}_{i+1,i}$ . Obrót działa lokalnie na dwóch sąsiednich wierszach i kolumnach, co powoduje pojawienie się dodatkowego elementu poza trójdziagonalnością, który następnie jest „spychany” w dół macierzy.

Po wystarczającej liczbie iteracji macierz  $\mathbf{T}$  staje się w przybliżeniu diagonalna, a jej wartości na przekątnej stanowią przybliżenia wartości własnych macierzy  $\mathbf{A}$ .

## 2.3 Działanie obrotu Givensa

W celu wyzerowania elementu  $\mathbf{T}_{i+1,i}$  stosujemy rotację Givensa działającą na parę elementów  $(a, b)$  znajdujących się w tej samej kolumnie:

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}, \quad r = \sqrt{a^2 + b^2}, \quad c = \frac{a}{r}, \quad s = \frac{b}{r}.$$

Oznacza to, że odpowiednia kombinacja liniowa wierszy  $i$  oraz  $i + 1$  pozwala zastąpić wartość  $b$  przez zero.

PRZED OBROTEM ( $G @ T$ ):

kolumny →					
	...	i	i+1	i+2	...
-----					
i	...	a	x	y	...
i+1	...	b	u	v	...

PO OBRODZIE:

kolumny →					
	...	i	i+1	i+2	...
-----					
i	...	r	*	*	...
i+1	...	0	*	*	...

Po zastosowaniu rotacji Givensa wiersz  $i$  oraz wiersz  $i + 1$  ulegają przekształceniu według wzorów:

$$\text{new\_row}_i = c \text{row}_i + s \text{row}_{i+1}, \quad \text{new\_row}_{i+1} = -s \text{row}_i + c \text{row}_{i+1}.$$

Rotacja wyzerowała element  $T_{i+1,i}$ , natomiast pozostałe elementy w dwóch wierszach zostały odpowiednio zmodyfikowane (oznaczone jako \*).

W przypadku macierzy trójdzielnej zmieniają się wyłącznie te elementy, które leżą w sąsiedztwie diagonalnym. Przykład lokalnego fragmentu macierzy przed obrotem:

PRZED:

	i-1	i	i+1
-----			
i	x	a	c
i+1	0	b	e

Po zastosowaniu obrotu Givensa otrzymujemy strukturę:

PO:

	i-1	i	i+1
-----			
i	x'	r	c'
i+1	0	0	e'

gdzie  $r$  jest nową wartością na diagonalnej, 0 jest wyzerowanym elementem pod diagonalą, a pozostałe wpisy  $x', c', e'$  trzeba wyliczyć.

Aby zachować symetrię oraz strukturę trójdzielnej macierzy, po wykonaniu obrotu na wierszach stosujemy ten sam obrót Givensa na odpowiednich kolumnach, co odpowiada operacji

$$T \leftarrow T G^T.$$

Obrót ten działa analogicznie jak wcześniej, lecz na kolumnach  $i$  oraz  $i + 1$ :

$$\text{new\_col}_i = c \text{col}_i + s \text{col}_{i+1}, \quad \text{new\_col}_{i+1} = -s \text{col}_i + c \text{col}_{i+1}.$$

Operacja kolumnowa usuwa efekt uboczny powstały po obrocie wierszy. W wyniku przekształcenia wierszy pojawia się dodatkowy element poza trójdzielnością, w pozycji  $T_{i,i+2}$  lub  $T_{i+1,i-1}$ .

Z tego powodu zarówno obrót wierszy

$$\mathbf{T} \leftarrow \mathbf{G} \mathbf{T},$$

jak i obrót kolumn

$$\mathbf{T} \leftarrow \mathbf{T} \mathbf{G}^T,$$

muszą działać lokalnie w zakresie od  $i - 1$  do  $i + 2$ . Tylko w tym otoczeniu mogą pojawić się niezerowe liczby po transformacji; wszystkie pozostałe elementy macierzy pozostają równe zero.

## 2.4 Algorytm QR

W klasycznym algorytmie QR wykonujemy faktoryzację

$$\mathbf{T}_k = \mathbf{Q}_k \mathbf{R}_k,$$

a następnie aktualizujemy macierz według wzoru

$$\mathbf{T}_{k+1} = \mathbf{R}_k \mathbf{Q}_k.$$

Macierz  $\mathbf{Q}_k$  jest iloczynem ortogonalnych rotacji Givensa, które zerują kolejne elementy pod diagonalą w macierzy  $\mathbf{T}_k$ . Gdybyśmy zapisywali wszystkie te rotacje jawnie, otrzymalibyśmy

$$\mathbf{Q}_k = \mathbf{G}_1^T \mathbf{G}_2^T \cdots \mathbf{G}_m^T,$$

gdzie każda macierz  $\mathbf{G}_j$  jest obrotem Givensa działającym na dwóch sąsiednich wierszach.

W mojej implementacji nie tworzymy jednak macierzy  $\mathbf{Q}_k$  jawnie. Zamiast tego stosujemy każdą rotację Givensa bezpośrednio do macierzy  $\mathbf{T}_k$ . Najpierw wykonujemy mnożenie z lewej strony

$$\mathbf{T}_k \leftarrow \mathbf{G}_j \mathbf{T}_k,$$

co odpowiada części operacji  $\mathbf{Q}_k^T \mathbf{T}_k$ , a następnie mnożymy przez odpowiednią rotację z prawej strony

$$\mathbf{T}_k \leftarrow \mathbf{T}_k \mathbf{G}_j^T,$$

co realizuje część operacji  $\mathbf{T}_k \mathbf{Q}_k$ .

Ponieważ w klasycznym zapisie QR zachodzi

$$\mathbf{R}_k = \mathbf{Q}_k^T \mathbf{T}_k,$$

możemy zapisać aktualizację macierzy w postaci

$$\mathbf{T}_{k+1} = (\mathbf{Q}_k^T \mathbf{T}_k) \mathbf{Q}_k,$$

czyli w pełnej postaci przekształcenia podobieństwa

$$\mathbf{T}_{k+1} = \mathbf{Q}_k^T \mathbf{T}_k \mathbf{Q}_k.$$

### 3 Implementacja

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

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

def house_holder(x: vector) -> vector:
    # znak x[0]
    sigma = 1.0 if x[0] >= 0 else -1.0

    # odbicie wektora
    v = x.copy()
    x_norm = np.linalg.norm(x)
    v[0] += sigma * x_norm

    # normalizacja wektora Householdera u
    v_norm = np.linalg.norm(v)
    u = v / v_norm

    return u

def tridiagonalize(A: matrix) -> tuple[matrix, matrix]:
    A = A.copy()
    n = A.shape[0]

    for k in range(n - 2):
        # opuszczamy pierwszy element
        x = A[k + 1:, k]

        u = house_holder(x)

        # # tworzenie macierzy Householdera
        # H_small = np.eye(n - k - 1) - 2.0 * np.outer(u, u)
        # # wstawiamy H_small w odpowiednie miejsce dużej macierzy Q_k
        # Q_k = np.eye(n)
        # Q_k[k + 1:, k + 1:] = H_small

        # transformacja podobieństwa
        # A = (Q_k.T @ A) @ Q_k
        # prowadzi do  $O(n^4)$ , robimy zatem inaczej:

        # -----

        # P= I 2uuT
        # PA= A 2u (uTA)
        # AP= A 2 (Au)uT

        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

def qr_step_tridiagonal(T: matrix) -> matrix:
    n = T.shape[0]
    T = T.copy()

    # QR T = Q R za pomocą rotacji Givensa (zerujemy elementy poddiagonalne)
    for i in range(n - 1):
        a = T[i, i]
        b = T[i + 1, i]
```

```

        r = np.hypot(a, b)
        c = a / r
        s = b / r

        apply_givens_rows(T, i, i+1, c, s)
        apply_givens_cols(T, i, i+1, c, s)

    return T

def apply_givens_rows(T, i, j, c, s):
    """
    Zastosowanie rotacji Givensa G na wierszach i,j:  $T := G @ T$ .
    """
    n = T.shape[0]

    left = max(0, i - 1)
    right = min(n - 1, i + 2)

    for col in range(left, right + 1):
        Ti = T[i, col]
        Tj = T[j, col]

        T[i, col] = c * Ti + s * Tj
        T[j, col] = -s * Ti + c * Tj

def apply_givens_cols(T, i, j, c, s):
    """
    Zastosowanie rotacji Givensa G na kolumnach i,j:  $T := T @ G^T$ .
    """
    n = T.shape[0]

    top = max(0, i - 1)
    bottom = min(n - 1, i + 2)

    for row in range(top, bottom + 1):
        Ti = T[row, i]
        Tj = T[row, j]

        T[row, i] = c * Ti + s * Tj
        T[row, j] = -s * Ti + c * Tj

def main():
    A: matrix = 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)

    T = tridiagonalize(A)
    np.set_printoptions(linewidth=np.inf)
    print(T)
    print("\n\nAfter diagonal ----- \n\n")
    eps = 1e-12
    max_iter = 1000

    for it in range(max_iter):
        diag_old = np.diag(T).copy()
        T = qr_step_tridiagonal(T)

```

```

diag_new = np.diag(T)

# sprawdzamy maksymalną zmianę przekątnej
diff = np.linalg.norm(diag_new - diag_old)
if diff < eps:
    print(f"Diagonal converged after {it} iterations, diff = {diff:.2e}")
    break
print(T)

if __name__ == "__main__":
    main()

```

### 3.1 Wyniki

Po wykonaniu redukcji do postaci trójdzielnej oraz kolejnych iteracji algorytmu QR, macierz  $T_k$  ulega diagonalizacji. Na przekątnej macierzy końcowej otrzymujemy przybliżone wartości własne macierzy początkowej  $A$ :

$$\lambda_1 = 4, \quad \lambda_2 = 3, \quad \lambda_3 = 2, \quad \lambda_4 = 1, \quad \lambda_5 = -1, \quad \lambda_6 = -2.$$

Są to wartości własne macierzy  $A$  uzyskane jedynie na podstawie transformacji podobieństwa, dzięki czemu zachowują pełną poprawność numeryczną. Elementy pozadiagonalne macierzy  $T_k$  są śmieciami rzędu  $10^{-9}$ – $10^{-16}$ , co pozwala traktować macierz jako diagonalną.

Końcowa macierz po 74 iteracjach, z błędem  $\epsilon = 10^{-12}$  ma postać:

$$A_{\text{diag}} \approx \begin{pmatrix} 4.00000000 & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & 3.00000000 & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & -2.00000000 & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & 2.00000000 & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & -1.00000000 & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & 1.00000000 \end{pmatrix}$$

## 4 Podsumowanie

- Zastosowano transformację Householdera do sprowadzenia macierzy  $A$  do postaci trójdzielnej, co zmniejszyło koszt dalszych obliczeń z rzędu  $O(n^3)$  do  $O(n)$  na każdą iterację QR.
- Wykorzystano rotacje Givensa do wykonywania kolejnych iteracji algorytmu QR bez jawnego wyznaczania macierzy  $Q_k$  i  $R_k$ .
- Przekształcenie podobieństwa  $T_{k+1} = Q_k^T T_k Q_k$  umożliwiło zachowanie wartości własnych macierzy.