Zadanie Numeryczne 01

Mateusz Wojtyna

1. Wstęp

Należało rozwiązać poniższy układ równań $\mathbf{A}_1\mathbf{x} = \mathbf{b}$:

$$\begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}$$

Dzięki własnościom macierzy $\mathbf{A}_1$ (symetryczna, dodatnio określona, *prawie* trójdiagonalna) udało się stworzyć algorytm do rozwiązania układu korzystając ze wzoru Shermana-Morrisona oraz faktoryzacji Cholesky'ego. Algorytm posiada złożoność czasową i pamięciową $\mathcal{O}(n)$.

2. Opis

2.1. Uproszczenie obliczeń

Wzór Shermana-Morrisona

$$\mathbf{A}_1^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

pozwala na wyrażenie odwrotności macierzy $\mathbf{A}_1$ przez odwrotność macierzy $\mathbf{A}$, gdzie

$$\mathbf{A} = \mathbf{A}_1 - \mathbf{u}\mathbf{v}^T$$
$$\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$$

Przyda się to do naszego układu, gdyż możemy stworzyć macierz trójdiagonalną $\mathbf{A}$, dla których istnieją szybkie algorytmy:

$$\mathbf{A} = \mathbf{A}_1 - \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{u}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{v}^T} = \mathbf{A}_1 - \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 \end{bmatrix}$$

Otrzymując $\mathbf{A}^{-1}$ i zarazem $\mathbf{A}_1^{-1}$ moglibyśmy już rozwiązać układ równań, ale z powodu wysokiego kosztu numerycznego znalezienia jawnej odwrotności macierzy, rozwiążemy zamiast tego równanie:

$$\mathbf{x} = \mathbf{A}_1^{-1}\mathbf{b} = \left(\mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}\right)\mathbf{b}$$

co sprowadza się do rozwiązania poniższych równań ($\mathbf{A}$, $\mathbf{b}, \mathbf{u}, \mathbf{v}$ są znane):

$$\mathbf{A}\mathbf{z} = \mathbf{b} \tag{1}$$

$$\mathbf{A}\mathbf{q} = \mathbf{u} \tag{2}$$

$$\mathbf{x} = \mathbf{z} - \frac{\mathbf{v}^T\mathbf{z}}{1 + \mathbf{v}^T\mathbf{q}}\mathbf{q} \tag{3}$$

2.2. Rozwiązanie równań

Ponieważ macierz $\mathbf{A}$ jest symetryczna i dodatnio określona, możemy wykorzystać faktoryzację Cholesky'ego:

$$\mathbf{A} = \mathbf{C}\mathbf{C}^T$$

gdzie $\mathbf{C}$ to macierz trójkątna dolna. Umożliwia to rozwiązanie równań (1), (2) w sposób efektywny obliczeniowo, wykorzystując *forward substitution* i *backsubstitution*:

$$\mathbf{A}\mathbf{z} = \mathbf{b} \implies \mathbf{C}\underbrace{(\mathbf{C}^T\mathbf{z})}_{\mathbf{y}} = \mathbf{b} \implies \begin{cases} \mathbf{C}\mathbf{y} = \mathbf{b} & \text{(forward substitution)} \\ \mathbf{C}^T\mathbf{z} = \mathbf{y} & \text{(backsubstitution)} \end{cases}$$

$$\mathbf{A}\mathbf{q} = \mathbf{u} \implies \mathbf{C}\underbrace{(\mathbf{C}^T\mathbf{q})}_{\mathbf{y}} = \mathbf{u} \implies \begin{cases} \mathbf{C}\mathbf{y} = \mathbf{u} & \text{(forward substitution)} \\ \mathbf{C}^T\mathbf{q} = \mathbf{y} & \text{(backsubstitution)} \end{cases}$$

Gdy już obliczyliśmy wektory $\mathbf{z}$ oraz $\mathbf{q}$ wystarczy podstawić je do równania (3) oraz wykonać proste operacje dla wektorów i liczb, aby otrzymać wynik.

3. Kod

Program napisano w Pythonie z użyciem pakietu *NumPy*.

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

# Dla przejrzystości kodu
array = NDArray[np.float64]
vector = NDArray[np.float64]


def cholesky_tridiagonal(diag: array, subdiag: array) -> tuple[array, array]:
    """
    Zwraca (diagonala C, poddiagonala C)
    """

    N = len(diag)
    C_diag = np.zeros(N, dtype=np.float64)
    C_subdiag = np.zeros(N - 1, dtype=np.float64)

    C_diag[0] = np.sqrt(diag[0])
    for i in range(1, N):
        C_subdiag[i - 1] = subdiag[i - 1] / C_diag[i - 1]
        C_diag[i] = np.sqrt(diag[i] - C_subdiag[i - 1] ** 2)

    return C_diag, C_subdiag


def back_substitution_tridiagonal(diag: array, subdiag: array, b: vector) -> vector:
    N = len(diag)
    x = np.zeros(N, dtype=np.float64)

    x[-1] = b[-1] / diag[-1]
    for i in range(N - 2, -1, -1):
        x[i] = (b[i] - subdiag[i] * x[i + 1]) / diag[i]

    return x


def forward_substitution_tridiagonal(diag: array, subdiag: array, b: vector) -> vector:
    N = len(diag)
    x = np.zeros(N, dtype=np.float64)

    x[0] = b[0] / diag[0]
    for i in range(1, N):
        x[i] = (b[i] - subdiag[i - 1] * x[i - 1]) / diag[i]

    return x


def solve_cholesky_tridiagonal(C_diag: array, C_subdiag: array, b: vector) -> vector:
    # Ax=b <=> C*(C^T*x)=b, C^T*x=y, Cy=b
```

```
    y = forward_substitution_tridiagonal(C_diag, C_subdiag, b)
    x = back_substitution_tridiagonal(C_diag, C_subdiag, y)
    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 main():
    np.set_printoptions(linewidth=np.inf)  # pyright: ignore[reportArgumentType]

    # Rozpisanie elementów diagonalnych macierzy A, oszczędzamy pamięć!
    A_diag = np.array([3, 4, 4, 4, 4, 4, 3], dtype=np.float64)
    A_subdiag = np.array([1, 1, 1, 1, 1, 1], dtype=np.float64)

    b = np.array([1, 2, 3, 4, 5, 6, 7], dtype=np.float64)
    u = v = np.array([1, 0, 0, 0, 0, 0, 1], dtype=np.float64)

    # 1. Faktoryzacja Cholesky'ego, O(n)
    C_diag, C_subdiag = cholesky_tridiagonal(A_diag, A_subdiag)

    # 2. Rozwiązanie Az=b, O(n)
    z = solve_cholesky_tridiagonal(C_diag, C_subdiag, b)

    # 3. Rozwiązanie Aq=u, O(n)
    q = solve_cholesky_tridiagonal(C_diag, C_subdiag, u)

    # 4. Rozwiąż równanie z wykorzystaniem wzoru Shermana-Morrisona, O(n)
    x = sherman_morrison(z, v, q)
    print("Wynik:", x)

    # Wypisz różnicę
    A_1 = np.array(
        [
            [4, 1, 0, 0, 0, 0, 1],
            [1, 4, 1, 0, 0, 0, 0],
            [0, 1, 4, 1, 0, 0, 0],
            [0, 0, 1, 4, 1, 0, 0],
            [0, 0, 0, 1, 4, 1, 0],
            [0, 0, 0, 0, 1, 4, 1],
            [1, 0, 0, 0, 0, 1, 4],
        ],
        dtype=np.float64,
    )
    print("Różnica:", np.matmul(A_1, x) - b)


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

4. Wyniki

Po wykonaniu programu otrzymano wyniki:

$$\mathbf{x} = \begin{bmatrix} -0.26016260 \\ 0.44715447 \\ 0.47154472 \\ 0.66666667 \\ 0.86178862 \\ 0.88617886 \\ 1.59349593 \end{bmatrix}$$

Po przemnożeniu $\mathbf{A}_1 \cdot \mathbf{x}$ otrzymano wyniki bardzo bliskie $\mathbf{b}$, różnice rzędu 10^{-16}:

$$\mathbf{A}_1 \cdot \mathbf{x} - \mathbf{b} = \begin{bmatrix} -1.11022302 \times 10^{-16} \\ -2.22044605 \times 10^{-16} \\ 0 \\ 0 \\ 0 \\ 0 \\ -8.88178420 \times 10^{-16} \end{bmatrix}$$

5. Podsumowanie

Metoda z użyciem wzoru Shermana-Morrisona oraz faktoryzacji Cholesky'ego umożliwia szybkie i dokładne obliczenie układu równań z macierzą *prawie* trójdiagonalną, którą można sprowadzić do trójdiagonalnej przez odjęcie od niej macierzy powstałej z mnożenia dwóch wektorów. Dzięki trójdiagonalnej strukturze macierzy faktoryzacja Cholesky'ego, *backsubstitution*, oraz *forward substitution* są możliwe do wykonania w czasie $\mathcal{O}(n)$. Wystarczy również wykorzystać jedynie $\mathcal{O}(n)$ pamięci, ponieważ tylko $3n-2$ elementów może różnić się od zera.