

Metody Numeryczne – Zadanie I

Mateusz Kamiński

Październik 2025

1. Wstęp

Celem zadania było rozwiązanie układu równań liniowych z macierzą rzadką o strukturze prawie trójdzielnej, która posiada dodatkowe jedynki w narożnikach. Macierz była symetryczna oraz dodatnio określona, co umożliwiło zastosowanie metody opartej na dekompozycji Choleskiego w połączeniu ze wzorem Shermana–Morrisona. Zastosowane podejście pozwoliło efektywnie rozwiązać układ z macierzą o postaci $\mathbf{A}_1 + \mathbf{u}\mathbf{v}^T$, bez potrzeby odwracania pełnej macierzy.

2. Opis

Rozpatrywany układ równań liniowych ma postać:

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 4 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{bmatrix}.$$
$$\mathbf{A}\mathbf{w} = \mathbf{b},$$

gdzie macierz \mathbf{A} ma strukturę prawie trójdzielną, z dodatkowymi jedynkami w narożnikach. Taka macierz może być przedstawiona jako:

$$\mathbf{A} = \mathbf{A}_1 + \mathbf{u}\mathbf{v}^T,$$

gdzie \mathbf{A}_1 jest macierzą trójdzielną, symetryczną oraz dodatnio określoną, natomiast \mathbf{u} i \mathbf{v} są wektorami kolumnowymi.

Rozkład Choleskiego

Ponieważ macierz \mathbf{A}_1 jest symetryczna i dodatnio określona, można ją rozłożyć na iloczyn:

$$\mathbf{A}_1 = \mathbf{C}\mathbf{C}^T,$$

gdzie \mathbf{C} jest macierzą dolnotrójkątną. Rozkład Choleskiego umożliwia rozwiązanie układów równań z macierzą \mathbf{A}_1 w sposób bardzo efektywny obliczeniowo, przy użyciu *forward substitution* oraz *back substitution*:

$$\mathbf{A}_1 \mathbf{z} = \mathbf{b} \quad \Rightarrow \quad \begin{cases} \mathbf{C} \mathbf{y} = \mathbf{b}, \\ \mathbf{C}^T \mathbf{z} = \mathbf{y}, \end{cases} \quad \mathbf{A}_1 \mathbf{q} = \mathbf{u} \quad \Rightarrow \quad \begin{cases} \mathbf{C} \mathbf{r} = \mathbf{u}, \\ \mathbf{C}^T \mathbf{q} = \mathbf{r}. \end{cases}$$

Dzięki temu możemy obliczyć wektory \mathbf{z} oraz \mathbf{q} bez potrzeby wyznaczania odwrotności macierzy \mathbf{A}_1 .

Algorytm Shermana–Morrisona

Wzór Shermana–Morrisona pozwala efektywnie wyznaczyć odwrotność macierzy, która różni się od znanej macierzy \mathbf{A}_1 o składnik rzędu 1:

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

Po wprowadzeniu oznaczeń:

$$\mathbf{A}_1 \mathbf{z} = \mathbf{b}, \quad \mathbf{A}_1 \mathbf{q} = \mathbf{u},$$

możemy zapisać rozwiązanie w postaci:

$$\mathbf{w} = \mathbf{z} - \frac{\mathbf{v}^T \mathbf{z}}{1 + \mathbf{v}^T \mathbf{q}} \mathbf{q}.$$

Ostatecznie obliczenie wektora \mathbf{w} sprowadza się do rozwiązania dwóch układów równań liniowych z tą samą macierzą \mathbf{A}_1 , a następnie wykonania prostych operacji skalarnych.

Analiza obliczeń

W Algorytmie Shermana–Morrisona należy rozwiązać dwa układy równań z tą samą macierzą \mathbf{A}_1 . Macierz ta jest **symetryczna**, **dodatnio określona** oraz **trójdzielna**, co pozwala przeprowadzić rozkład Choleskiego $\mathbf{A}_1 = \mathbf{C} \mathbf{C}^T$ w czasie liniowym $\mathbf{O}(n)$, ponieważ w trakcie faktoryzacji nie powstają elementy poza trzema głównymi diagonalami.

Następnie rozwiązanie układów:

$$\mathbf{A}_1 \mathbf{z} = \mathbf{b}, \quad \mathbf{A}_1 \mathbf{q} = \mathbf{u}$$

wymaga dwóch podstawień:

- **forward substitution** — rozwiązanie $\mathbf{C} \mathbf{y} = \mathbf{b}$ lub $\mathbf{C} \mathbf{r} = \mathbf{u}$,
- **back substitution** — rozwiązanie $\mathbf{C}^T \mathbf{z} = \mathbf{y}$ lub $\mathbf{C}^T \mathbf{q} = \mathbf{r}$.

Oba te etapy mają złożoność obliczeniową jak i pamięciową $\mathbf{O}(n)$, ponieważ macierz \mathbf{C} jest również trójdzielna. Końcowy krok obliczenia wektora \mathbf{w} : wymaga jedynie dwóch iloczynów skalarnych i operacji wektorowych, co również kosztuje $\mathbf{O}(n)$.

Łączna złożoność całego algorytmu wynosi więc $\mathbf{O}(n)$.

3. Kod

```
1 import numpy as np
2
3 def cholesky_tridiagonal(diagonal: np.ndarray, subdiagonal: np.ndarray)
  -> np.ndarray:
4     n = diagonal.shape[0]
5     C_diag = np.zeros(n, dtype=np.float64)
6     C_subdiag = np.zeros(n - 1, dtype=np.float64)
7
8     C_diag[0] = np.sqrt(diagonal[0])
9     for i in range(1, n):
10         C_subdiag[i - 1] = subdiagonal[i - 1] / C_diag[i - 1]
11         C_diag[i] = np.sqrt(diagonal[i] - C_subdiag[i - 1] ** 2)
12
13     return C_diag, C_subdiag
14
15
16 def forward_substitution_tridiagonal(diagonal: np.ndarray, subdiagonal:
  np.ndarray, b: np.ndarray) -> np.ndarray:
17     n = b.shape[0]
18     y = np.zeros(n, dtype=np.float64)
19
20     y[0] = b[0] / diagonal[0]
21     for i in range(1, n):
22         y[i] = (b[i] - subdiagonal[i - 1] * y[i - 1]) / diagonal[i]
23
24     return y
25
26
27 def backward_substitution_tridiagonal(diagonal: np.ndarray, subdiagonal:
  np.ndarray, y: np.ndarray) -> np.ndarray:
28     n = y.shape[0]
29     z = np.zeros(n, dtype=np.float64)
30
31     z[-1] = y[-1] / diagonal[-1]
32     for i in range(n - 2, -1, -1):
33         z[i] = (y[i] - subdiagonal[i] * z[i + 1]) / diagonal[i]
34
35     return z
36
37 def sherman_morrison_formula(z: np.ndarray, q: np.ndarray, v: np.ndarray
  ) -> np.ndarray:
38     """
39     wzór Shermana i Morrisona :
40     
$$w = z - (v^T z) / (1 + v^T q) * q$$

41     """
42
43     vTz = np.dot(v, z)
44     vTq = np.dot(v, q)
45     alpha = float(vTz / (1.0 + vTq))
46
47     return z - alpha * q
48
49
50 def main():
51     #  $A = A1 + uv^T$ 
52     diagonal = np.array([3, 4, 4, 4, 4, 4, 3], dtype=np.float64)
```

```

53 subdiagonal = np.array([1, 1, 1, 1, 1, 1, 1], dtype=np.float64)
54
55 b = np.array([1, 2, 3, 4, 5, 6, 7], dtype=np.float64)
56 v = np.array([1, 0, 0, 0, 0, 0, 1], dtype=np.float64)
57 u = np.array([1, 0, 0, 0, 0, 0, 1], dtype=np.float64)
58
59 # --- Krok (a): A z = b ---
60 diag_C, sub_C = cholesky_tridiagonal(diagonal, subdiagonal)
61
62 y = forward_substitution_tridiagonal(diag_C, sub_C, b)
63 z = backward_substitution_tridiagonal(diag_C, sub_C, y)
64
65 # --- Krok (b): A q = u ---
66 r = forward_substitution_tridiagonal(diag_C, sub_C, u)
67 q = backward_substitution_tridiagonal(diag_C, sub_C, r)
68
69 # --- Krok (c) rozwiązanie ---
70 w = sherman_morrison_formula(z, q, v)
71
72 # === ŁADNY PRINT ===
73 print("\n" + "=" * 60)
74 print(" ROZWIĄZANIE UKŁADU (A1 + u   v ) w = b ")
75 print("=" * 60)
76 print(f"{'i':>3} | {'w[i]':>10}")
77 print("-" * 18)
78 for i, val in enumerate(w, start=1):
79     print(f"{'i':>3} | {'val':>10.10f}")
80 print("-" * 18)
81
82 # Macierz wyłącznie do sprawdzenia poprawności rozwiązania
83 A1 = np.array([
84     [3, 1, 0, 0, 0, 0, 0],
85     [1, 4, 1, 0, 0, 0, 0],
86     [0, 1, 4, 1, 0, 0, 0],
87     [0, 0, 1, 4, 1, 0, 0],
88     [0, 0, 0, 1, 4, 1, 0],
89     [0, 0, 0, 0, 1, 4, 1],
90     [0, 0, 0, 0, 0, 1, 3]
91 ], dtype=np.float64)
92
93 A = A1 + np.outer(u, v)
94 check = A @ w
95 ok = np.allclose(check, b)
96
97 # === Weryfikacja równania ===
98 print("\nWeryfikacja równania A w      b:\n")
99 print(f"{'i':>3} | {'A w':>10} | {'b':>10} | {'różnica':>10}")
100 print("-" * 38)
101 for i in range(len(b)):
102     diff = check[i] - b[i]
103     print(f"{'i + 1':>3} | {'check[i]:>10.6f} | {'b[i]:>10.6f} | {'diff
104 :>10.2e}")
105 print("-" * 38)
106 print(f"\n Układ spełniony: {ok}\n")
107 print("=" * 60)
108 if __name__ == "__main__":
109     main()

```

4. Wyniki

Po wykonaniu programu otrzymano rozwiązanie:

$$\mathbf{w} = \begin{bmatrix} -0.26016 \\ 0.44715 \\ 0.47154 \\ 0.66667 \\ 0.86179 \\ 0.88618 \\ 1.59350 \end{bmatrix}$$

oraz potwierdzenie zgodności układu $\mathbf{A} \cdot \mathbf{w} \approx \mathbf{b}$, przy czym błędy resztowe są rzędu 10^{-16} .

Porównanie wartości $\mathbf{A} \cdot \mathbf{w}$ i \mathbf{b}

i	$\mathbf{A} \cdot \mathbf{w}$	\mathbf{b}	Różnica
1	1.000000	1.000000	$-1.11 \cdot 10^{-16}$
2	2.000000	2.000000	0.00
3	3.000000	3.000000	$-4.44 \cdot 10^{-16}$
4	4.000000	4.000000	0.00
5	5.000000	5.000000	$-8.88 \cdot 10^{-16}$
6	6.000000	6.000000	0.00
7	7.000000	7.000000	0.00

5. Analiza wyników

Otrzymane wyniki potwierdzają, że zastosowanie wzoru Shermana–Morrisona umożliwia szybkie rozwiązanie układu z macierzą o strukturze zbliżonej do trójdzielnej, bez konieczności pełnego odwracania macierzy. Dekompozycja Choleskiego pozwoliła na liniową złożoność obliczeniową $\mathcal{O}(\mathbf{n})$ dla macierzy \mathbf{A}_1 , a poprawka rzędu $\mathbf{u}\mathbf{v}^T$ została wykonana w czasie $\mathcal{O}(\mathbf{n})$, co czyni metodę bardzo wydajną. Resztowe wartości $(\mathbf{A} \cdot \mathbf{w} - \mathbf{b})$ są rzędu 10^{-16} , co oznacza wysoką dokładność rozwiązania i stabilność numeryczną całego algorytmu.

6. Wnioski

Metoda wykorzystująca dekompozycję Choleskiego i wzór Shermana–Morrisona jest wydajnym i stabilnym narzędziem do rozwiązywania układów z macierzami o strukturze zbliżonej do trójdzielnej, lecz zawierających dodatkowe elementy spoza głównej przekątnej. Pozwala na znaczną redukcję liczby operacji arytmetycznych i eliminuje konieczność obliczania odwrotności macierzy, zachowując wysoką dokładność numeryczną.