Mnożenie skompresowanych macierzy

Maciej Borowiec

18 stycznia 2026

1 Przebieg algorytmów

1.1 Mnożenie skompresowanych macierzy

Funkcja **matmul** przyjmuje korzenie drzew v i u reprezentujących dwie skompresowane macierze i zwraca wynik mnożenia tych macierzy.

- Jeśli v i u to liście drzewa:
  - Jeśli $v.\text{rank} = 0$ lub $u.\text{rank} = 0$ to zwracamy macierz zer w odpowiednim rozmiarze.
  - W przeciwnym wypadku zwracamy $v.U \cdot (v.V \cdot u.U) \cdot u.V$
- Jeśli ani v, ani u nie są liściami drzewa to zwracamy:

  $$\begin{bmatrix} \textbf{matmul}(v_0, u_0) + \textbf{matmul}(v_1, u_2) & \textbf{matmul}(v_0, u_1) + \textbf{matmul}(v_1, u_3) \\ \textbf{matmul}(v_2, u_0) + \textbf{matmul}(v_3, u_2) & \textbf{matmul}(v_2, u_1) + \textbf{matmul}(v_3, u_3) \end{bmatrix}$$

  gdzie v_i, u_i to i-te dziecko danego wierzchołka drzewa.
- Jeśli v to liść, a u nie to zwracamy:

  $$\begin{bmatrix} \textbf{matmul}(U_1V_1, u_0) + \textbf{matmul}(U_1V_2, u_2) & \textbf{matmul}(U_1V_1, u_1) + \textbf{matmul}(U_1V_2, u_3) \\ \textbf{matmul}(U_2V_1, u_0) + \textbf{matmul}(U_2V_2, u_2) & \textbf{matmul}(U_2V_1, u_1) + \textbf{matmul}(U_2V_2, u_3) \end{bmatrix}$$

  gdzie u_i to i-te dziecko u oraz $v.U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$, $v.V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$
- Jeśli u to liść, a v nie to zwracamy:

  $$\begin{bmatrix} \textbf{matmul}(v_0, U_1V_1) + \textbf{matmul}(v_1, U_2V_1) & \textbf{matmul}(v_0, U_1V_2) + \textbf{matmul}(v_1, U_2V_2) \\ \textbf{matmul}(v_2, U_1V_1) + \textbf{matmul}(v_3, U_2V_1) & \textbf{matmul}(v_2, U_1V_2) + \textbf{matmul}(v_3, U_2V_2) \end{bmatrix}$$

  gdzie v_i to i-te dziecko v oraz $u.U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$, $u.V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$

1.2 Mnożenie skompresowanej macierzy przez wektor

Funkcja **vecmul** przyjmuje korzeń drzewa v reprezentującego skompresowaną macierz oraz wektor $\mathbf{x}$, a zwraca wynik mnożenia macierzy przez wektor.

- Jeśli v to liść drzewa:
  - Jeśli $v.\text{rank} = 0$ to zwracamy wektor zer o odpowiedniej długości.
  - W przeciwnym wypadku zwracamy $v.U \cdot (v.V \cdot \mathbf{x})$
- W przeciwnym wypadku zwracamy:

$$\begin{bmatrix} \mathbf{vecmul}(v_0, \mathbf{x}_1) + \mathbf{vecmul}(v_1, \mathbf{x}_2) \\ \mathbf{vecmul}(v_2, \mathbf{x}_1) + \mathbf{vecmul}(v_3, \mathbf{x}_2) \end{bmatrix}$$

  gdzie v_i to i-te dziecko v oraz $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}$

2 Kod

```python
# Multiply compressed matrices
def multiply_matrices(v, u):
    # both leaves
    if len(v.children) == 0 and len(u.children) == 0:
        if v.rank == 0:
            r_start, r_end, c_start, c_end = v.region
            return np.zeros((r_end-r_start, c_end-c_start))
        elif u.rank == 0:
            r_start, r_end, c_start, c_end = u.region
            return np.zeros((r_end-r_start, c_end-c_start))
        else:
            return v.U@(v.V@u.U)@u.V

    # neither leaves
    if len(v.children) > 0 and len(u.children) > 0:
        res1 = multiply_matrices(v.children[0], u.children[0])
             + multiply_matrices(v.children[1], u.children[2])
        res2 = multiply_matrices(v.children[0], u.children[1])
             + multiply_matrices(v.children[1], u.children[3])
        res3 = multiply_matrices(v.children[2], u.children[0])
             + multiply_matrices(v.children[3], u.children[2])
        res4 = multiply_matrices(v.children[2], u.children[1])
             + multiply_matrices(v.children[3], u.children[3])

    # v leaf
    if len(v.children) == 0 and len(u.children) > 0:
        if v.rank == 0:
            r_start, r_end, c_start, c_end = v.region
            return np.zeros((r_end-r_start, c_end-c_start))
        half_U = v.U.shape[0] // 2
        half_V = v.V.shape[1] // 2
        U1 = v.U[:half_U, :]
        U2 = v.U[half_U:, :]
        V1 = v.V[:, :half_V]
        V2 = v.V[:, half_V:]
        U1V1 = Node(U1, V1)
        U1V2 = Node(U1, V2)
        U2V1 = Node(U2, V1)
        U2V2 = Node(U2, V2)

        res1 = multiply_matrices(U1V1, u.children[0])
             + multiply_matrices(U1V2, u.children[2])
        res2 = multiply_matrices(U1V1, u.children[1])
             + multiply_matrices(U1V2, u.children[3])
        res3 = multiply_matrices(U2V1, u.children[0])
             + multiply_matrices(U2V2, u.children[2])
        res4 = multiply_matrices(U2V1, u.children[1])
             + multiply_matrices(U2V2, u.children[3])

    # u leaf
    if len(v.children) > 0 and len(u.children) == 0:
        if u.rank == 0:
            r_start, r_end, c_start, c_end = u.region
            return np.zeros((r_end-r_start, c_end-c_start))
        half_U = u.U.shape[0] // 2
        half_V = u.V.shape[1] // 2
        U1 = u.U[:half_U, :]
        U2 = u.U[half_U:, :]
        V1 = u.V[:, :half_V]
        V2 = u.V[:, half_V:]
        U1V1 = Node(U1, V1)
        U1V2 = Node(U1, V2)
        U2V1 = Node(U2, V1)
        U2V2 = Node(U2, V2)

```

```
        res1 = multiply_matrices(v.children[0], U1V1)
             + multiply_matrices(v.children[1], U2V1)
        res2 = multiply_matrices(v.children[0], U1V2)
             + multiply_matrices(v.children[1], U2V2)
        res3 = multiply_matrices(v.children[2], U1V1)
             + multiply_matrices(v.children[3], U2V1)
        res4 = multiply_matrices(v.children[2], U1V2)
             + multiply_matrices(v.children[3], U2V2)

    # return result
    res = np.zeros((res1.shape[0]+res3.shape[0],
                    res1.shape[1]+res2.shape[1]))
    half_r = res1.shape[0]
    half_c = res1.shape[1]
    res[:half_r, :half_c] = res1
    res[:half_r, half_c:] = res2
    res[half_r:, :half_c] = res3
    res[half_r:, half_c:] = res4
    return res
```

Listing 1: Mnożenie macierzy

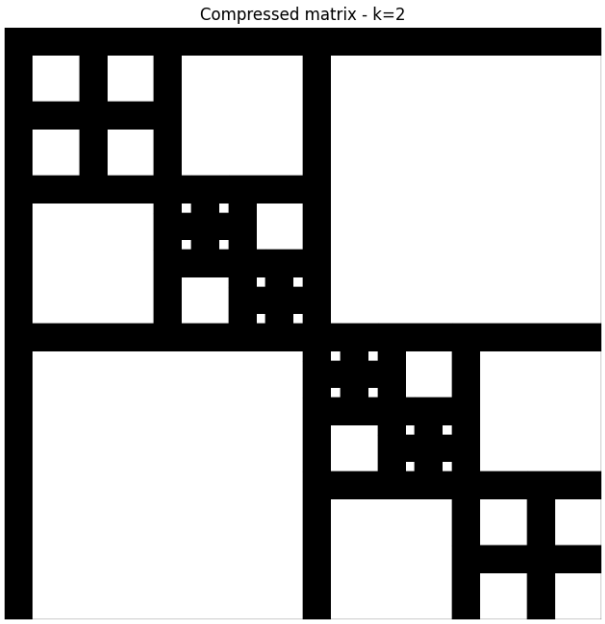
```
# Multiply compressed matrix by vector
def multiply_by_vector(v, vec):
    # leaf
    if len(v.children) == 0:
        if v.rank == 0:
            return np.zeros(vec.shape[0])
        return v.U @ (v.V @ vec)

    # not leaf
    half = vec.shape[0] // 2
    vec1 = vec[:half]
    vec2 = vec[half:]
    y11 = multiply_by_vector(v.children[0], vec1)
    y12 = multiply_by_vector(v.children[1], vec2)
    y21 = multiply_by_vector(v.children[2], vec1)
    y22 = multiply_by_vector(v.children[3], vec2)
    result = np.zeros(vec.shape[0])
    result[:half] = y11 + y12
    result[half:] = y21 + y22
    return result
```

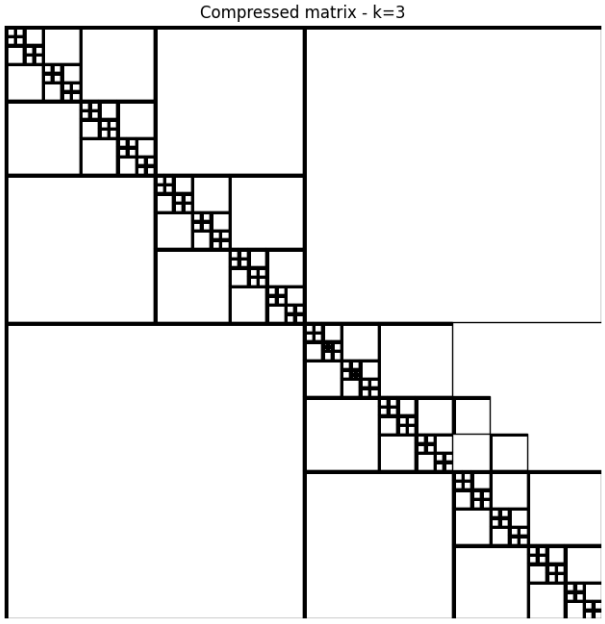
Listing 2: Mnożenie macierzy przez wektor

3 Rysunki przedstawiające skompresowane macierze

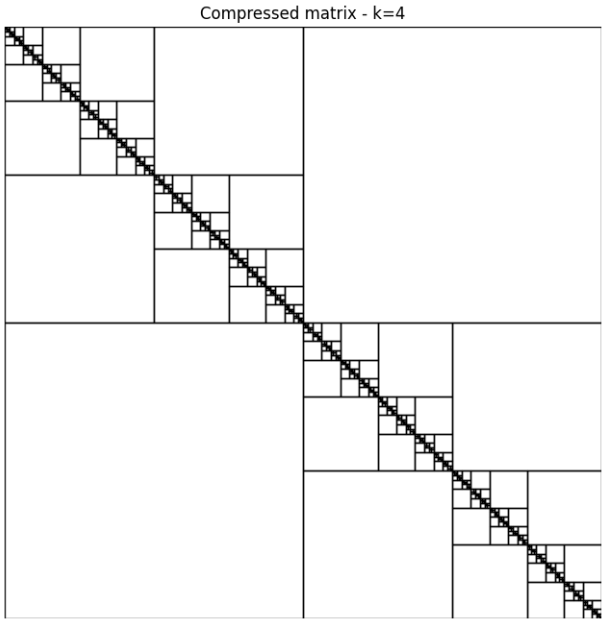
Poniżej przedstawiono rysunki ilustrujące skompresowane macierze.



Rysunek 1: Skompresowana macierz dla $k = 2$



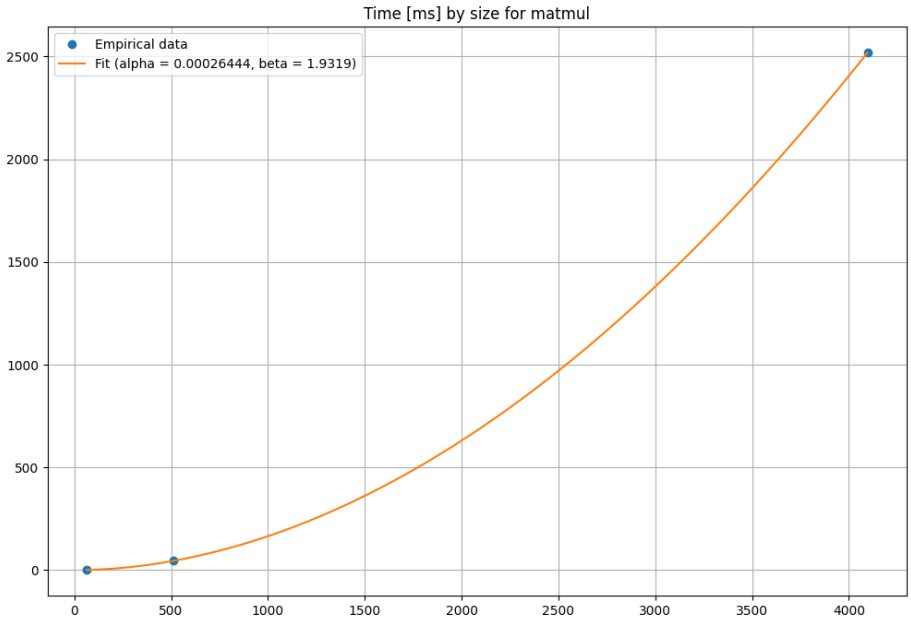
Rysunek 2: Skompresowana macierz dla $k = 3$



Rysunek 3: Skompresowana macierz dla $k = 4$

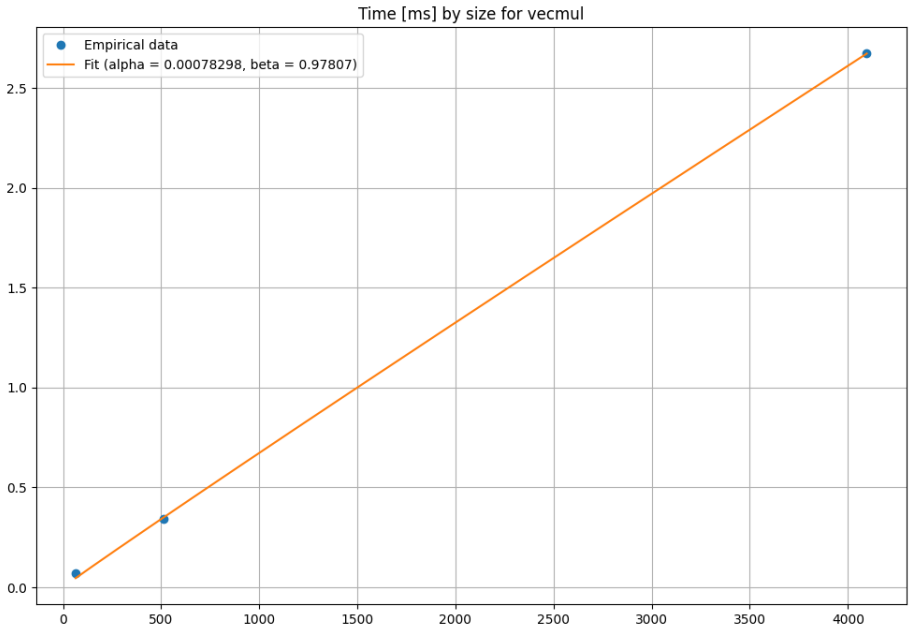
4 Czas mnożenia oraz dopasowanie

Poniżej przedstawiono wykresy przedstawiające czasy mnożenia wraz z dopasowaniem funkcji αn^{β}.



Rysunek 4: Czas mnożenia macierzy i dopasowanie

Złożoność eksperymentalna wynosi około: $2.7 \cdot 10^{-4} \cdot n^{1.93} = O(n^{1.93})$.



Rysunek 5: Czas mnożenia macierzy przez wektor i dopasowanie

Złożoność eksperymentalna wynosi około: $7.8 \cdot 10^{-4} \cdot n^{0.98} = O(n^{0.98})$.

5 Norma Frobeniusa^2

Poniżej przedstawiono tabelę porównującą wyniki używając normy Frobeniusa^2.

	Mnożenie macierzy	Mnożenie macierzy przez wektor
$k = 2$	74.666	6.2352
$k = 3$	1803.6	288.7
$k = 4$	20769	3924.9

Tabela 1: Norma Frobeniusa^2 dla danych wyników mnożenia