Algorytmy macierzowe - laboratorium 4

Marta Bukowińska
Jakub Karczewski

15.01.2024

1 Treść zadania

1. Wybrać ulubiony język programowania.
2. Wygenerować macierz o rozmiarze $2^{3k} = 2^k \cdot 2^k \cdot 2^k$ dla $k = 2, 3, 4$, która opisuje strukturę topologii trójwymiarowej siatki zbudowanej z elementów sześciennych.
   - Wiersz w macierzy odpowiada wierzchołkowi siatki.
   - Niezerowe losowe wartości w kolumnach oznaczają sąsiadujące wierzchołki siatki.
3. Zaimplementować rekurencyjną procedurę kompresji macierzy z Zadania 3.
4. Narysować skompresowaną macierz, wykorzystując rysowacz z Zadania 3.
5. Wykonać mnożenie skompresowanej macierzy przez wektor (20 punktów).
6. Wykonać mnożenie skompresowanej macierzy przez samą siebie (20 punktów).

2 Fragmenty kodu

2.1 Mnożenie macierzy przez wektor

```
def matrix_vector_mult(v, X):
    if not v.sons:
        if v.rank == 0:
            return np.zeros(len(X))
        return np.matmul(v.U, np.matmul(v.V, X))
    rows = len(X)
    X1 = X[: rows // 2]
    X2 = X[rows // 2 :]
    Y11 = matrix_vector_mult(v.sons[0], X1)
    Y12 = matrix_vector_mult(v.sons[1], X2)
```

```
    Y21 = matrix_vector_mult(v.sons[2], X1)
    Y22 = matrix_vector_mult(v.sons[3], X2)
    return np.hstack((np.add(Y11, Y12), np.add(Y21, Y22)))
```

2.2 Mnożenie macierzy przez macierz

```
def rSVDofCompressed(v, w):
    U = np.hstack((v.U, w.U))
    V = np.vstack((v.V, w.V))
    M = np.matmul(U, V)
    return CreateTree(M, 0, M.shape[0]-1, 0, M.shape[1]-1, r, eps)

def create_subnode(v, U_slice, V_slice):
    u = Node()
    u.rank = v.rank
    u.U = v.U[U_slice, :]
    u.V = v.V[:, V_slice]
    u.sons = []
    return u

def matrix_matrix_add(v, w):
    if not v.sons and not w.sons:
        if v.rank == 0 and w.rank == 0:
            res = Node(0)
            return res
        if v.rank != 0 and w.rank != 0:
            return rSVDofCompressed(v, w)
    if v.rank != 0 and w.rank == 0:
        return v
    if v.rank == 0 and w.rank != 0:
        return w
    if v.sons and w.sons:
        res = Node()
        res.sons.append(matrix_matrix_add(v.sons[0], w.sons[0]))
        res.sons.append(matrix_matrix_add(v.sons[1], w.sons[1]))
        res.sons.append(matrix_matrix_add(v.sons[2], w.sons[2]))
        res.sons.append(matrix_matrix_add(v.sons[3], w.sons[3]))
        return res
    if not v.sons and w.sons:
        half = len(v.U) // 2
        u11 = create_subnode(v, slice(0, half), slice(0, half))
        u12 = create_subnode(v, slice(0, half), slice(half, None))
        u21 = create_subnode(v, slice(half, None), slice(0, half))
        u22 = create_subnode(v, slice(half, None), slice(half, None
            ))
        res = Node()
        res.sons.append(matrix_matrix_add(u11, w.sons[0]))
        res.sons.append(matrix_matrix_add(u12, w.sons[1]))
        res.sons.append(matrix_matrix_add(u21, w.sons[2]))
        res.sons.append(matrix_matrix_add(u22, w.sons[3]))
        return res
    if v.sons and not w.sons:
        half = len(w.U) // 2
        u11 = create_subnode(w, slice(0, half), slice(0, half))
        u12 = create_subnode(w, slice(0, half), slice(half, None))
```

```
        u21 = create_subnode(w, slice(half, None), slice(0, half))
        u22 = create_subnode(w, slice(half, None), slice(half, None
            ))
        res = Node()
        res.sons.append(matrix_matrix_add(v.sons[0], u11))
        res.sons.append(matrix_matrix_add(v.sons[1], u12))
        res.sons.append(matrix_matrix_add(v.sons[2], u21))
        res.sons.append(matrix_matrix_add(v.sons[3], u22))
        return res
```

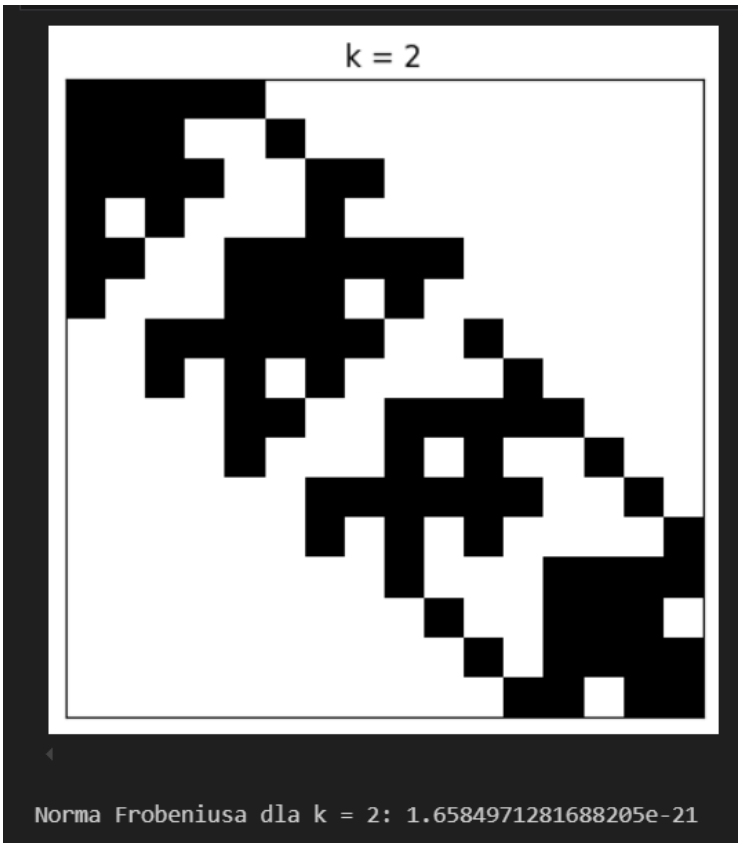
```
def matrix_matrix_mult(v, w):
    if not v.sons and not w.sons:
        if v.rank == 0 and w.rank == 0:
            res = Node(0)
            return res
        if v.rank != 0 and w.rank != 0:
            res = Node(v.rank)
            res.U = v.U * (v.V * w.U)
            res.V = w.V
            return res

    if v.rank != 0 and w.rank == 0:
        return w

    if v.rank == 0 and w.rank != 0:
        return v

    if v.sons and w.sons:
        res = Node()
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[0], w.sons[0]),
                matrix_matrix_mult(v.sons[1], w.sons[2])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[0], w.sons[1]),
                matrix_matrix_mult(v.sons[1], w.sons[3])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[2], w.sons[0]),
                matrix_matrix_mult(v.sons[3], w.sons[2])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[2], w.sons[1]),
                matrix_matrix_mult(v.sons[3], w.sons[3])
            )
```

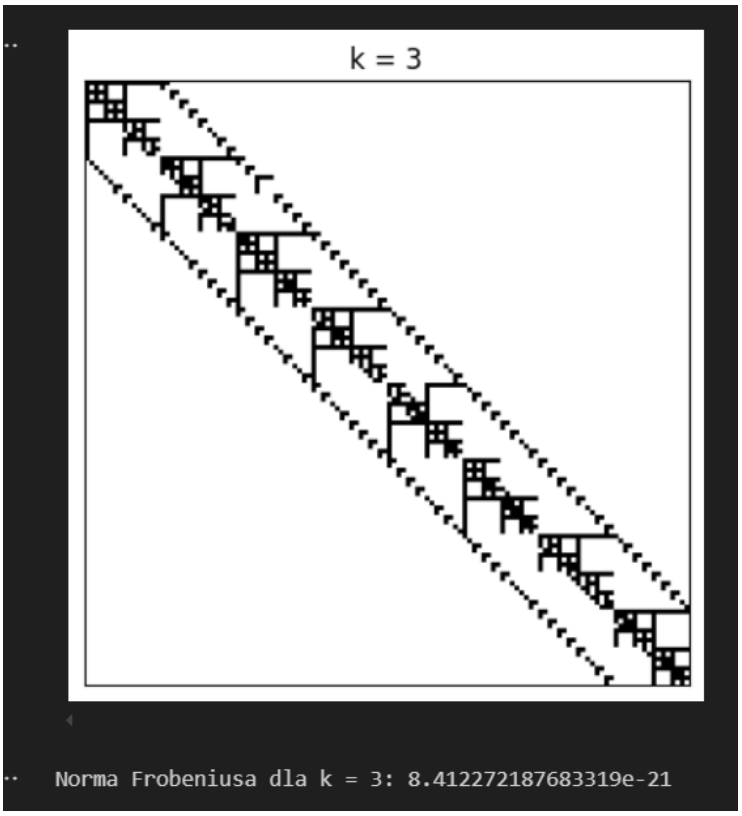
```
        )
        return res

    if not v.sons and w.sons:
        half = len(v.U) // 2
        u11 = create_subnode(v, slice(0, half), slice(0, half))
        u12 = create_subnode(v, slice(0, half), slice(half, None))
        u21 = create_subnode(v, slice(half, None), slice(0, half))
        u22 = create_subnode(v, slice(half, None), slice(half, None
            ))
        res = Node()
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(u11, w.sons[0]),
                matrix_matrix_mult(u12, w.sons[2])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(u11, w.sons[1]),
                matrix_matrix_mult(u12, w.sons[3])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(u21, w.sons[0]),
                matrix_matrix_mult(u22, w.sons[2])
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(u21, w.sons[1]),
                matrix_matrix_mult(u22, w.sons[3])
            )
        )
        return res

    if v.sons and not w.sons:
        half = len(w.U) // 2
        u11 = create_subnode(w, slice(0, half), slice(0, half))
        u12 = create_subnode(w, slice(0, half), slice(half, None))
        u21 = create_subnode(w, slice(half, None), slice(0, half))
        u22 = create_subnode(w, slice(half, None), slice(half, None
            ))
        res = Node()
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[0], u11),
                matrix_matrix_mult(v.sons[1], u21)
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[0], u12),
                matrix_matrix_mult(v.sons[1], u22)
            )
        )
```

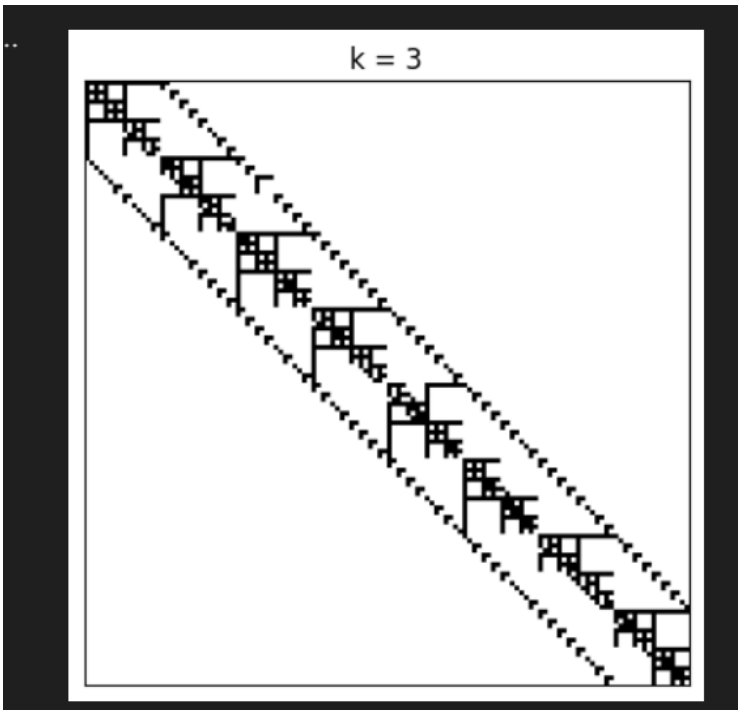
```
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[2], u11),
                matrix_matrix_mult(v.sons[3], u21)
            )
        )
        res.sons.append(
            matrix_matrix_add(
                matrix_matrix_mult(v.sons[2], u12),
                matrix_matrix_mult(v.sons[3], u22)
            )
        )
        return res
```

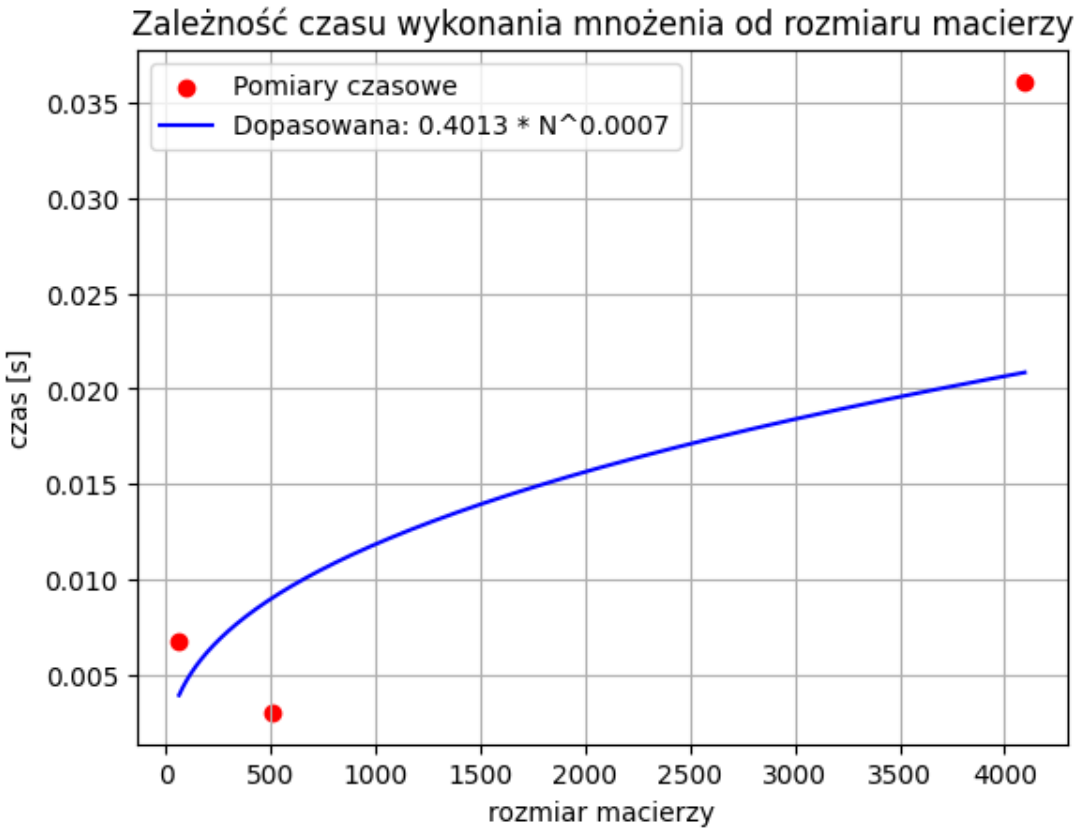
3 Wykresy

3.1 Mnożenie macierzy przez wektor

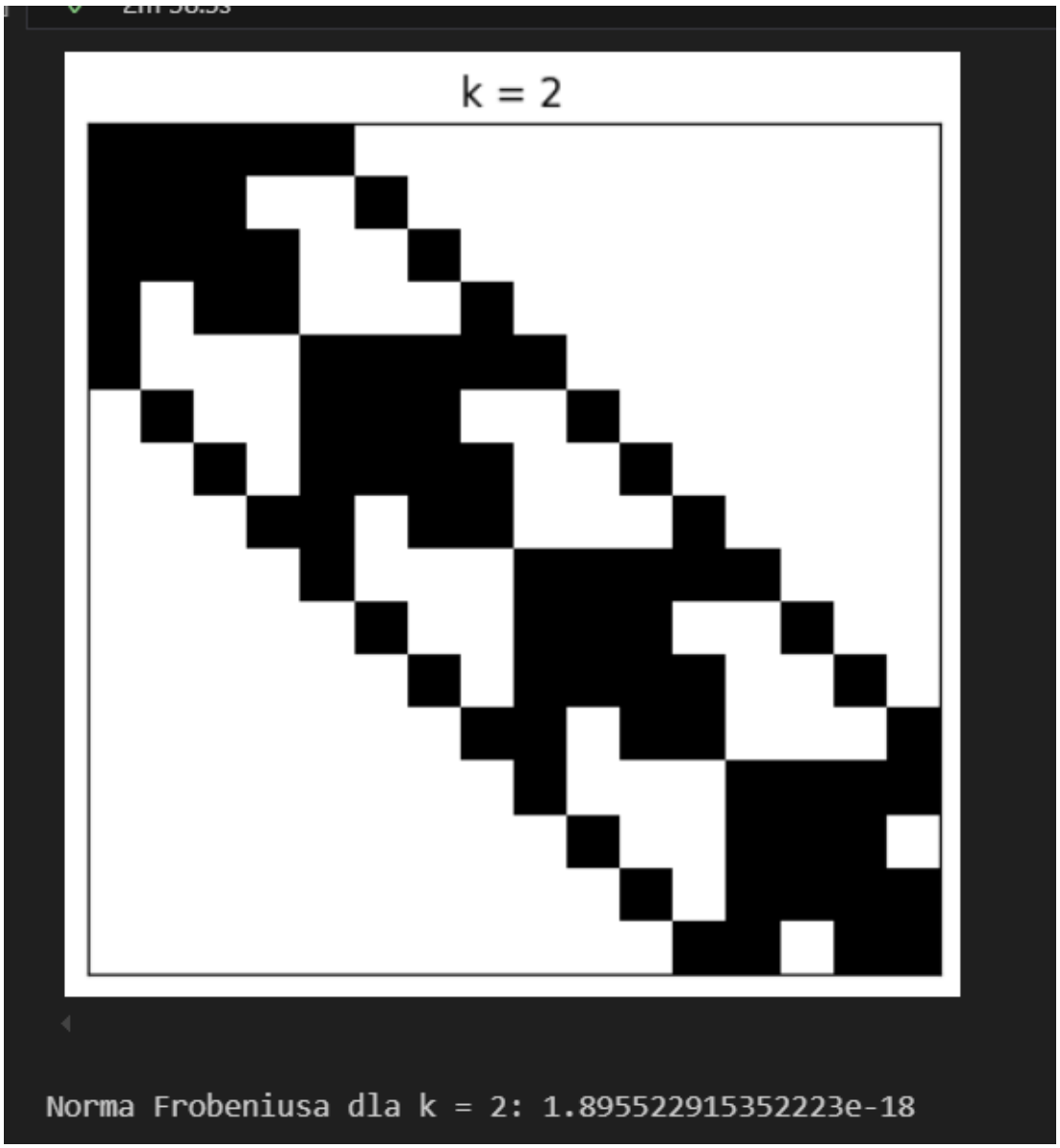


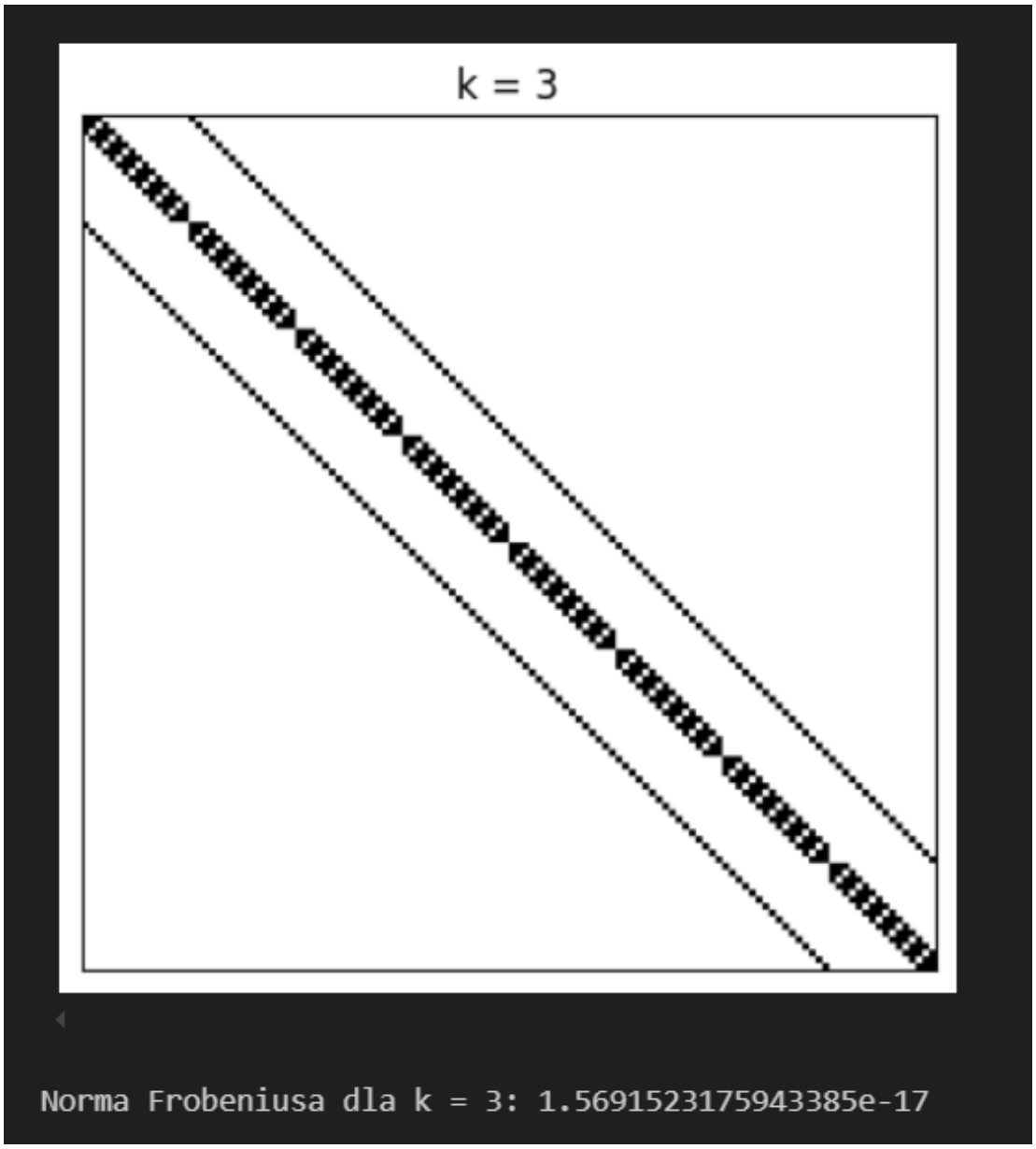

k = 3
Norma Frobeniusa dla k = 3: 8.412272187683319e-21

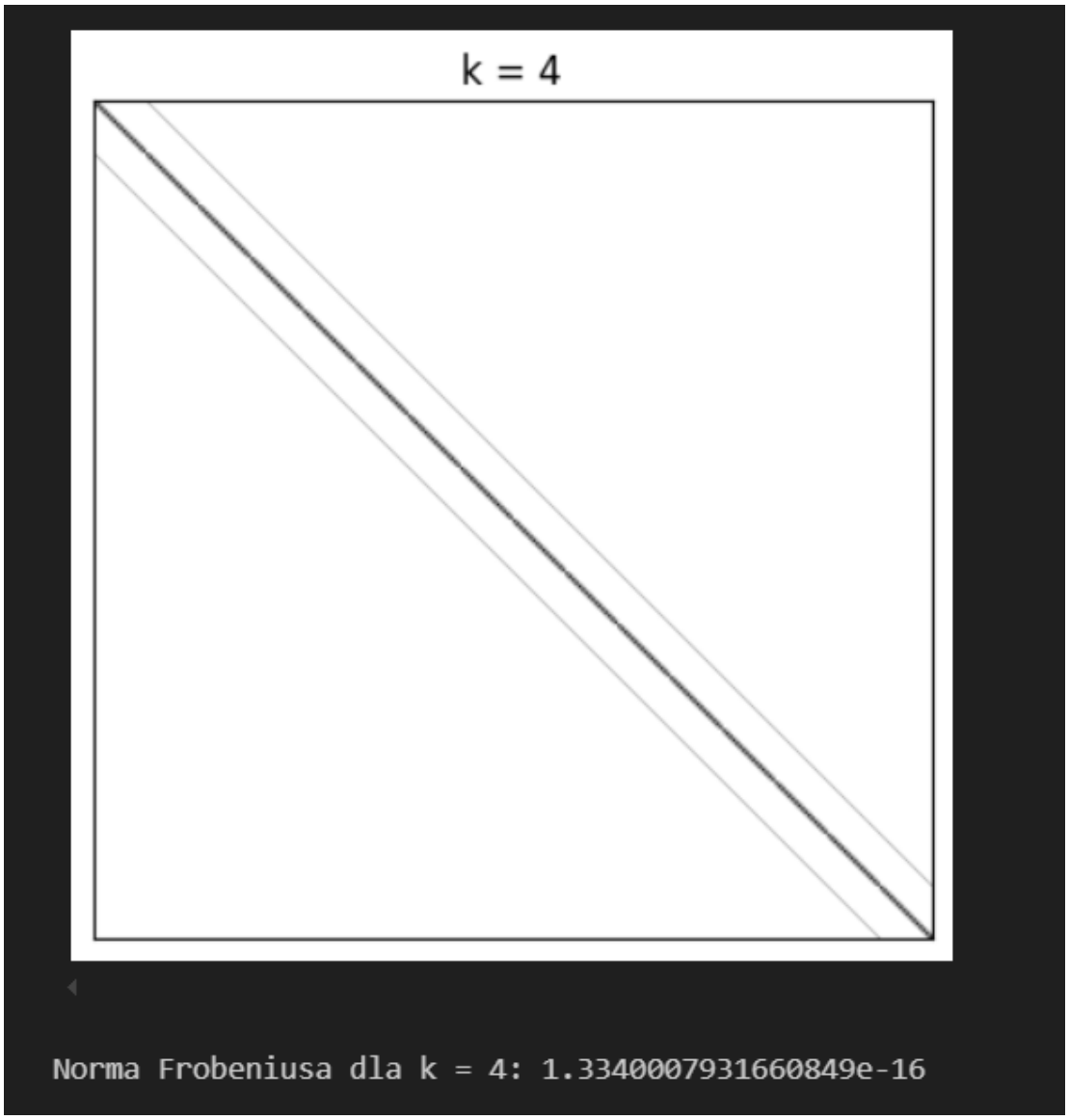

k = 3
Norma Frobeniusa dla k = 3: 8.412272187683319e-21

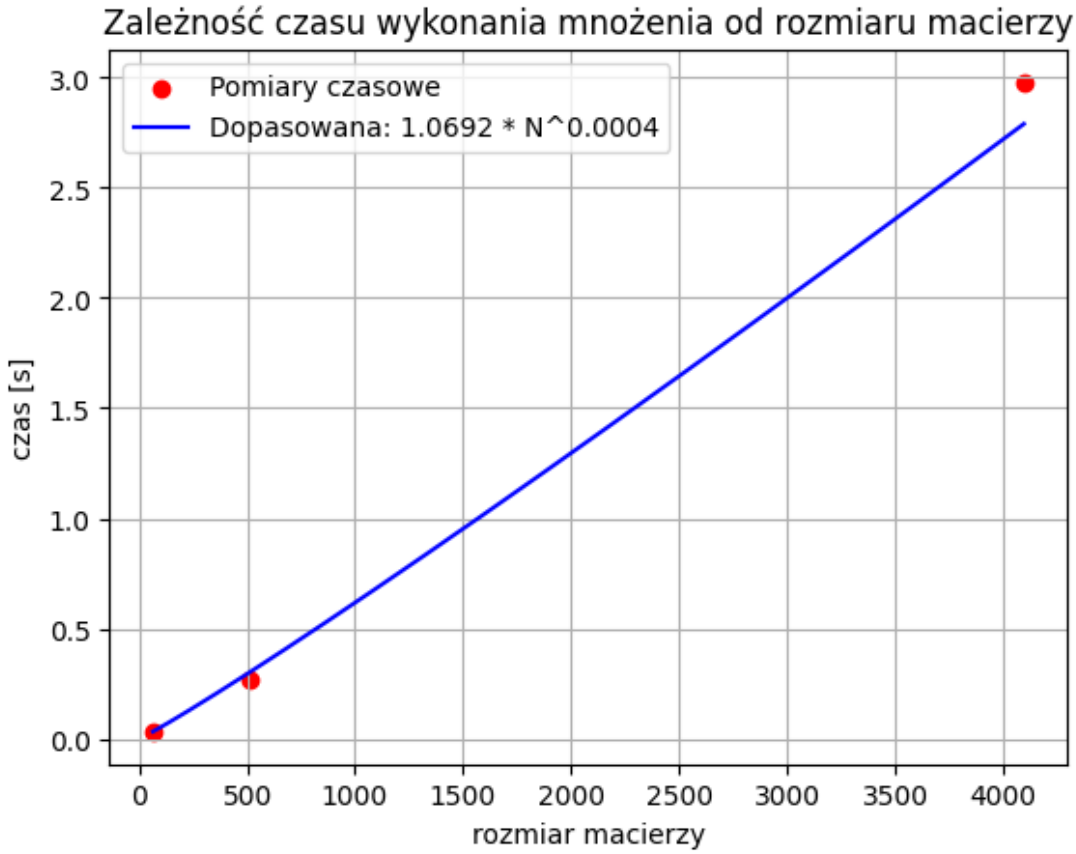

Zależność czasu wykonania mnożenia od rozmiaru macierzy
Pomiary czasowe
Dopasowana: 0.4013 * N^0.0007
czas [s]
rozmiar macierzy
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
500
1000
1500
2000
2500
3000
3500
4000

3.2 Mnożenie macierzy przez macierz




k = 3
Norma Frobeniusa dla k = 3: 1.5691523175943385e-17


k = 4
Norma Frobeniusa dla k = 4: 1.3340007931660849e-16


Zależność czasu wykonania mnożenia od rozmiaru macierzy
Pomiary czasowe
Dopasowana: 1.0692 * N^0.0004
czas [s]
rozmiar macierzy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
500
1000
1500
2000
2500
3000
3500
4000

4 Wnioski