Ćwiczenie 3

Algorytmy macierzowe

Autorzy:
Gabriel Kaźmierczak
Dariusz Piwowarski

Spis treści

1 Kod programu (i pseudokod jednocześnie, bo jest to Python)

1.1 Główne metody do kompresji i dekompresji

```
def compress_matrix(M, r, epsilon):
    # Jeśli macierz składa się z samych zer to kompresujemy ją zapisując
    # jej wymiary oraz rząd równy 0.
    if not np.any(M):
        return Node(*M.shape, 0)

    # Jeśli rozmiar macierzy jest mniejszy od r, to nie ma sensu jej
    # kompresować, bo i tak zajmie to więcej pamięci niż bez kompresji.
    if min(M.shape[0], M.shape[1]) <= r + 1:
        node = Node(*M.shape, None)
        node.matrix = M
        return node

    # Wyznaczmy truncated svd, dokładnie dla r + 1 wartości osobliwych,
    # aby sprawdzić czy najmniej znacząca z nich, czyli ostatnia,
    # jest mniejsza od epsilon.
    U, Sigma, VT = svds(M, k=r + 1)

    # Jeśli wartość ta jest rzeczywiście mniejsza od epsilon to zapisujemy
    # macierz w skompresowanej formie czyli U, Sigma i VT.
    # W przeciwnym wypadku dzielimy macierz na 4 podmacierze i wywołujemy
    # metodę kompresji rekurencyjnie.
    if abs(Sigma[0]) < epsilon:
        node = Node(*M.shape, r)
        node.U = U[:, 1:]
        node.Sigma = Sigma[1:]
        node.VT = VT[1:]
    else:
        M11, M12, M21, M22 = split_matrix(M)
        node = Node(*M.shape, None)
        node.children = [
            compress_matrix(M11, r, epsilon),
            compress_matrix(M12, r, epsilon),
            compress_matrix(M21, r, epsilon),
            compress_matrix(M22, r, epsilon)
        ]

    return node
```

Fragment 1: compress_matrix

```python
def decompress(node):
    if node.rank is not None:
        if node.rank > 0:
            # jeśli rząd macierzy większy od zera to odtwarzamy macierz z
            # postaci skompresowanej mnożąc U * Sigma @ VT
            return node.U * node.Sigma @ node.VT
        else:
            # jeśli rząd macierzy równy zero to odtwarzamy macierz jako
            # macierz samych zer
            return np.zeros((node.n, node.m))
    elif node.matrix is not None:
        # przypadek gdy podmacierz nie została skompresowana
        # (bo nie było to opłacalne)
        return node.matrix
    else:
        # przypadek gdy podmacierz została podzielona i skompresowana
        # reurencyjnie, wywołujemy zatem rekurencyjnie dekompresję i
        # sklejamy macierze
        return np.vstack(
            (
                np.hstack((decompress(node.children[0]),
                decompress(node.children[1]))),
                np.hstack((decompress(node.children[2]),
                decompress(node.children[3]))),
            )
        )
```

Fragment 2: decompress_matrix

1.2 Klasa Node reprezentująca wierzchołek drzewa skompresowanej macierzy

Klasa ta zawiera również metodę `draw_matrix()`, która implementuje "rysowacz"

```python
class Node:
    def __init__(self, n, m, rank):
        self.n = n
        self.m = m
        self.rank = rank
        self.U = None
        self.Sigma = None
        self.VT = None
        self.matrix = None
        self.children = []

    def __get_matrix_to_draw(self):
        if self.rank is not None:
            if self.rank > 0:
                m = np.zeros((self.n, self.m))
                m[:, :self.rank] = 1
                m[:self.rank, :] = 1
                return m
            else:
                return np.zeros((self.n, self.m))
        elif self.matrix is not None:
            return np.ones((self.n, self.m))
        else:
            return np.vstack(
                (
                    np.hstack((self.children[0].__get_matrix_to_draw(),
                    self.children[1].__get_matrix_to_draw())),
                    np.hstack((self.children[2].__get_matrix_to_draw(),
                    self.children[3].__get_matrix_to_draw())),
                )
            )

    def draw_matrix(self):
        fig, ax = plt.subplots(figsize=(16, 9))
        ax.matshow(
          self.__get_matrix_to_draw(),
```

Fragment 3: Node

2 Testy implementacji

2.1 Test działania algorytmu

W celu zbadania poprawności działania implementacji wykonujemy kolejno kompresję i dekompresję macierzy `256x256`, gdzie 10 procent to wartości niezerowe. Następnie rysujemy macierz "rysowaczem", żeby zobaczyć efekt kompresji, a na koniec porównujemy zdekompresowaną macierz z macierzą wejściową stosując metrykę:

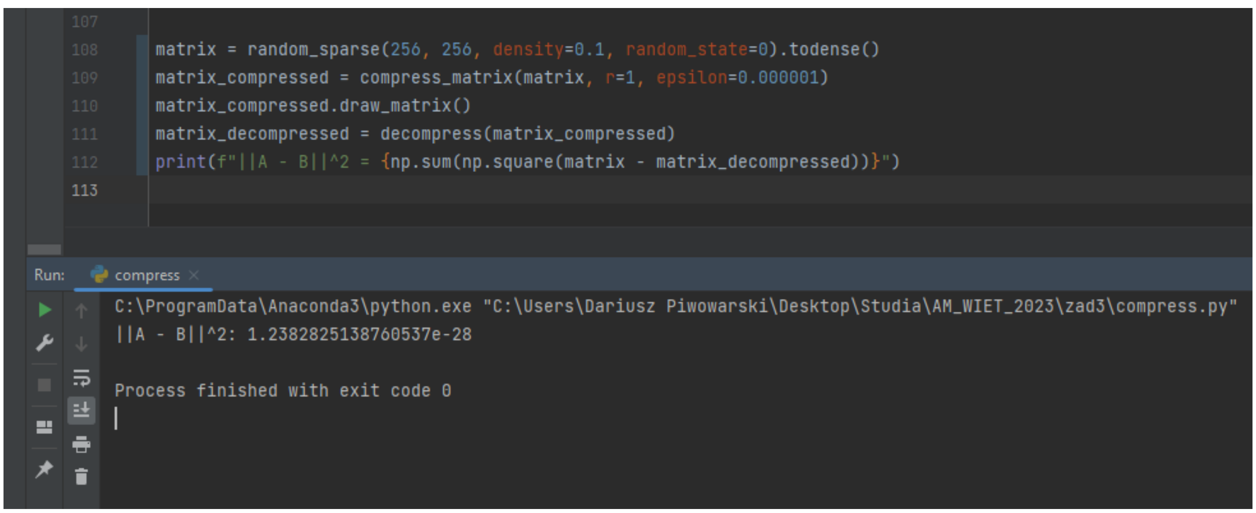
$$||A - B||^2$$

Stosujemy przy tym następujące wartości r i epsilon:

$$r = 1$$

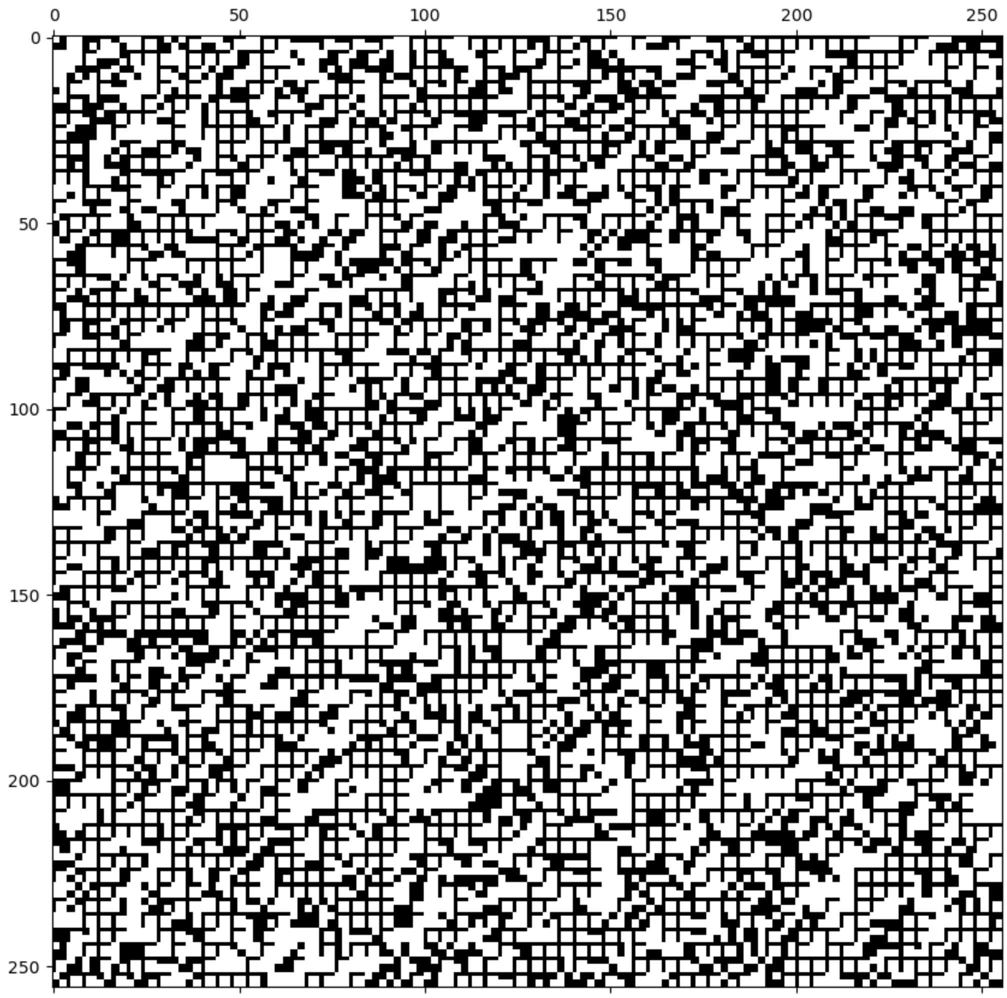
$$\epsilon = 0.000001$$

Wyniki:



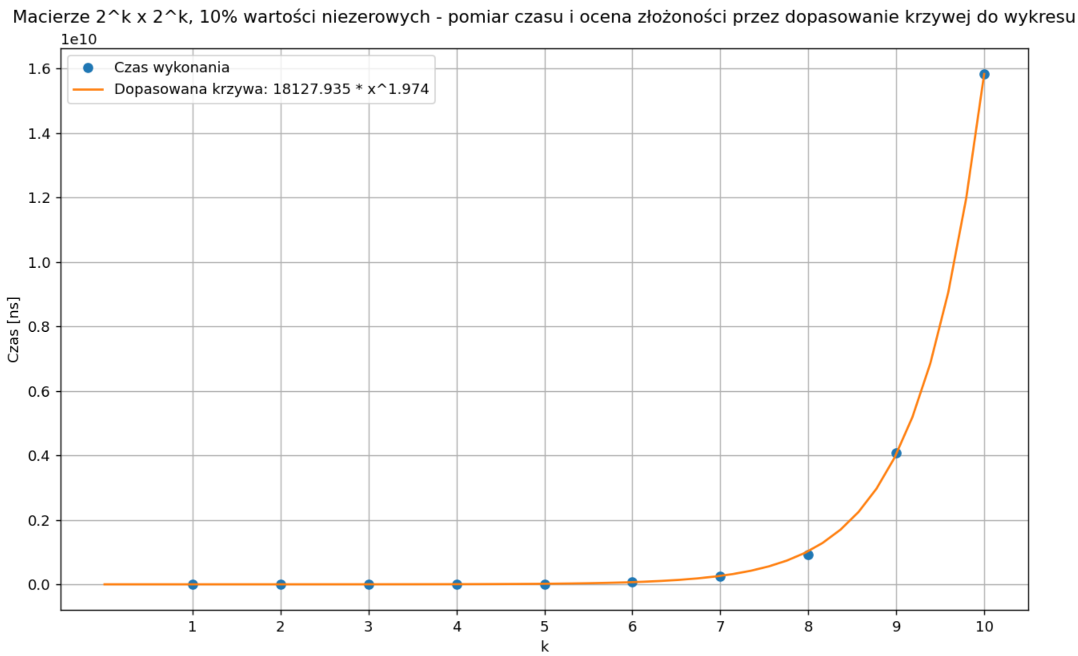
Rysunek 1: $||A - B||^2$

Wartość metryki $||A - B||^2$ jest praktycznie równa równa zeru (uzyskana wartość jest rzędu 1^{-28}). Co pokazuje, że przy tak wybranym epsilonie kompresja jest praktycznie bezstratna.

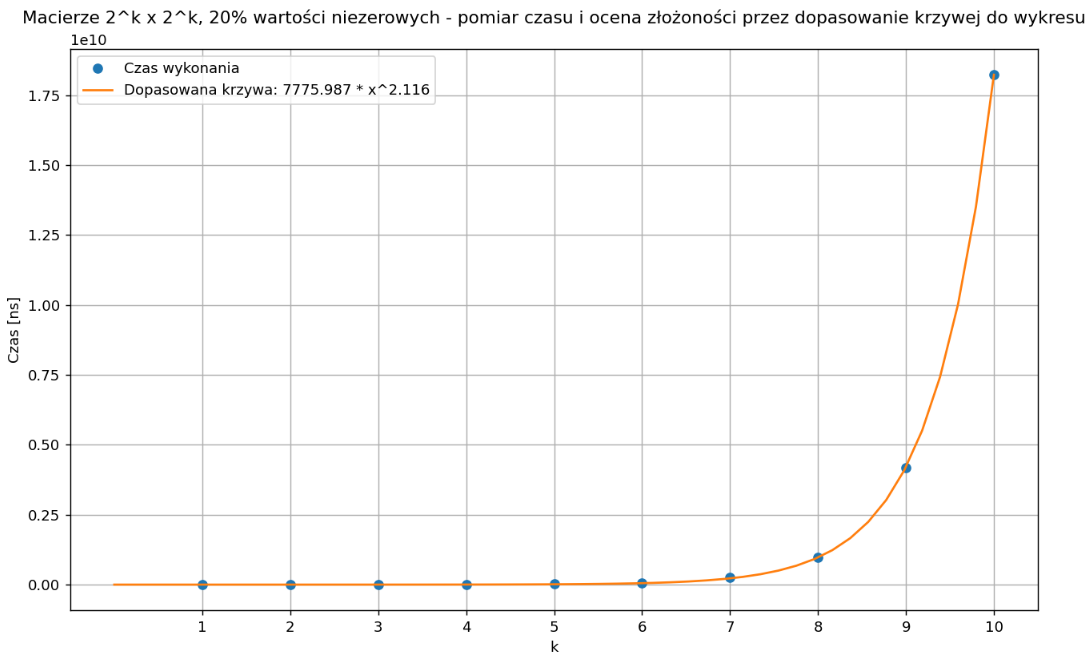


Rysunek 2: Tak natomiast wygląda macierz po kompresji.

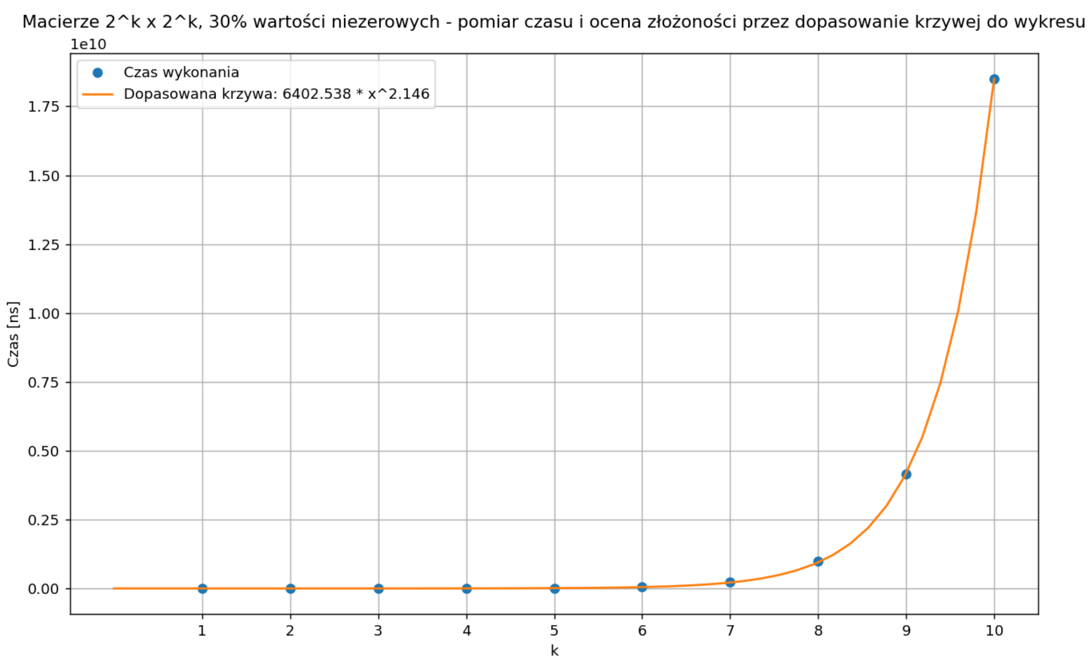
2.2 Pomiar czasu wykonania



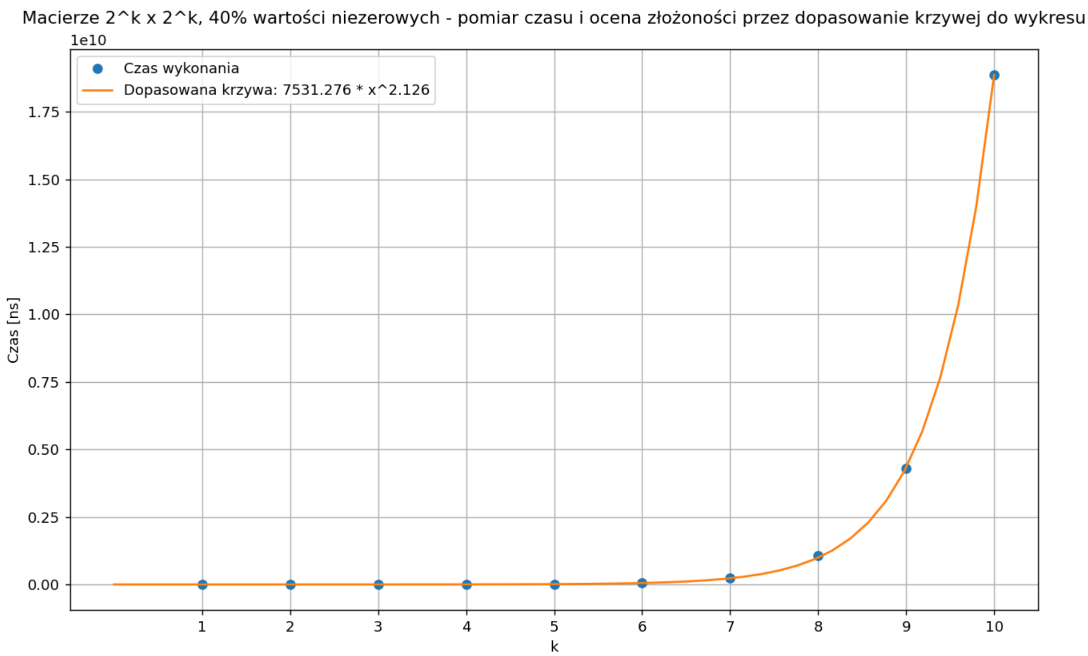
Rysunek 3: 10% wartości niezerowych



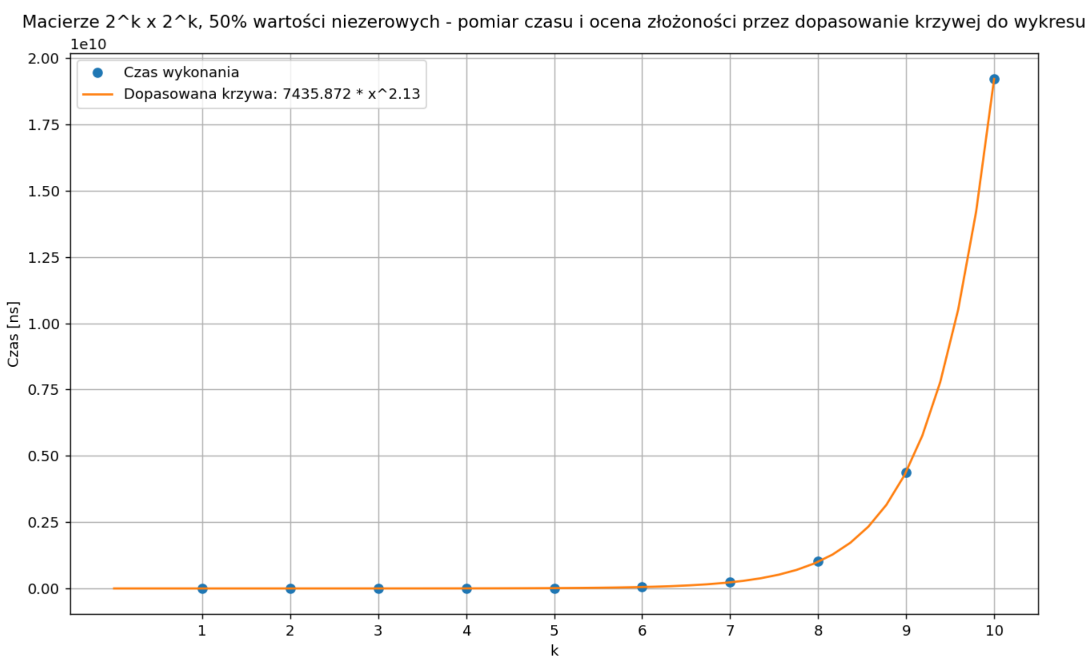
Rysunek 4: 20% wartości niezerowych



Rysunek 5: 30% wartości niezerowych



Rysunek 6: 40% wartości niezerowych



Rysunek 7: 50% wartości niezerowych

2.3 Szacowana złożoność obliczeniowa

Na podstawie zmierzonych czasów i dopasowanych krzywych, złożoność obliczeniową możemy oszacować jako:

$$O(N^2)$$

dla macierzy N x N. Czyli złożoność jest tak naprawdę rzędu rozmiaru macierzy.