Algorytmy Macierzowe
Sprawozdanie 3
Rekurencyjne kompresja macierzy

Przemek Węglik
Szymon Paszkiewicz

6 grudnia 2022

Spis treści

1 Opis algorytmu **2**

2 Fragmenty kodu **2**

3 Benchmarki **3**
3.1 Czasy dla różnego stopnia wypłnienia macierzy 3
3.2 Rysunki 5
3.3 Błąd dekompresji 8

1 Opis algorytmu

Traktujemy macierz jak drzewo czwórkowe. Każda ćwiartka macierzy odpowiada jednemu węzłowi-dziecku. Podczas każdego podziału podejmujemy decyzję czy będziemy to dziecko dzielić dalej czy raczej kompresować.

Kompresja opłaca nam się wtedy jeśli rząd macierzy jest niewielki. Wtedy używając algorytmu $TruncatedSVD$ możemy zmniejszyć złożoność pamięciową z $O(n^2)$ do praktycznie $O(n)$, jeśli rząd macierzy jest dostatecznie mały.

Potem rekurencyjnie wykonujemy tę procedurę dla każdego dziecka.

2 Fragmenty kodu

Funkcja kompresująca:

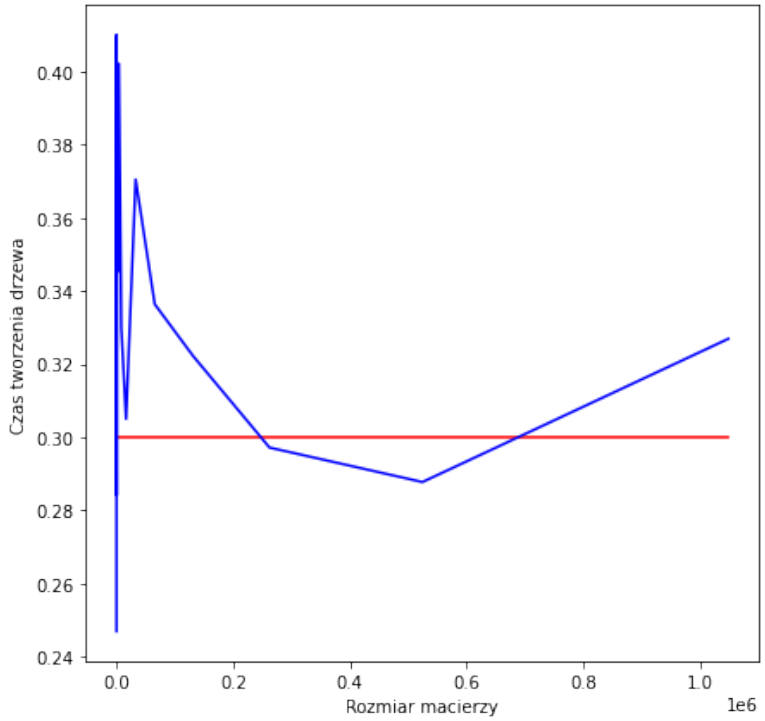
```
def compress_matrix(A: np.ndarray, first_row: int, last_row: int,
first_col: int, last_col: int, r: int) -> TreeLeaf:
    v = TreeLeaf()
    v.size = (first_row, last_row, first_col, last_col)
    U, D, V = truncatedSVD(A, r + 1)
    if consist_of_zeros(A):
        v.rank = 0
    else:
        v.rank = r
        v.singular_values = D[0 : r]
        v.U = U[:, 0 : r]
        v.V =  V[0 : r, :]
    return v
```

Funkcja budująca drzewo

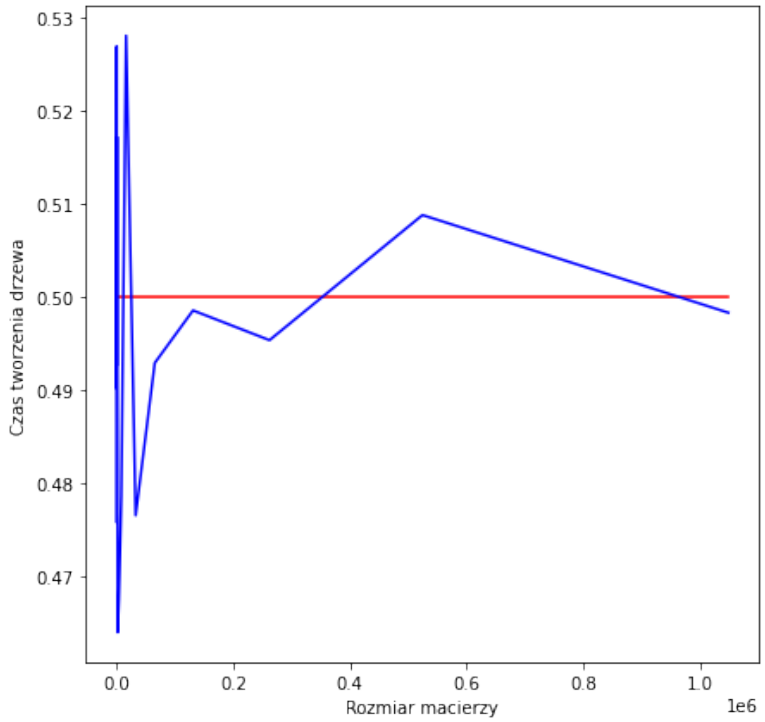
```
def create_tree(A: np.ndarray, first_row: int, last_row: int,
first_col: int, last_col: int, r: int, eps: float) -> TreeNode:
    new_A = A[first_row : last_row, first_col : last_col]
    U, D, V = truncatedSVD(new_A, r + 1)
    if r + 1 > D.shape[0] or D[r] < eps:
        if D.shape[0] <= 2:
            v = compress_matrix(new_A, first_row, last_row,
            first_col, last_col, 1)
        else:
            v = compress_matrix(new_A, first_row, last_row,
            first_col, last_col, r)
    else:
        v = TreeSplit()
        middle_row = (first_row + last_row) // 2
        middle_col = (first_col + last_col) // 2
        v.left_upper = create_tree(A, first_row, middle_row,
        first_col, middle_col, r, eps)
        v.right_upper = create_tree(A, first_row, middle_row,
        middle_col, last_col, r, eps)
        v.left_lower = create_tree(A, middle_row, last_row,
        first_col, middle_col, r, eps)
        v.right_lower = create_tree(A, middle_row, last_row,
        middle_col, last_col, r, eps)
    return v
```

3 Benchmarki

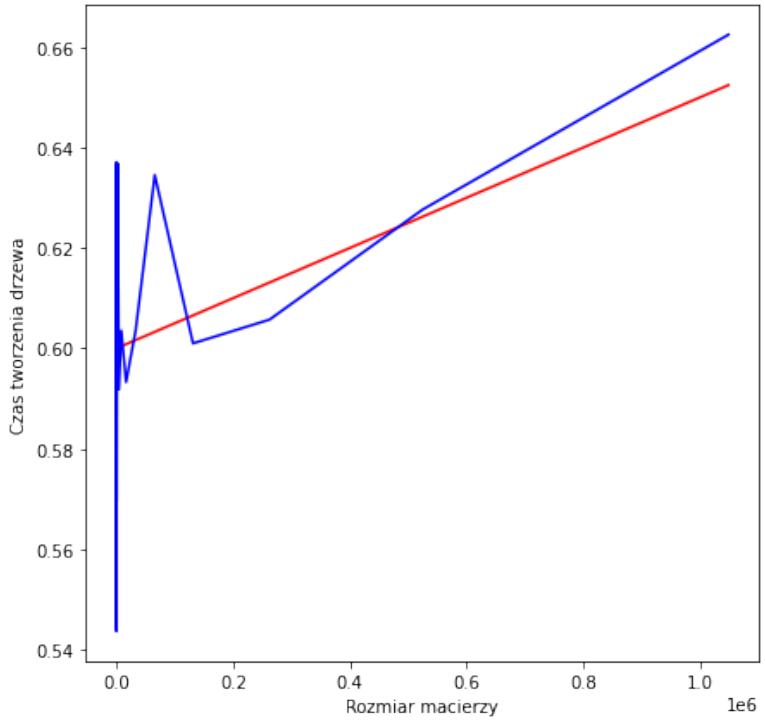
3.1 Czasy dla różnego stopnia wypłnienia macierzy



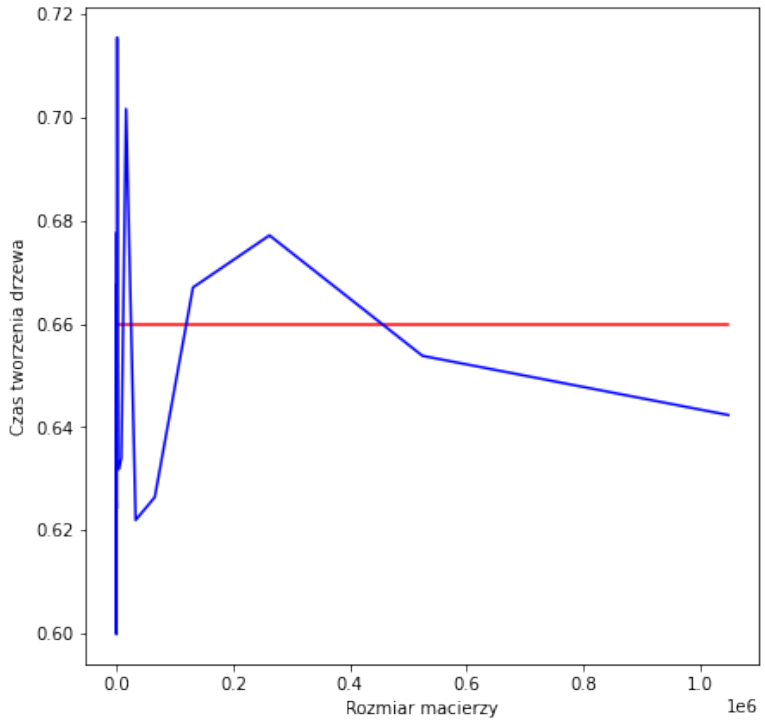
Rysunek 1: Wykres czasu kompresji od rozmiaru macierzy dla wypełnienia macierzy 10% z dopasowaną prostą y = 0.3.



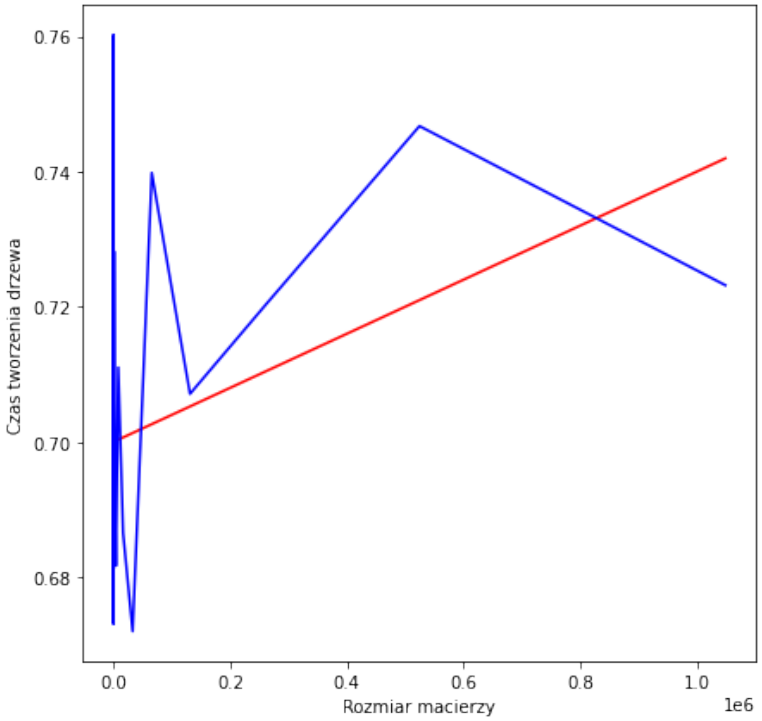
Rysunek 2: Wykres czasu kompresji od rozmiaru macierzy dla wypełnienia macierzy 20% z dopasowaną prostą y = 0.5.



Rysunek 3: Wykres czasu kompresji od rozmiaru macierzy dla wypełnienia macierzy 30% z dopasowaną prostą $y = 0.5 * 10^{-7} * x + 0.6$.

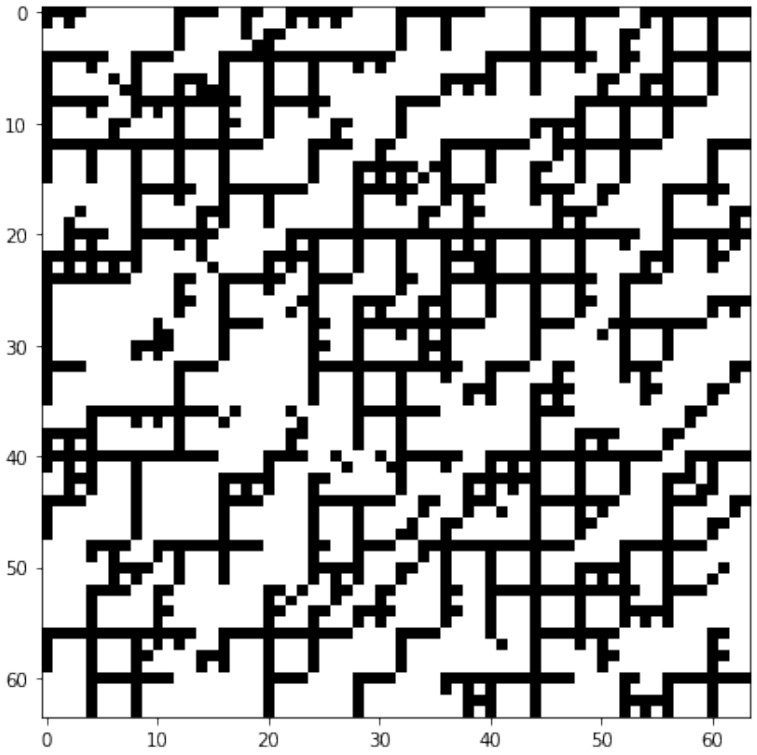


Rysunek 4: Wykres czasu kompresji od rozmiaru macierzy dla wypełnienia macierzy 40% z dopasowaną prostą $y = 0.66$.

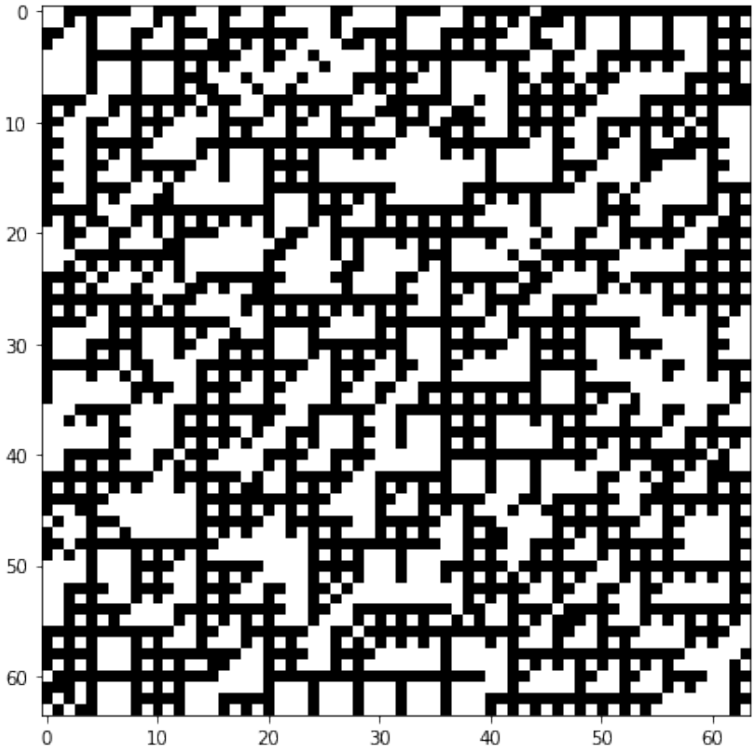


Rysunek 5: Wykres czasu kompresji od rozmiaru macierzy dla wypełnienia macierzy 50% z dopasowaną prostą $y = 0.4 * 10^{-7} * x + 0.7$.

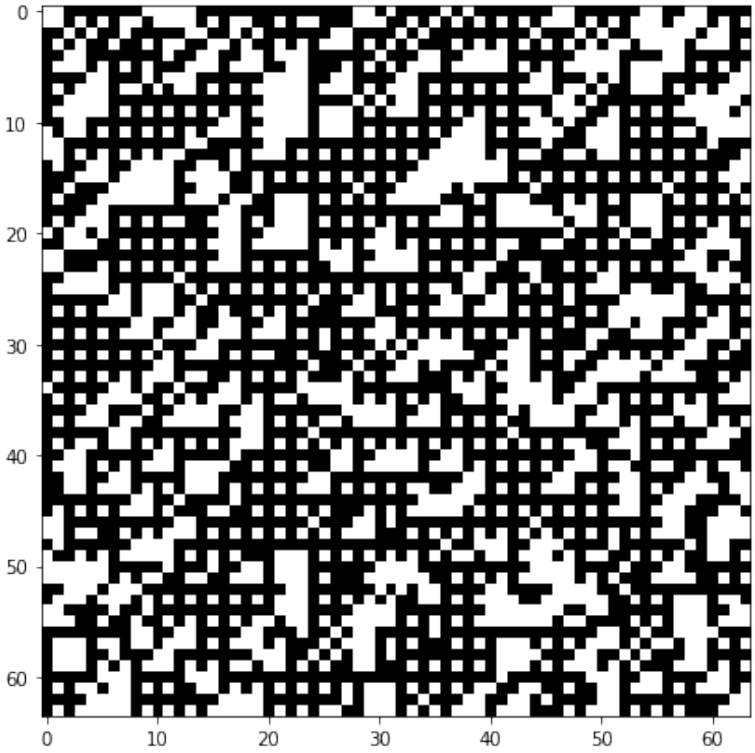
3.2 Rysunki



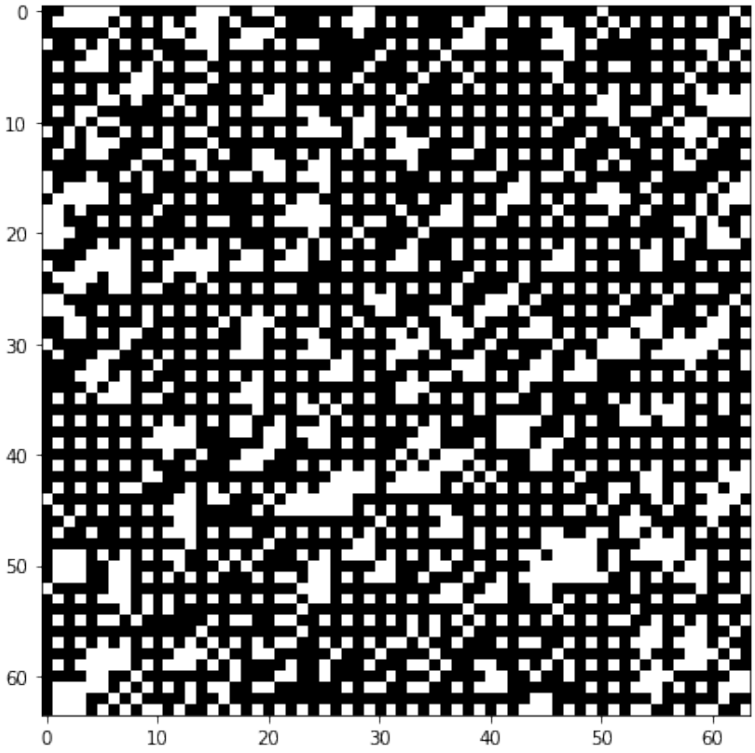
Rysunek 6: Wizualizacja podziału macierzy dla wypełnienia 10%.



Rysunek 7: Wizualizacja podziału macierzy dla wypełnienia 20%.



Rysunek 8: Wizualizacja podziału macierzy dla wypełnienia 30%.



Rysunek 9: Wizualizacja podziału macierzy dla wypełnienia 40%.

Rysunek 10: Wizualizacja podziału macierzy dla wypełnienia 50%.

3.3 Błąd dekompresji

Wypełnienie	0.1	0.2	0.3	0.4	0.5
Błąd	bliski 0	bliski 0	0.008	0.016	0.02

Tabela 1: Błąd średniokwadratowy kompresji dla macierzy $128x128$