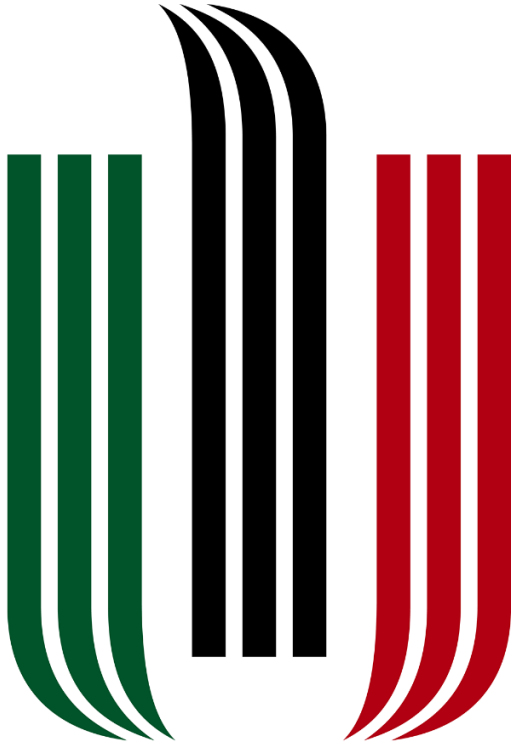


AGH

Algorytmy Macierzowe

Sprawozdanie z laboratorium nr.3

Władysław Jerzy Nieć, Paweł Surdyka

Zadania

- Proszę napisać rekurencyjną kompresje macierzy z wykorzystaniem częściowego SVD (20 punktów) dla wybranych parametrów δ=najmniejsza wartość osobliwa (wyrzucamy mniejsze) i b= maksymalny rank (liczba wartości osobliwych)
- Proszę zaimplementować rysowacz (10 punktów)

Rekurencyjna kompresja macierzy

Klasa MatrixTree reprezentuje strukturę drzewa dla macierzy i udostępnia metody do manipulacji oraz wizualizacji tej struktury.

```python
class MatrixTree:
    def __init__(self, matrix: np.ndarray):
        # Inicjalizacja
        self.matrix = matrix
        self.shape = matrix.shape
        self.zeros = None
        self.A11 = None
        self.A12 = None
        self.A21 = None
        self.A22 = None
        self.u = None
        self.vT = None

    def compute(self):
        if self.matrix is not None:
            return self.matrix
        elif self.zeros is not None:
            return np.zeros(self.zeros)
        elif self.A11 is not None:
            a11, a12, a21, a22 = (i.compute() for i in (self.A11, self.A12, self.A21, self.A22))
            U = np.hstack((a11, a12))
            L = np.hstack((a21, a22))
            return np.vstack((U, L))
        else:
            return self.u @ self.vT

    def paint(self):
        # Generacja macierzy obrazu
        if self.matrix is not None:
            image = np.zeros(self.matrix.shape)
        elif self.zeros is not None:
            image = np.ones(self.zeros)
        elif self.A11 is not None:
            a11, a12, a21, a22 = (i.paint() for i in (self.A11, self.A12, self.A21, self.A22))
            U = np.hstack((a11, a12))
            L = np.hstack((a21, a22))
            image = np.vstack((U, L))
        else:
            shape = (self.u.shape[0], self.vT.shape[1])
            image = np.ones(shape)
            image[:, :self.u.shape[1]] = np.zeros(self.u.shape)
            image[:self.vT.shape[0], :] = np.zeros(self.vT.shape)
        return image

    def draw(self):
        # Rysuje obraz na podstawie macierzy obrazu przy użyciu biblioteki Matplotlib.
        plt.gray()
        plt.imshow(self.paint())
        plt.show()
```

Funkcja do dekompresji

```
def decompose(tree: MatrixTree, u: np.ndarray, d: np.ndarray, vT: np.ndarray) -> None:
    if len(d)==0:
        tree.zeros=tree.matrix.shape
        tree.matrix = None
        return
    tree.matrix = None
    d = np.diag(d) if len(d)>1 else np.array([d])
    tree.u = u @ d
    tree.vT = vT
```

Głowna funkcja służąca do kompresji

```
def compress(tree: MatrixTree, sigma: float, b: int):
    # Sprawdź, czy wszystkie elementy macierzy są bliskie zeru; jeśli tak, ustaw macierz na zera
    if not np.any(abs(tree.matrix) > 1e-4):
        tree.zeros = tree.matrix.shape
        tree.matrix = None
        return

    # Sprawdź, czy rozmiar macierzy jest mniejszy lub równy 1; jeśli tak, zakończ bez kompresji
    if tree.matrix.size <= 1:
        return

    # Wykonaj dekompozycję SVD na macierzy
    u, s, vT = np.linalg.svd(tree.matrix)

    # Określ nową rangę na podstawie progu wartości osobliwych 'sigma'
    new_rank = np.sum(np.where(s >= sigma, 1, 0))

    # Jeśli nowa ranga jest mniejsza lub równa określonemu progu rangi 'b', skompresuj macierz
    if new_rank <= b:
        u = u[:, :new_rank]
        vT = vT[:new_rank, :]
        decompose(tree, u, s[:new_rank], vT)
        return
    else:
        # Podziel macierz na podmacierze
        A11, A12, A21, A22 = divide(tree.matrix)
        tree.matrix = None

        # Utwórz obiekty MatrixTree dla podmacierzy
        tree.A11 = MatrixTree(A11)
        tree.A12 = MatrixTree(A12)
        tree.A21 = MatrixTree(A21)
        tree.A22 = MatrixTree(A22)

        # Rekurencyjnie kompresuj podmacierze
        compress(tree.A11, sigma, b)
        compress(tree.A12, sigma, b)
        compress(tree.A21, sigma, b)
        compress(tree.A22, sigma, b)
```

Testy

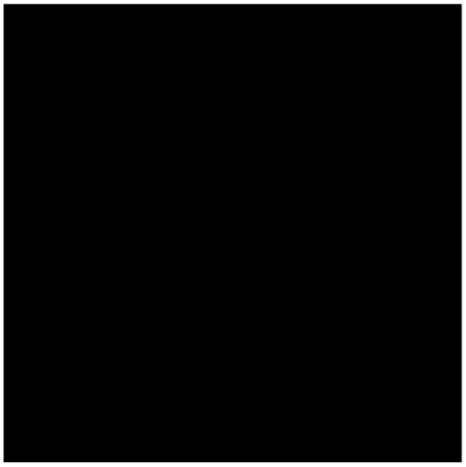
Ilość zer(%)	sigma	b	błąd średniokwadratowy	czas(s)
99	σ0	1	118.23	5.22
99	σ0	4	117.94	7.19
99	σ1024	1	117.99	19.07
99	σ1024	4	114.94	12.58
99	σ2047	1	0.0	20.01
99	σ2047	4	0.0	15.08
98	σ0	1	167.23	5.97
98	σ0	4	165.80	5.46
98	σ1024	1	165.51	10.48
98	σ1024	4	164.14	10.08
98	σ2047	1	0.0	25.66
98	σ2047	4	0.0	17.44
95	σ0	1	264.52	5.38
95	σ0	4	264.54	5.32
95	σ1024	1	258.56	9.98
95	σ1024	4	258.51	9.87
95	σ2047	1	0.0	42.93
95	σ2047	4	0.0	23.57
90	σ0	1	359.26	5.36
90	σ0	4	359.98	6.01
90	σ1024	1	358.04	10.11
90	σ1024	4	358.07	10.66
90	σ2047	1	0.04	67.36
90	σ2047	4	0.0	39.13
80	σ0	1	487.11	5.57
80	σ0	4	528.59	9.31
80	σ1024	1	485.39	13.46
80	σ1024	4	485.48	15.01
80	σ2047	1	0.03	100.79
80	σ2047	4	0.28	55.85

Pomiary dla algorytmu dekonstrukcji

Wykresy H-macierzy

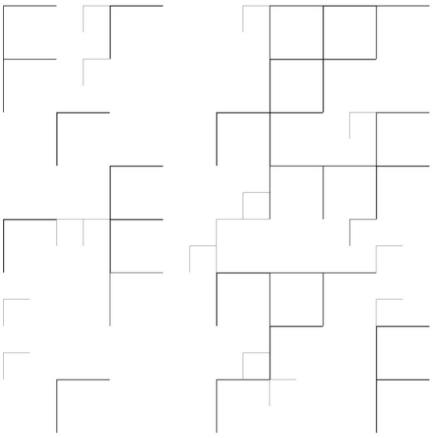
Wykresy dla macierzy wypełnionej w 99% zerami:

δ = σ0, b = 1

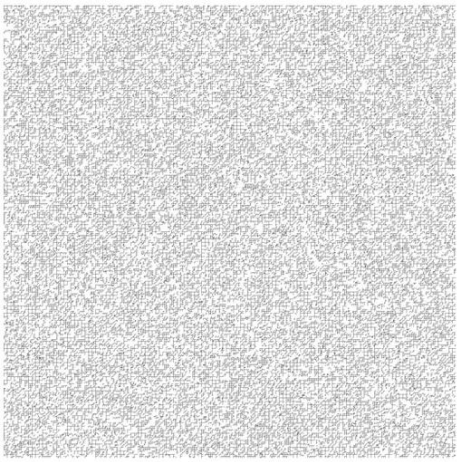


δ = σ0, b = 4

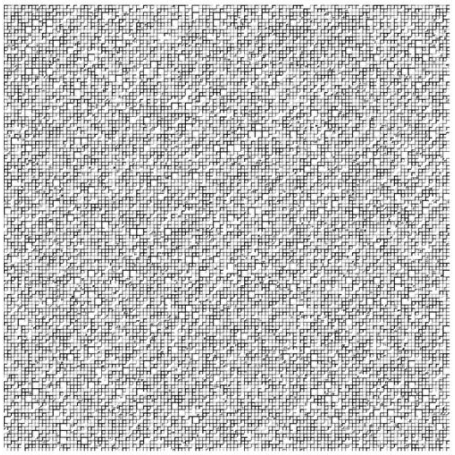
δ = σ1024, b = 1



δ = σ1024, b = 4

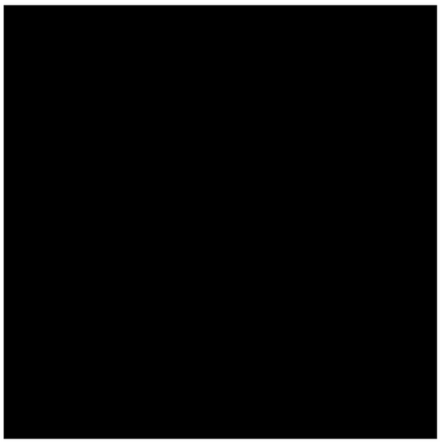


δ = σ2047, b = 1



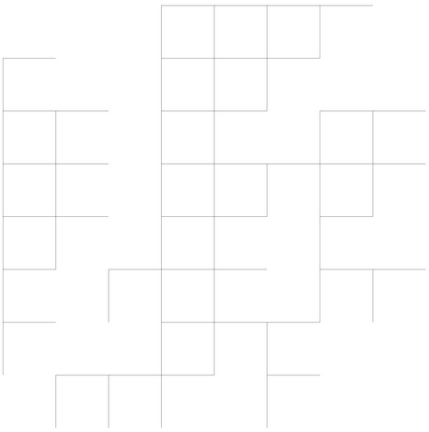
δ = σ2047, b = 4

Wykresy dla macierzy wypełnionej w 98% zerami:

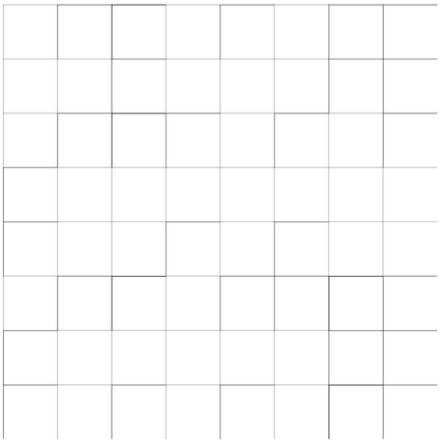


δ = σ0, b = 1

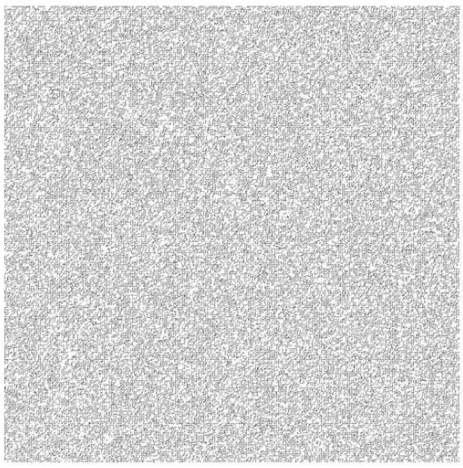
δ = σ0, b = 4



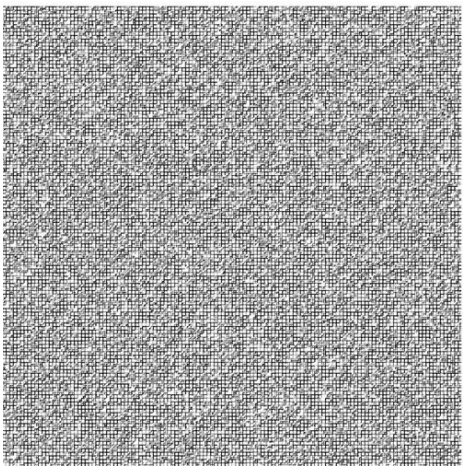
δ = σ1024, b = 1



δ = σ1024, b = 4

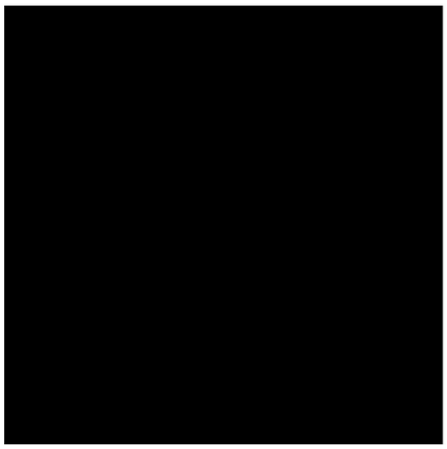


δ = σ2047, b = 1



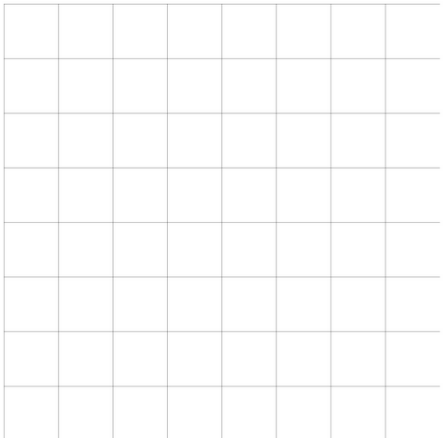
δ = σ2047, b = 4

Wykresy dla macierzy wypełnionej w 95% zerami:

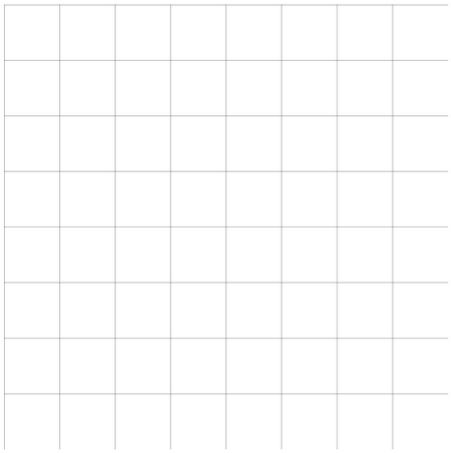


δ = σ0, b = 1

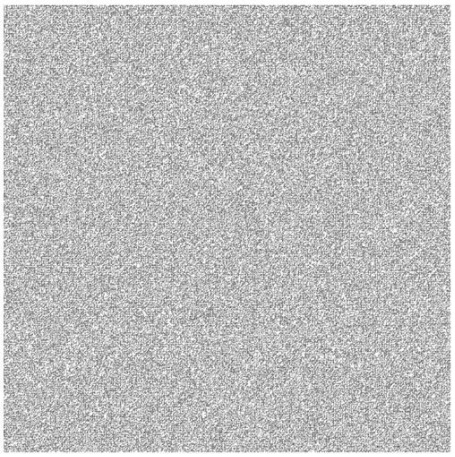
δ = σ0, b = 4



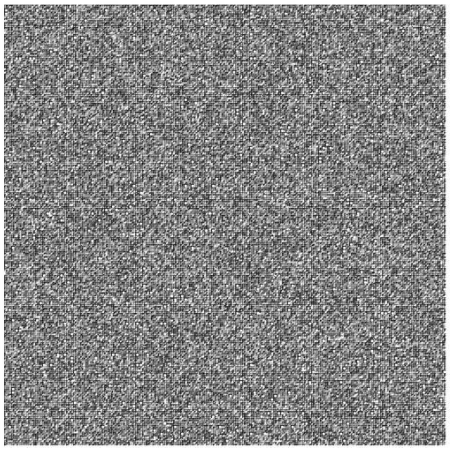
δ = σ1024, b = 1



δ = σ1024, b = 4



δ = σ2047, b = 1



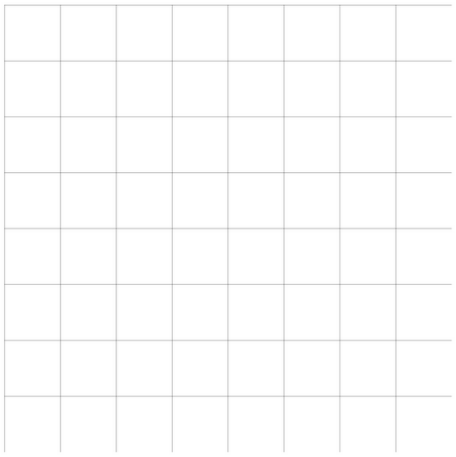
δ = σ2047, b = 4

Wykresy dla macierzy wypełnionej w 90% zerami:

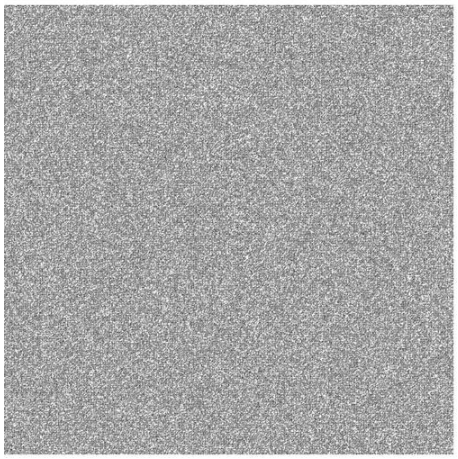
δ = σ0, b = 1

δ = σ0, b = 4

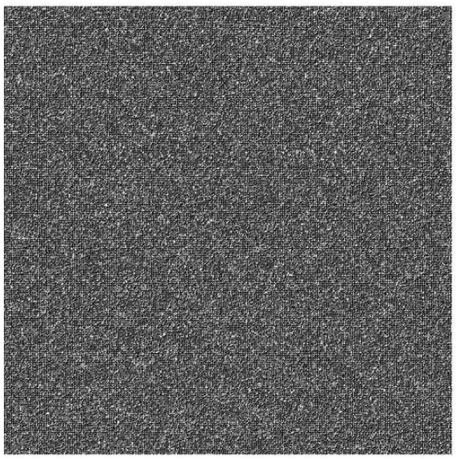
δ = σ1024, b = 1



δ = σ1024, b = 4



δ = σ2047, b = 1



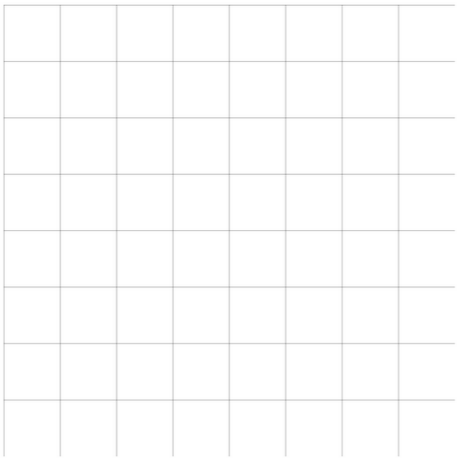
δ = σ2047, b = 4

Wykresy dla macierzy wypełnionej w 80% zerami:

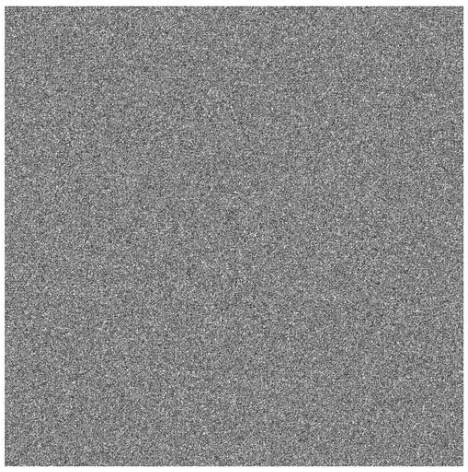
δ = σ0, b = 1

δ = σ0, b = 4

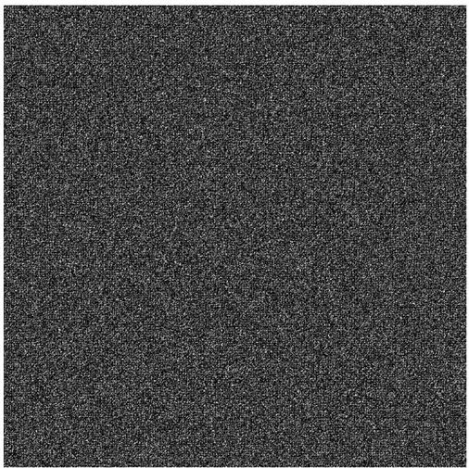
δ = σ1024, b = 1



δ = σ1024, b = 4



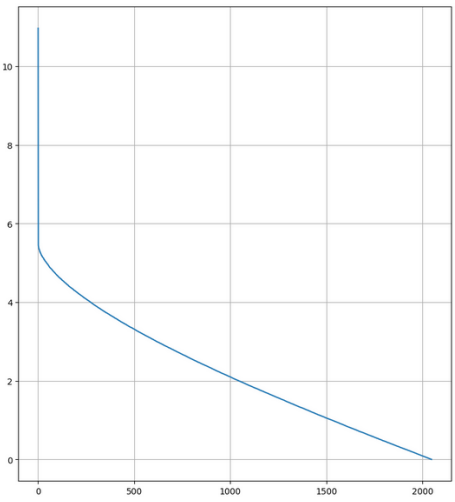
δ = σ2047, b = 1



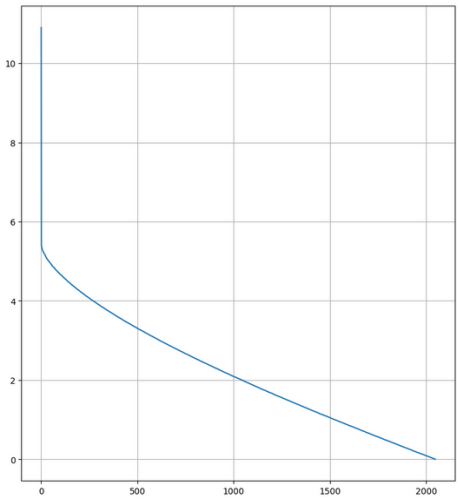
δ = σ2047, b = 4

Wykresy wartości osobliwych

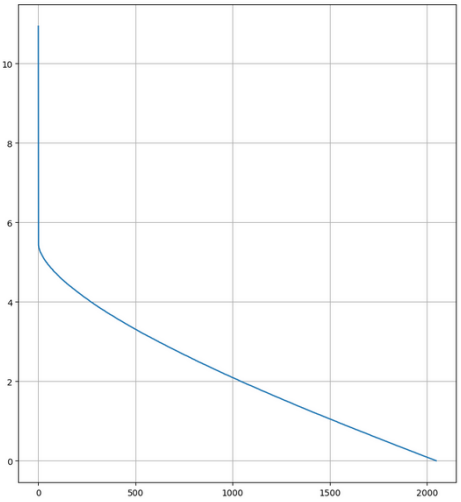
Wykresy dla macierzy wypełnionej w 99% zerami:



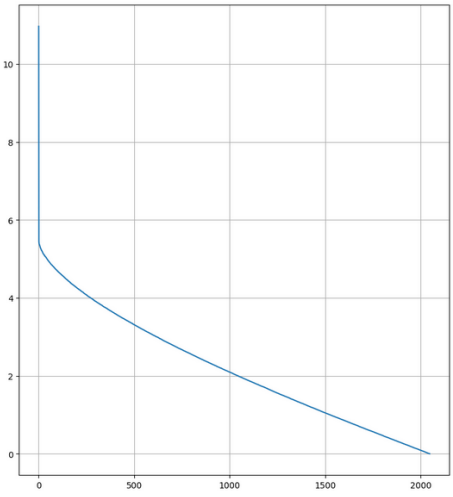
δ = σ0, b = 1



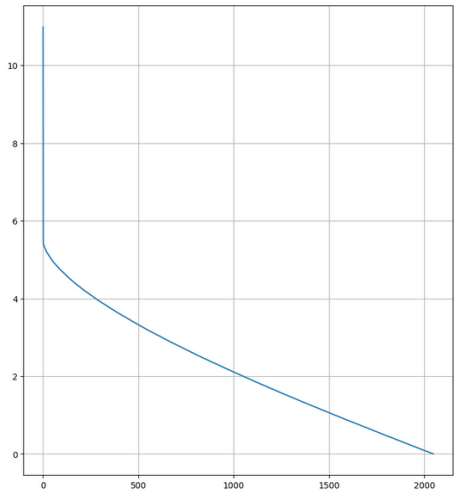
δ = σ0, b = 4



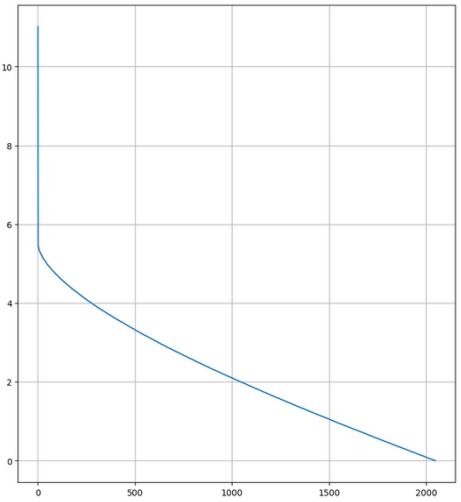
δ = σ1024, b = 1



δ = σ1024, b = 4

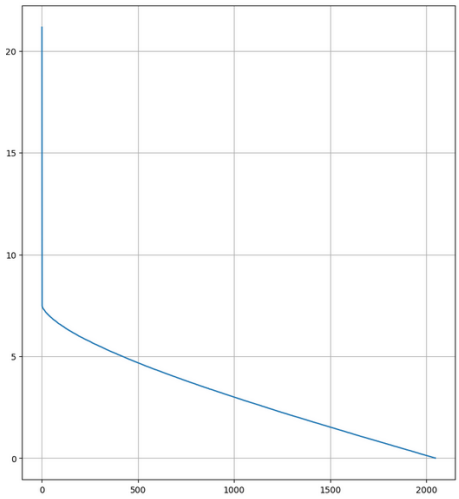


δ = σ2047, b = 1

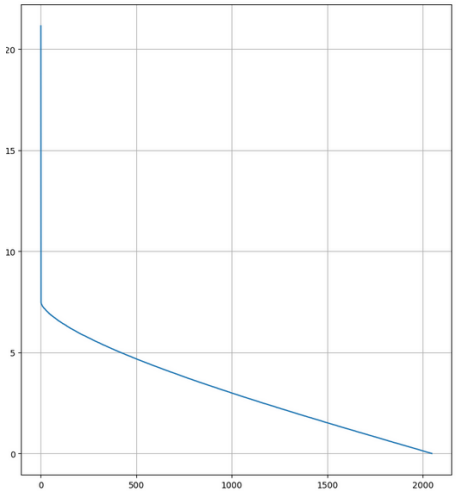


δ = σ2047, b = 4

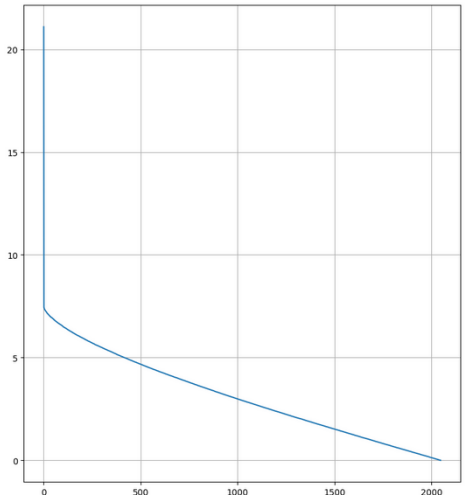
Wykresy dla macierzy wypełnionej w 98% zerami:



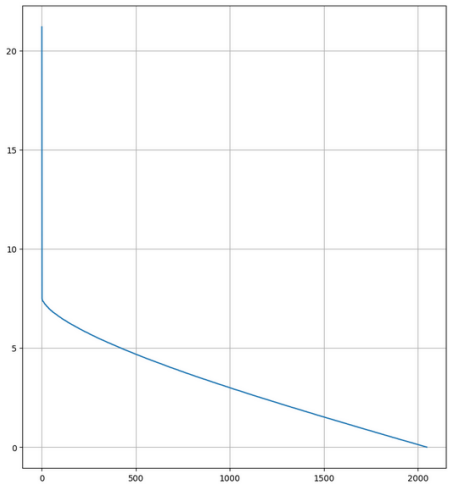
δ = σ0, b = 1



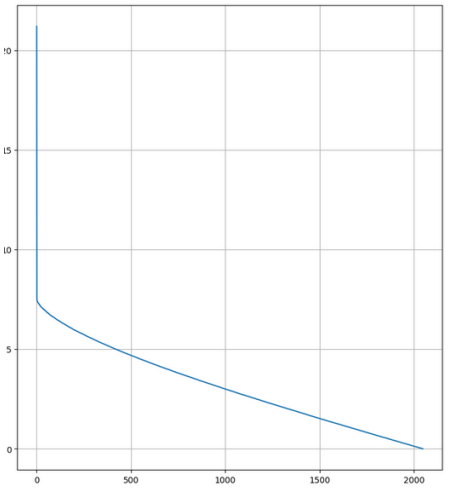
δ = σ0, b = 4



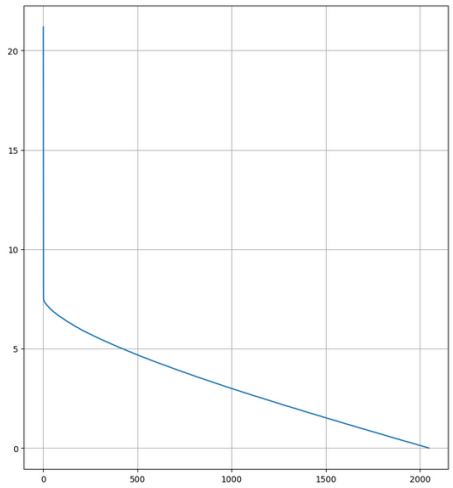
δ = σ1024, b = 1



δ = σ1024, b = 4

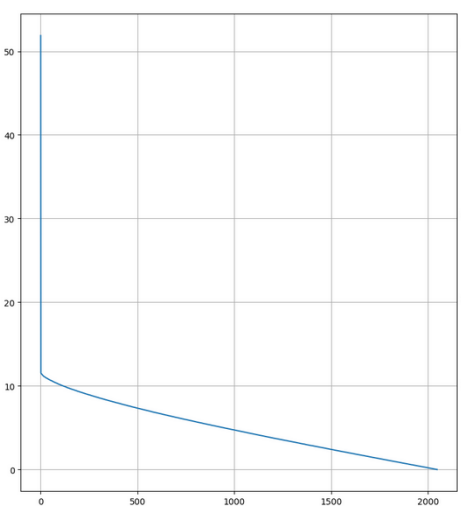


δ = σ2047, b = 1

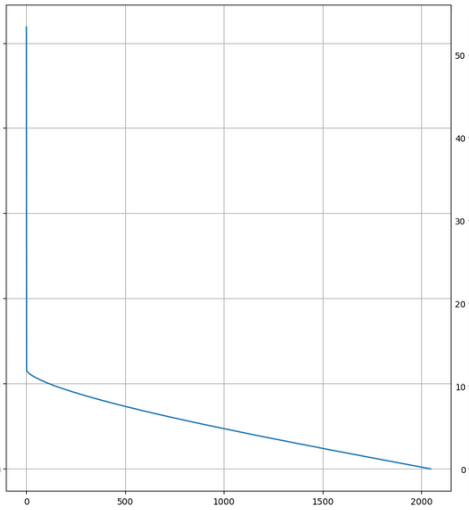


δ = σ2047, b = 4

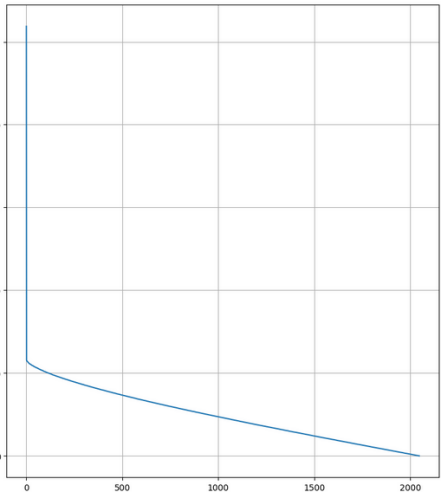
Wykresy dla macierzy wypełnionej w 95% zerami:



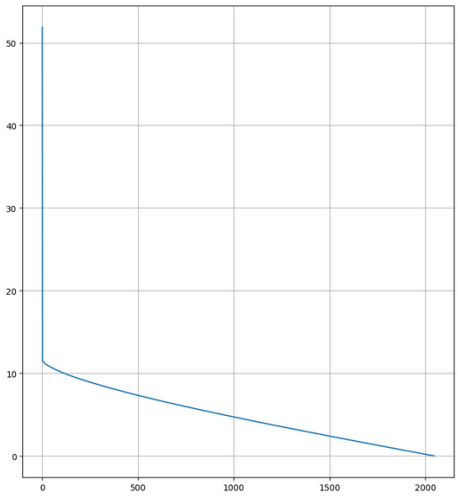
δ = σ0, b = 1



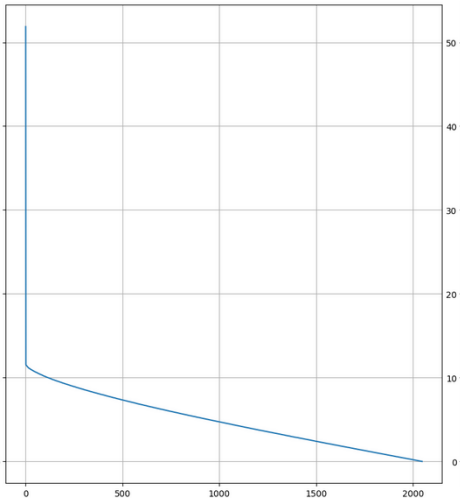
δ = σ0, b = 4



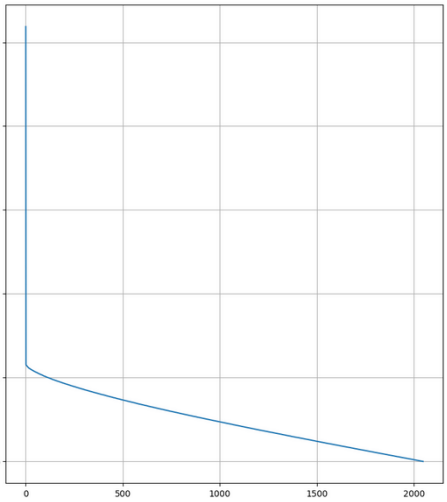
δ = σ1024, b = 1



δ = σ1024, b = 4

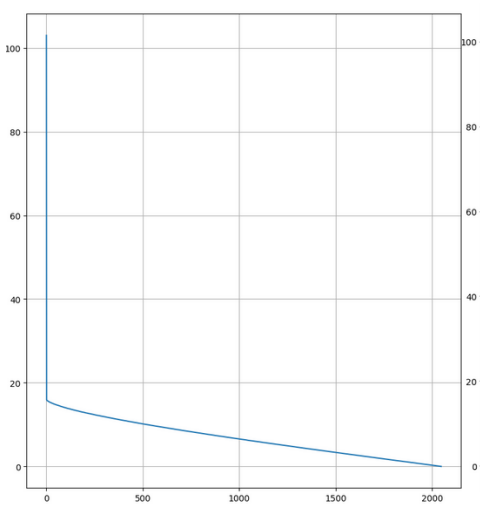


δ = σ2047, b = 1

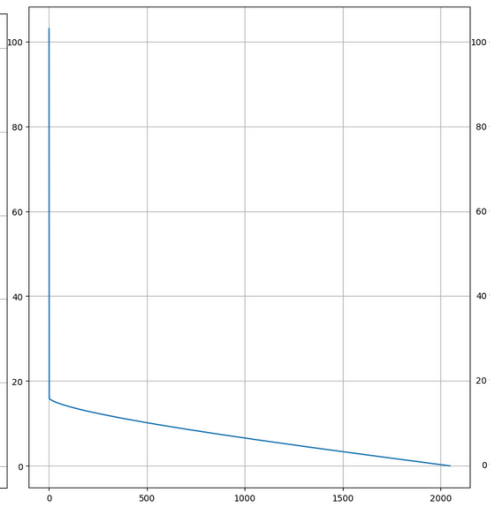


δ = σ2047, b = 4

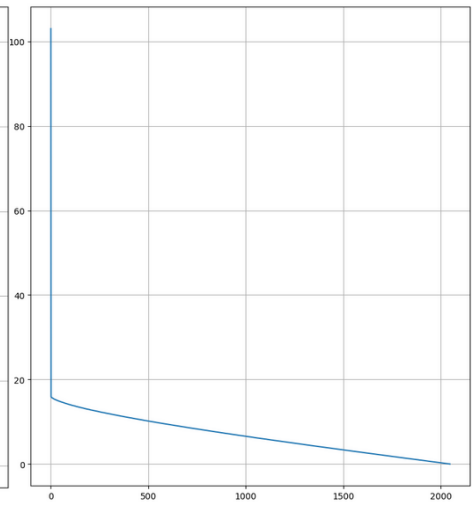
Wykresy dla macierzy wypełnionej w 90% zerami:



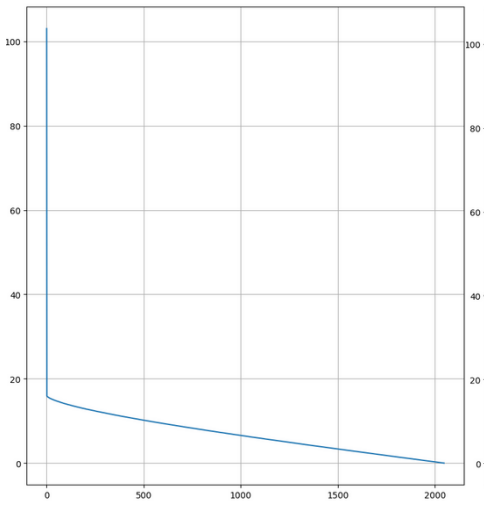
δ = σ0, b = 1



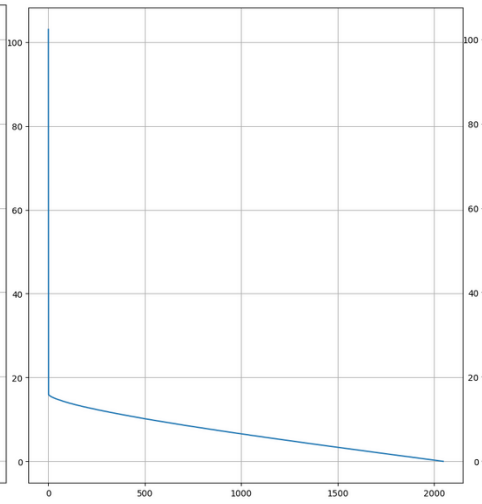
δ = σ0, b = 4



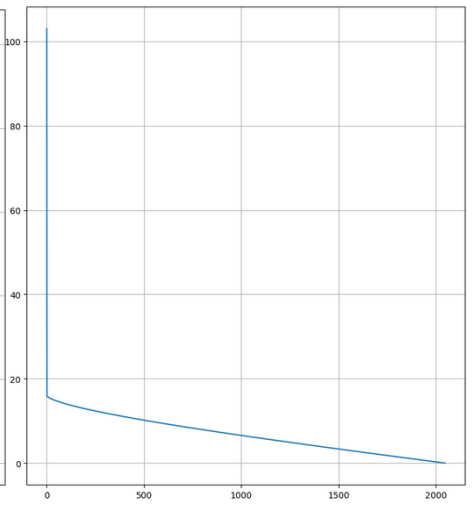
δ = σ1024, b = 1



δ = σ1024, b = 4

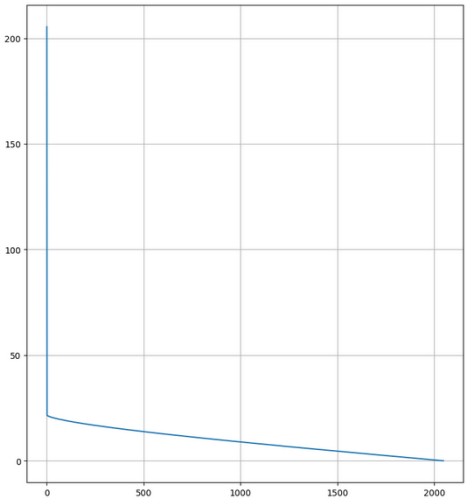


δ = σ2047, b = 1

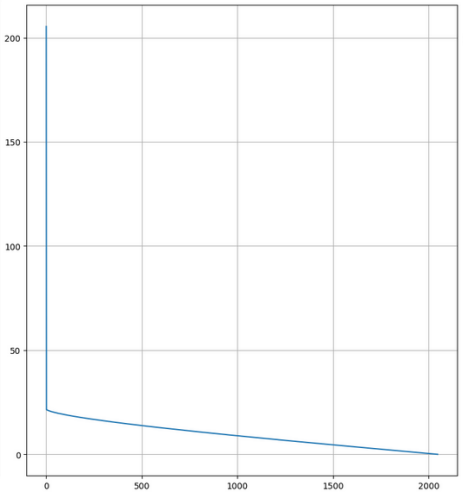


δ = σ2047, b = 4

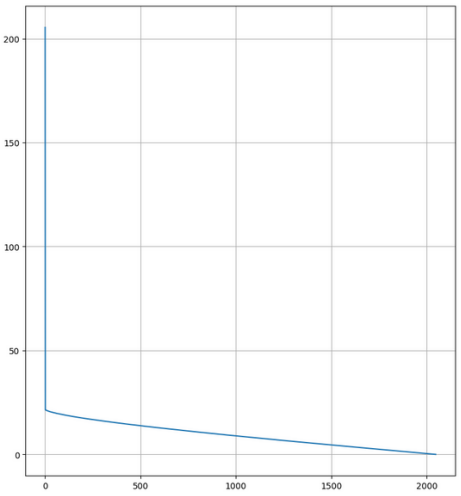
Wykresy dla macierzy wypełnionej w 80% zerami:



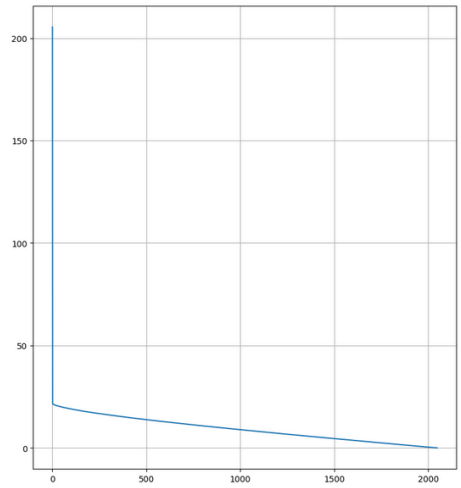
δ = σ0, b = 1



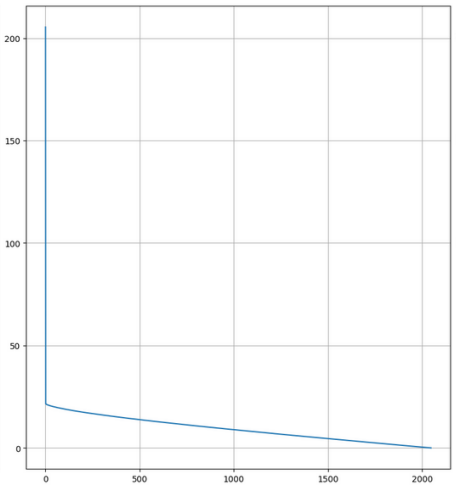
δ = σ0, b = 4



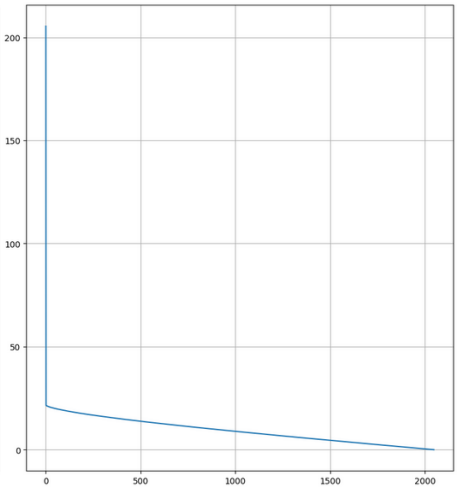
δ = σ1024, b = 1



δ = σ1024, b = 4



δ = σ2047, b = 1



δ = σ2047, b = 4

Wnioski

- Wartość pierwszej z osobliwości jest znacznie większa niż kolejne

- Wraz ze spadkiem wartości osobliwej wartość błędu spada a czas kompresji rośnie

- Im więcej wartości niezerowych tym wartości błędu są większe oraz wydłuża się czas kompresji

- Wartości osobliwości są niemal identycznie dla wszystkich przypadków z taką samą ilością zer