Algorytmy macierzowe

Sprawozdanie z laboratorium nr. 3

Szymon Nowak-Trzos, Dominik Adamczyk

13 grudnia 2023

1 Treść zadania

Zadanie polegało na zaimplementowaniu algorytmu stratnej kompresji macierzy z wykorzystaniem $\boldsymbol{TruncatedSVD(A, \delta, b)}$, oraz zaimplementowania funkcji rysującej macierze.

2 Wstęp

Zgodnie z treścią, zaimplementowany został wskazany algorytm. Całe zadanie zostało napisane w języku Python 3.10. Testy przeprowadzono na urządzeniu z cpu Intel Core i5-1135g7 o bazowej częstotliwości 2.4GHz.

3 Użyte rozwiązania

Do budowy drzewa zaimplementowana została klasa Node:

```
class Node:
    def __init__(self):
        self.rank = None
        self.size = None
        self.singular_values = None
        self.U = None
        self.V = None
        self.sons = None
```

Dodatkowo zaimplementowane zostały funkcje pomocnicze do dzielenia macierzy na części oraz sklejania macierzy z tych części.

```
def matrix_partition(A):
    n = A.shape[0] // 2
    m = A.shape[1] // 2
    A11 = A[:n, :m]
    A12 = A[:n, m:]
    A21 = A[n:, :m]
    A22 = A[n:, m:]
    return A11, A12, A21, A22

def matrix_repartition(A11, A12, A21, A22):
    C = np.block([[A11, A12],
                  [A21, A22]])
    return C
```

Algorytmy zostały zaimplementowane w osobnych funkcjach. Do mnożenia macierzy użyto mnożenia z biblioteki numpy w celach przyśpieszenia obliczeń.

4 Algorytm kompresji macierzy

Poniżej prezentujemy opisy i kody implementowanych algorytmów.

4.1 Truncated SVD

Jako pierwszy został zaimplementowany algorytm częściowej dekompozycji SVD.

Algorytm ten wykorzystuje ***np.linalg.svd(A)*** do obliczenia pełnej dekompozycji SVD, a następnie ucina macierze zgodnie z $\boldsymbol{\delta}$ oraz $\boldsymbol{b}$.

Pseudokod:

1. Oblicz U, s i V poprzez `np.linalg.svd(A)`.
2. Znajdź indeks i w tablicy wartości osobliwych taki, że po lewej od niego są wartości większe niż $delta$
3. Zakładając, że maksymalna liczba wartości osobliwych do zachowania to b:
   - Wybierz minimum spośród i (obliczonego na podstawie $delta$), b i długości listy wartości osobliwych.
4. Przytnij wartości własne i wektory własne na podstawie obliczonego indeksu.
5. **Zwróć przycięte macierze U, s i V.**

```
def truncated_SVD(A, delta, b):
    b = b-1
    U, s, V = np.linalg.svd(A)
    i = find_s_index_for_delta(s, delta)
    idx = min(i, b, len(s))
    return U[:, :idx + 1], s[:idx+1], V[:idx + 1, :]
```

4.2 Kompresja macierzy

Poniższy algorytm tworzy węzeł w drzewie ze skompresowaną macierzą funkcją $\boldsymbol{truncated\_SVD}()$.

Pseudokod:

1. Sprawdź, czy macierz A zawiera jakiekolwiek wartości niezerowe:
   - Jeśli nie, utwórz węzeł v o zerowym rzędzie (0) i ustaw rozmiar na kształt macierzy A.
   - Zwróć węzeł v.
2. Oblicz częściową dekompozycję SVD macierzy A przy użyciu funkcji `truncated_SVD()`
3. Oblicz rangę macierzy na podstawie wartości osobliwych s poprzez funkcję `matrix_rank(s)`.
4. Utwórz węzeł v reprezentujący skompresowaną macierz:
   - Ustaw $rank$ węzła na obliczony rząd macierzy.
   - Oblicz $U \times s$ i przypisz do $v.U$.
   - Przypisz V do $v.V$.
   - Ustaw rozmiar węzła na kształt macierzy A.
5. **Zwróć węzeł** v.

```python
def compress_matrix(A, delta, b):
    if not np.any(A):
        v = Node()
        v.rank = 0
        v.size = A.shape
        return v
    U, s, V = truncated_SVD(A, delta, b)
    rank = matrix_rank(s)
    v = Node()
    v.rank = rank
    v.U = U * s
    v.V = V
    v.size = A.shape
    return v
```

4.3 Tworzenie drzewa

Jeśli spełniony jest **warunek dopuszczalności** to kompresujemy węzeł drzewa, jeśli nie to rozbijamy macierz węzła na cztery podmacierze i tworzymy tam rekurencyjnie drzewa.

Pseudokod:

1. Ogranicz $r + 1$ między 0 a wymiarem macierzy A
2. Oblicz obciętą dekompozycję SVD macierzy A przy użyciu funkcji `truncated_SVD()`
3. Sprawdź **warunek dopuszczalności -** czy ostatnia wartość osobliwa jest mniejsza niż e; lub czy rozmiar macierzy U jest mniejszy niż r:
   - Jeśli tak, utwórz węzeł v poprzez wywołanie funkcji `compress_matrix(A, e, r)`.
4. W przeciwnym razie:
   - Utwórz węzeł v typu Node().
   - Podziel macierz A na podmacierze $A11$, $A12$, $A21$ i $A22$ przy pomocy funkcji `matrix_partition(A)`.
   - Ustaw węzły potomne $v.sons$ poprzez rekurencyjne wywołanie funkcji `create_tree` dla każdej z podmacierzy.
5. **Zwróć węzeł** v.

```
def create_tree(A, r, e):
    r = bound(min(A.shape[0], A.shape[1]), 0, r+1)
    U, s, V = truncated_SVD(A, e, r + 1)
    if s[-1] < e or U.shape[0] <= r:
        v = compress_matrix(A, e, r)
    else:
        v = Node()
        A11, A12, A21, A22 = matrix_partition(A)
        v.sons = [create_tree(A11, r, e),
                  create_tree(A12, r, e),
                  create_tree(A21, r, e),
                  create_tree(A22, r, e)]
    return v
```

4.4 Dekompresja macierzy

Rekurencyjnie przechodzimy drzewo i dla każdego liścia przemnażamy $\boldsymbol{U}$ i $\boldsymbol{V}$.

Pseudokod:

1. Jeśli istnieją węzły potomne $v.sons$:
   - Odzyskaj macierze dla każdego z węzłów potomnych $v.sons$ poprzez rekurencyjne wywołanie funkcji `recover_matrix`.
   - Użyj funkcji `matrix_repartition` do ponownego połączenia odzyskanych macierzy w macierz m.
2. W przeciwnym razie, jeśli $v.rank$ wynosi 0:
   - Utwórz macierz m o rozmiarze $v.size$ zawierającą same zera.
3. W przeciwnym razie:
   - Utwórz macierz m poprzez wykonanie operacji mnożenia macierzy U i V, gdzie U i V są przechowywane w węźle v.
4. **Zwróć macierz** m.

```
def recover_matrix(v):
    if v.sons:
        m = matrix_repartition(recover_matrix(v.sons[0]), recover_matrix(v.sons[1]),
                               recover_matrix(v.sons[2]), recover_matrix(v.sons[3]))
    elif v.rank == 0:
        m = np.zeros(v.size)
    else:
        m = v.U @ v.V
    return m
```

4.5 Rysowanie drzewa oraz macierzy

Reprezentacje skompresowanej macierzy uzyskujemy przez ułożenie $\boldsymbol{U}$ i $\boldsymbol{V}$ w macierzy zer.

Aby uzyskać reprezentację całego drzewa przechodzimy po nim rekurencyjnie i, tak samo jak przy dekompresji macierzy, składamy dużą macierz z mniejszych.

```
def U_V_to_array(U, V):
    n = U.shape[0]
    m = U.shape[1]
    repr = np.zeros((n, n))
    repr[:n, :m] = U
    repr[:m, :n] = V
    return repr

def tree_to_repr(v):
    if v.sons:
        m = matrix_repartition(tree_to_repr(v.sons[0]), tree_to_repr(v.sons[1]),
                               tree_to_repr(v.sons[2]), tree_to_repr(v.sons[3]))
    elif v.rank == 0:
        m = np.zeros(v.size)
    else:
        m = U_V_to_array(v.U, v.V)
    return m

def show_array(repr, zeros=False):
    if zeros:
        plt.spy((repr != 0).astype(int))
    else:
        plt.spy(repr)
    plt.grid(False)
    plt.show()
```

5 Wykresy

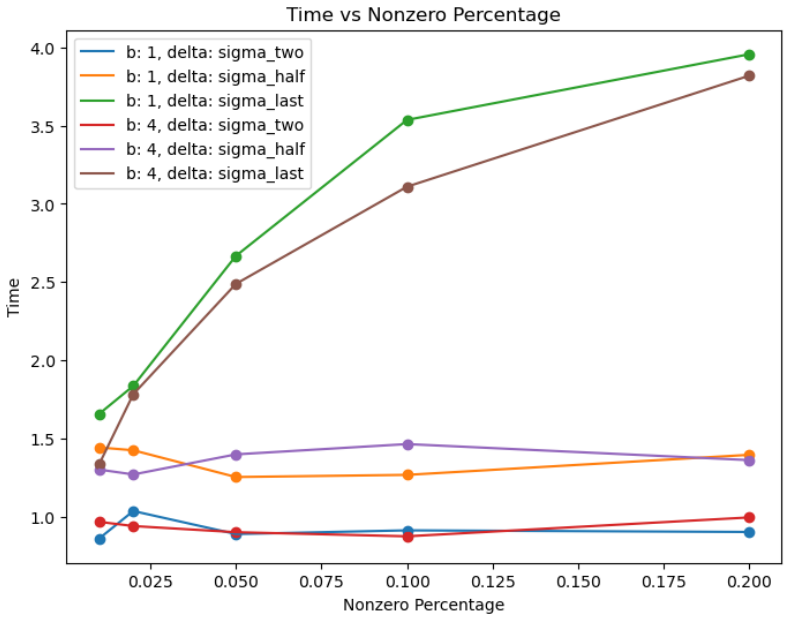
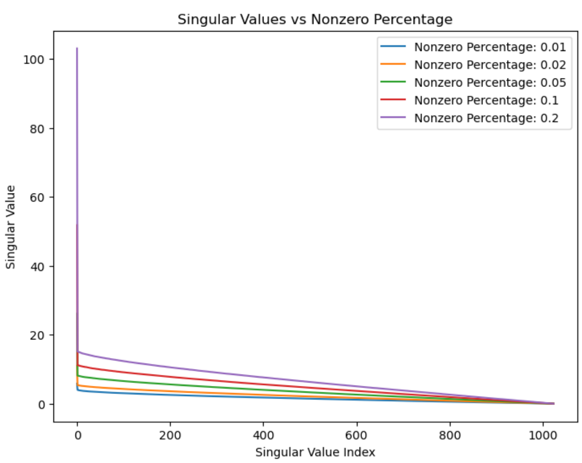
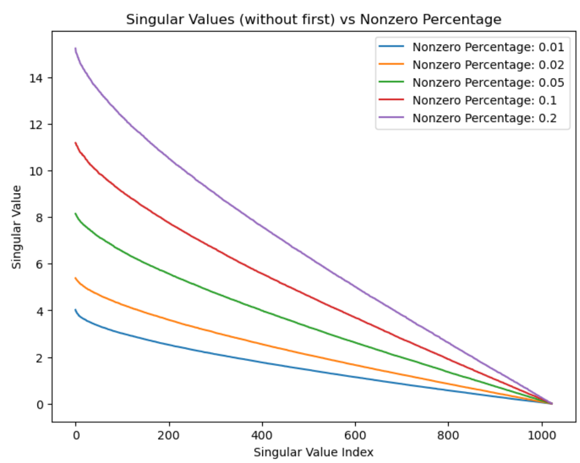


Tabela 1: Wykres czasów algorytmu



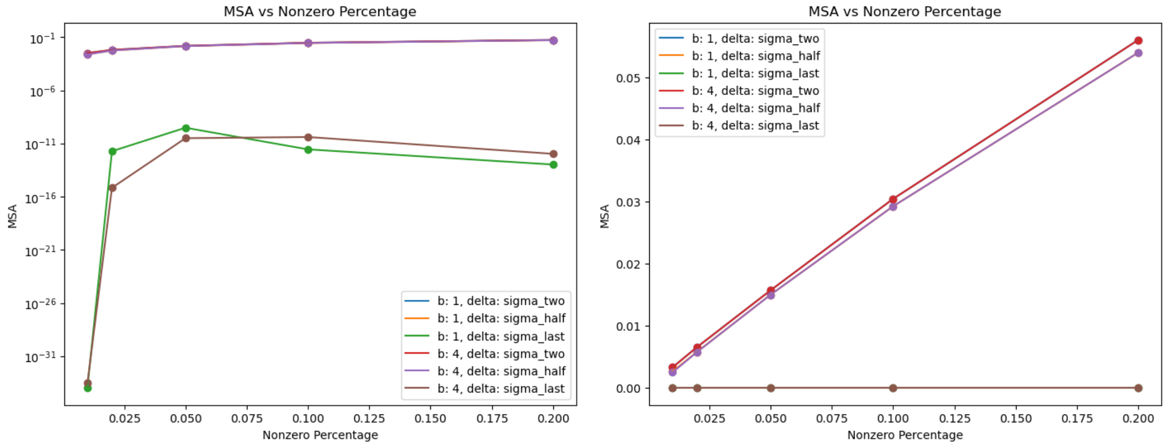
(a) Wartości osobliwe



(b) Wartości osobliwe - bez pierwszej

Czasy intuicyjnie zwiększają się ze zwiększającym się zakresem wartości osobliwych. Najdłużej wykonuje się algorytm dla $\delta = \sigma_{last}$. Nie ma dużej różnicy pomiędzy algorymtem dla $b = 1$ i $b = 4$.

Pierwsza wartość osobliwa jest zawsze znacząco większa od pozostałych.



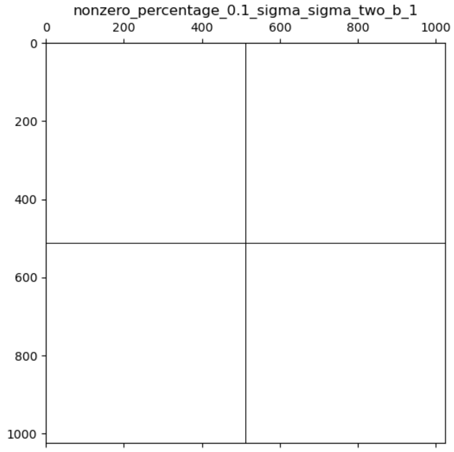
(a) Błędy średniokwadratowe - skala logarytmiczna

(b) Błędy średniokwadratowe

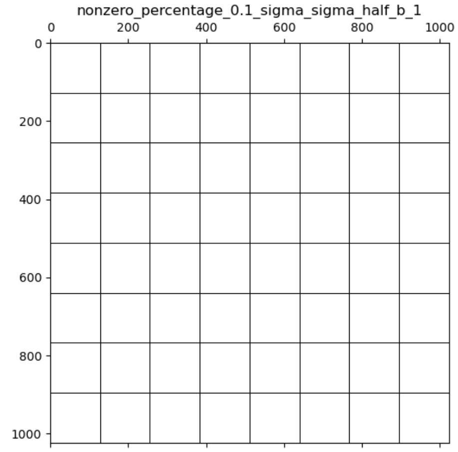
Rysunek 3: Wykresy

Na wykresach widzimy, że błąd średniokwadratowy jest najmniejszy dla $\delta = \sigma_{last}$ - jest to logiczne, bo prawie nic nie wyrzucamy z oryginalnej dekompozycji SVD. Nie ma zauważalnej różnicy między $\delta = \sigma_{two}$ oraz $\delta = \sigma_{half}$. Widać, że błąd dla nich wzrasta liniowo wraz ze wzrostem ilości niezerowych elementów macierzy.

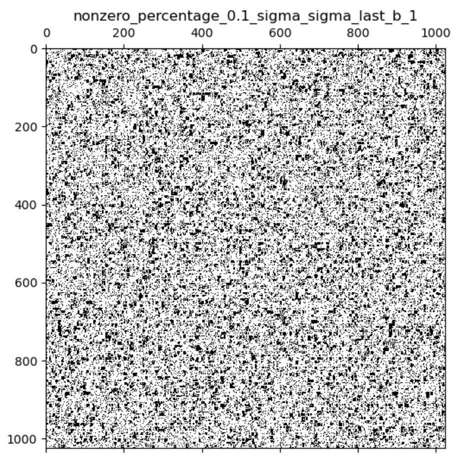
6 H macierze



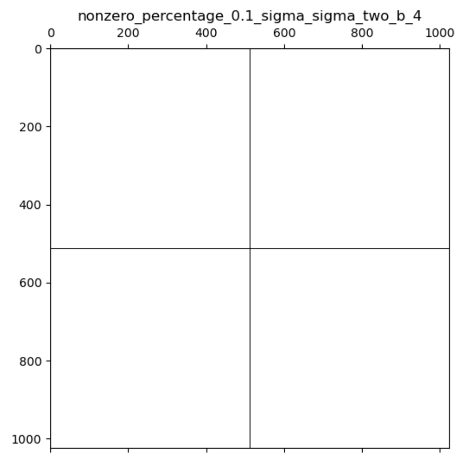
(a) $nonzero_percentage = 0.1, b = 1, \delta = \sigma_{two}$



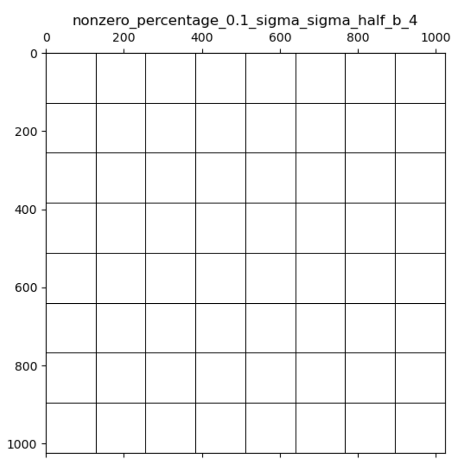
(b) $nonzero_percentage = 0.1, b = 1, \delta = \sigma_{half}$



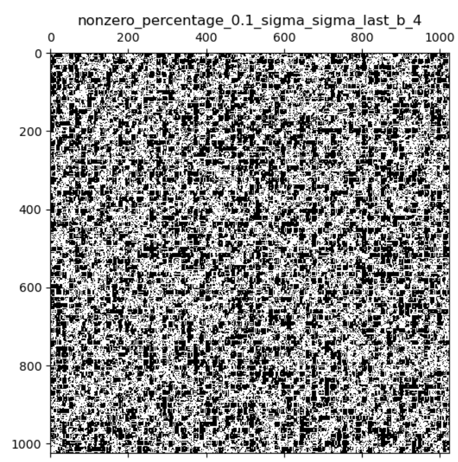
(c) $nonzero_percentage = 0.1, b = 1, \delta = \sigma_{last}$



(d) $nonzero_percentage = 0.1, b = 4, \delta = \sigma_{two}$

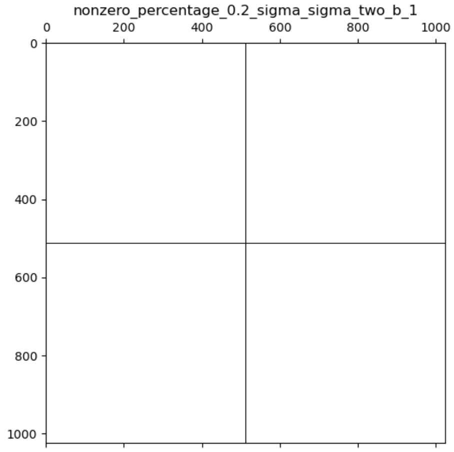


(e) $nonzero_percentage = 0.1, b = 4, \delta = \sigma_{half}$

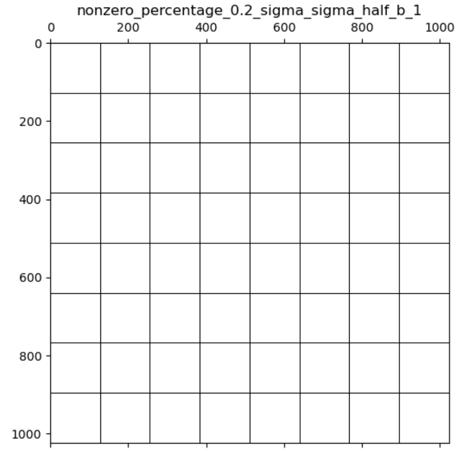


(f) $nonzero_percentage = 0.1, b = 4, \delta = \sigma_{last}$

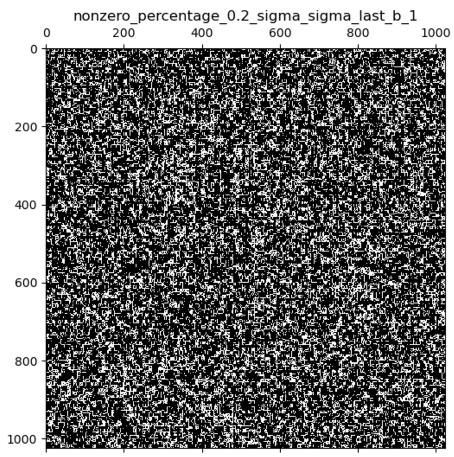
Rysunek 4: H macierze



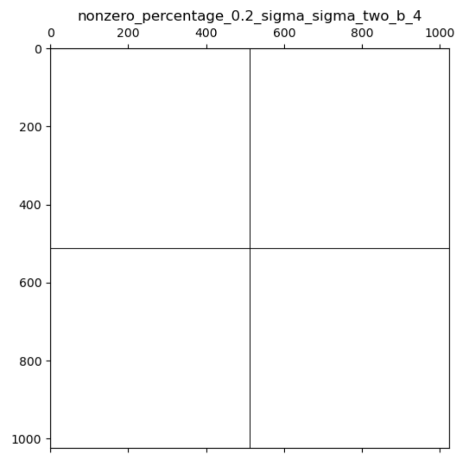
(a) $nonzero_percentage = 0.2, b = 1, \delta = \sigma_{two}$



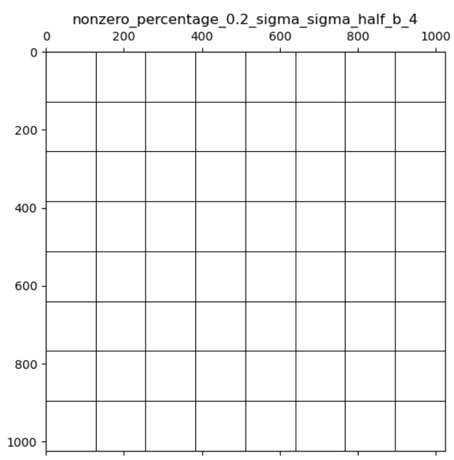
(b) $nonzero_percentage = 0.2, b = 1, \delta = \sigma_{half}$



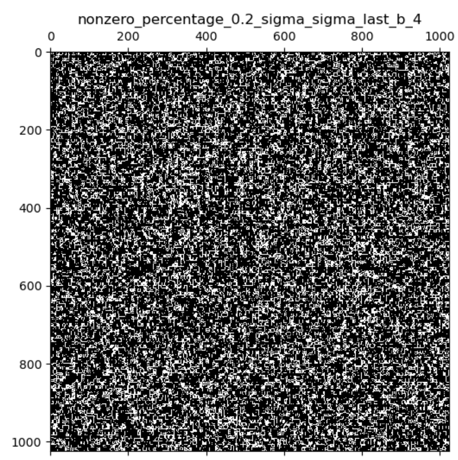
(c) $nonzero_percentage = 0.2, b = 1, \delta = \sigma_{last}$



(d) $nonzero_percentage = 0.2, b = 4, \delta = \sigma_{two}$

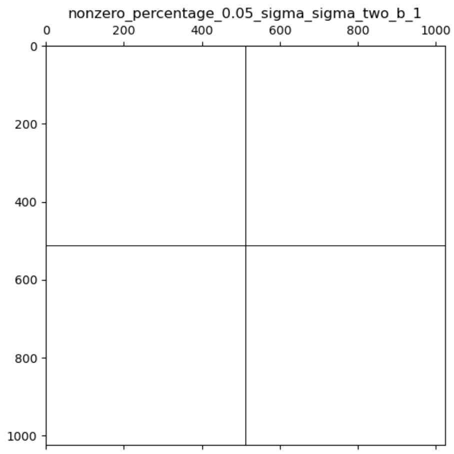


(e) $nonzero_percentage = 0.2, b = 4, \delta = \sigma_{half}$

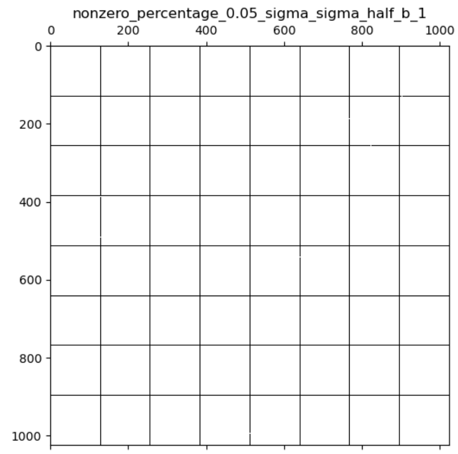


(f) $nonzero_percentage = 0.2, b = 4, \delta = \sigma_{last}$

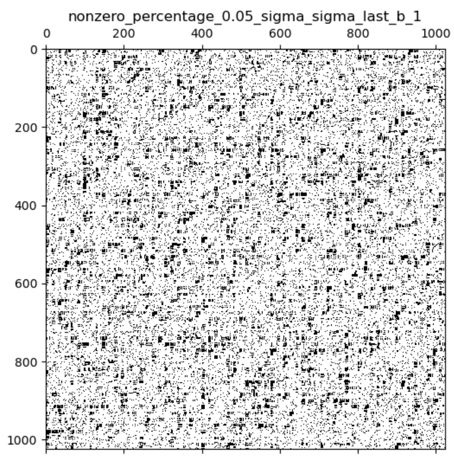
Rysunek 5: H macierze



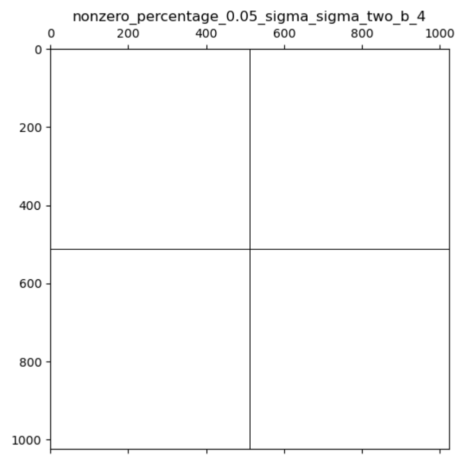
(a) $nonzero_percentage = 0.05, b = 1, \delta = \sigma_{two}$



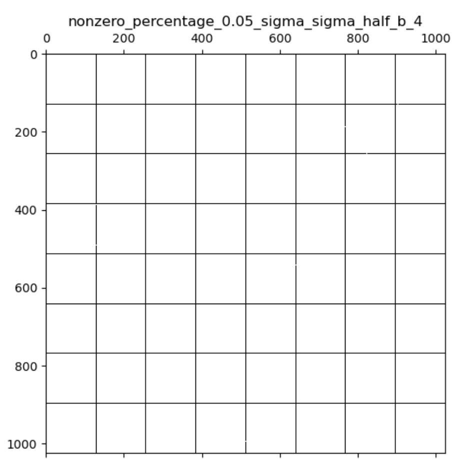
(b) $nonzero_percentage = 0.05, b = 1, \delta = \sigma_{half}$



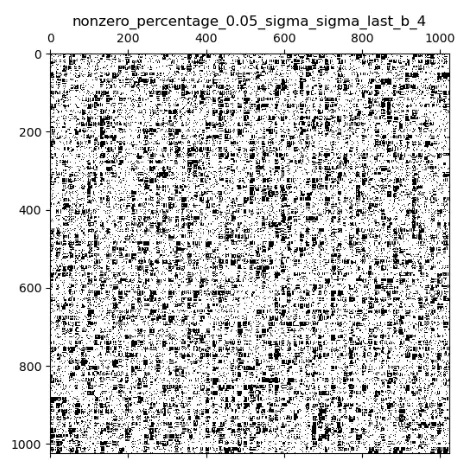
(c) $nonzero_percentage = 0.05, b = 1, \delta = \sigma_{last}$



(d) $nonzero_percentage = 0.05, b = 4, \delta = \sigma_{two}$

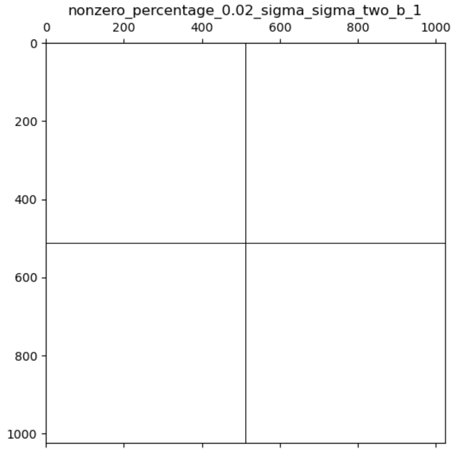


(e) $nonzero_percentage = 0.05, b = 4, \delta = \sigma_{half}$

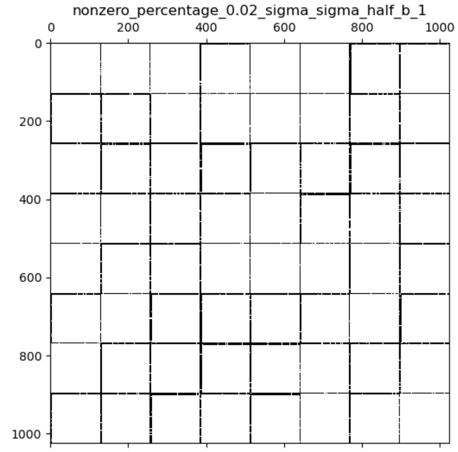


(f) $nonzero_percentage = 0.05, b = 4, \delta = \sigma_{last}$

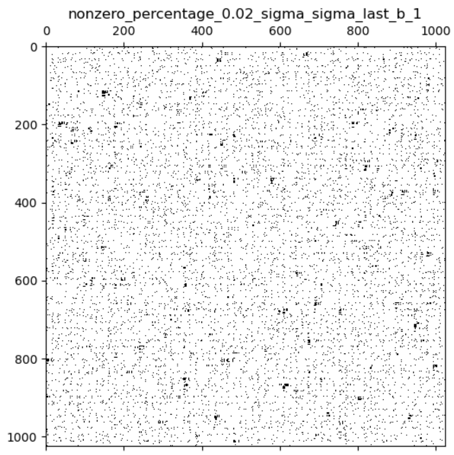
Rysunek 6: H macierze



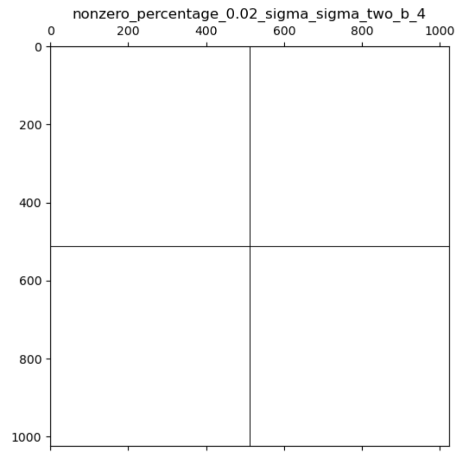
(a) $nonzero_percentage = 0.02, b = 1, \delta = \sigma_{two}$



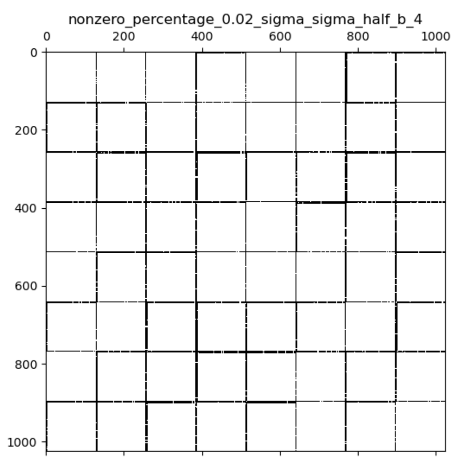
(b) $nonzero_percentage = 0.02, b = 1, \delta = \sigma_{half}$



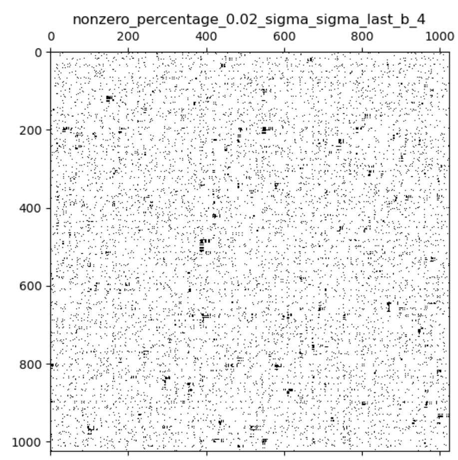
(c) $nonzero_percentage = 0.02, b = 1, \delta = \sigma_{last}$



(d) $nonzero_percentage = 0.02, b = 4, \delta = \sigma_{two}$

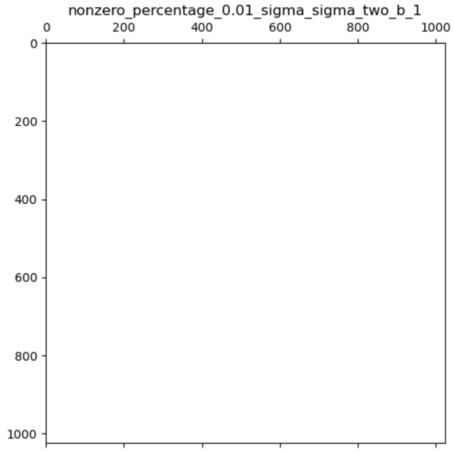


(e) $nonzero_percentage = 0.02, b = 4, \delta = \sigma_{half}$

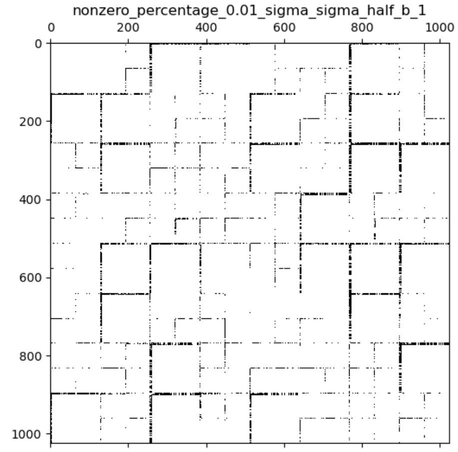


(f) $nonzero_percentage = 0.02, b = 4, \delta = \sigma_{last}$

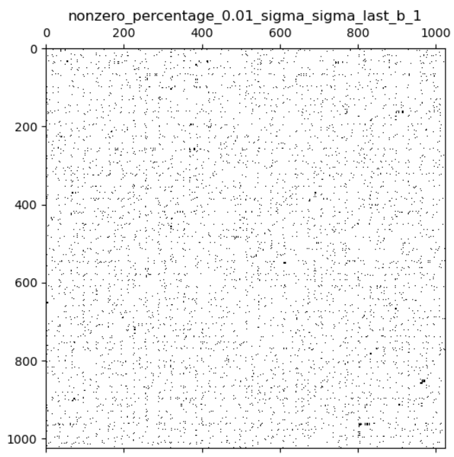
Rysunek 7: H macierze



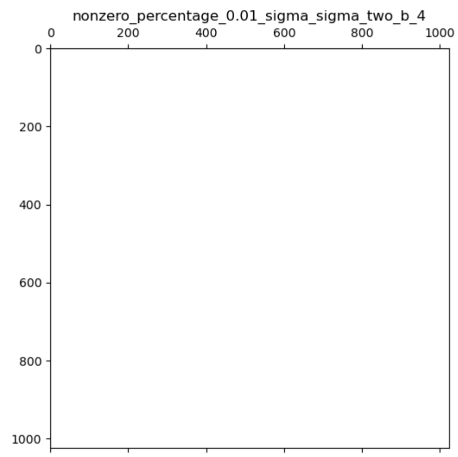
(a) $nonzero_percentage = 0.01, b = 1, \delta = \sigma_{two}$



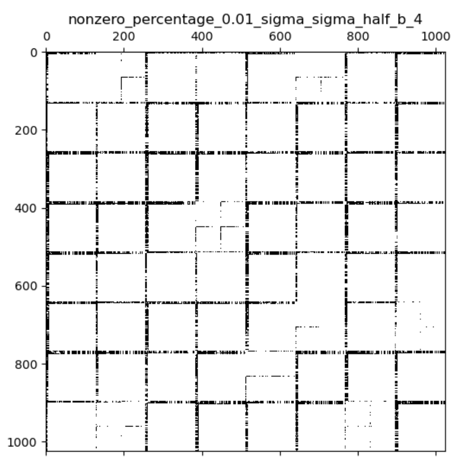
(b) $nonzero_percentage = 0.01, b = 1, \delta = \sigma_{half}$



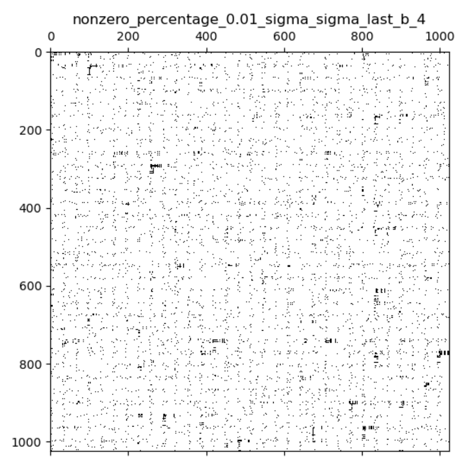
(c) $nonzero_percentage = 0.01, b = 1, \delta = \sigma_{last}$



(d) $nonzero_percentage = 0.01, b = 4, \delta = \sigma_{two}$



(e) $nonzero_percentage = 0.01, b = 4, \delta = \sigma_{half}$



(f) $nonzero_percentage = 0.01, b = 4, \delta = \sigma_{last}$

Rysunek 8: H macierze