Przemysław Rola, Juliusz Wasieleski
Informatyka, III rok, grupa 6
listopad 2023

Algorytmy macierzowe – kompresja Singular Value Decomposition – sprawozdanie

1. Opis ćwiczenia

Naszym zadaniem było , po wybraniu naszego ulubionego języka, wygenerowanie losowych macierzy o dużych rozmiarach (np. 2^{10}) i zaimplementowanie stratnego algorytmu SVD (Singular value decomposition co na polski tłumaczy się jako Rozkład według wartości osobliwych)

Następnie, mieliśmy sprawdzić działanie naszej implementacji dla różnych wartości maksymalnego liczby wartości osobliwych (max rank), różnej minimalnej wartości osobliwej i różnego wypełnienia zerami. Mieliśmy policzyć błąd naszej stratnej kompresji (jako różnicę między oryginalną i zdekompresowaną macierzą) oraz graficznie przedstawić drzewo kompresji na podstawie warunku dopuszczalności.

2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonaliśmy w języku Python przy użyciu Jupyer Notebooka. Do obliczeń, przechowywania danych użyliśmy bibliotek *numpy, pandas, scipy.*

Do rysowania wykresów użyliśmy biblioteki *matplotlib.*

Wszystkie obliczenia prowadziliśmy na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

Jako procenty wypełnienia zerami przyjęliśmy (zgodnie z poleceniem): 99%, 98%, 95%, 90%, 80%. Literą b określaliśmy maksymalną liczbę wartości osobliwych branych pod uwagę podczas kompresji, za wartości b przyjmowaliśmy 1 i 4. Literą σ oznaczaliśmy wartości osobliwe, jako minimalne przyjmowaliśmy wartości o indeksach 0, 511 i 1023 (wynika to z naszej implementacji ponieważ odrzucamy dopiero wszystkie mniejsze wartości).

3. Implementacje

3.1 Singular value decomposition

3.1.1 Pseudokod

stwórz_drzewo*(r, epsylon) :*

U, D, V = truncatedSVD(A[t_min:t_max, s_min :s_max], r+1)

Jeżeli *t_max == t_min + r***:**

stwórz_drzewo(U, D, V)

W przeciwnym wypadku, jeżeli *D[r] < epsylon***:**

stwórz_liść (U, D, V)

W przeciwnym wypadku:

stwórz węzeł v

dodaj do węzła v nowe_drzewo(t_min, t_new_max, s_min, s_new_max)

dodaj do węzła v nowe_drzewo(t_min, t_new_max, s_new_max + 1, s_max)

dodaj do węzła v nowe_drzewo(t_new_max + 1, t_max, s_min, s_new_max)

dodaj do węzła v nowe_drzewo(t_new_max + 1, t_max, s_new_max + 1, s_max)

3.1.2 Istotne fragmenty implementacji

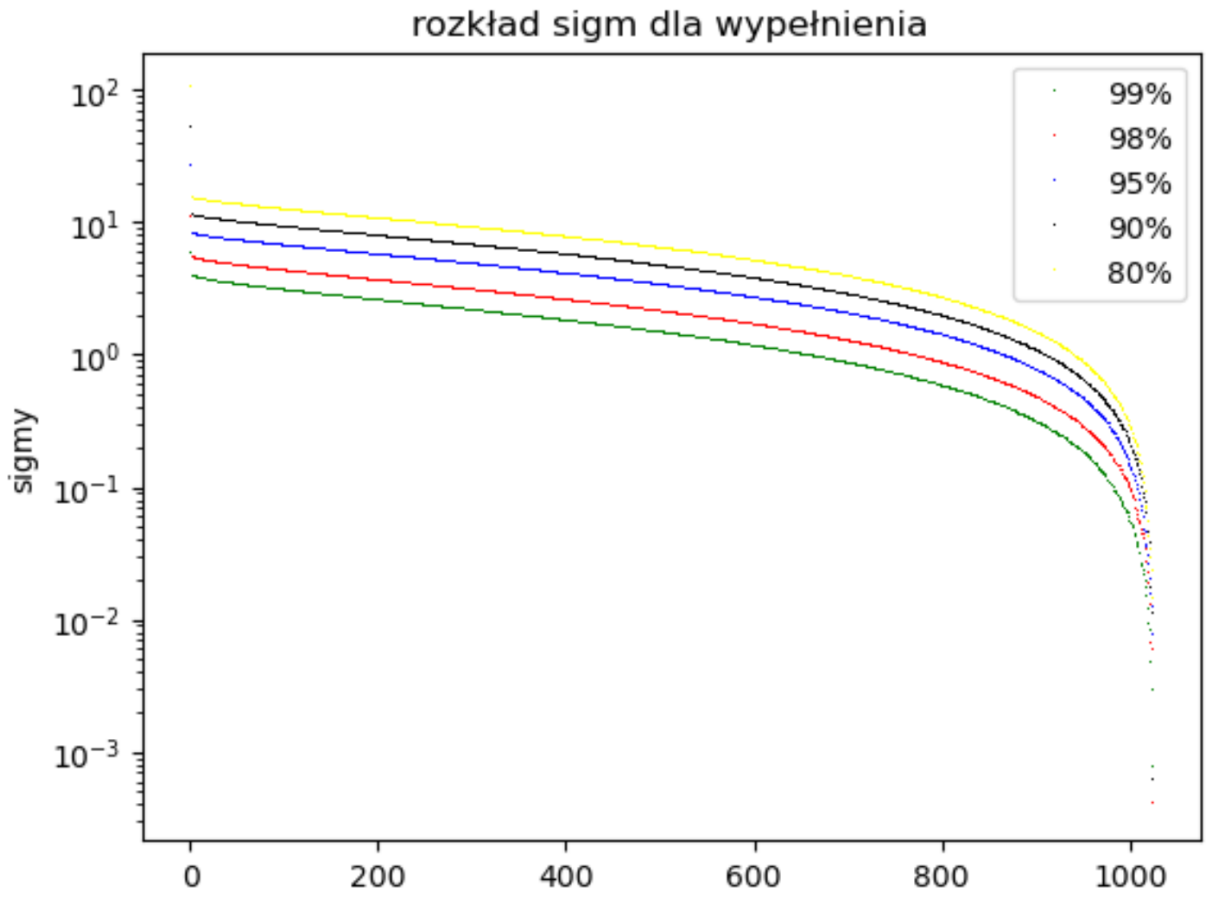
Niektóre kwestie rozwiązaliśmy inaczej niż w pseudokodzie przedstawionym na wykładzie. Zacznijmy od tego, że u nas każdy węzeł jest instancją klasy Compress_Tree(). Compress_Tree w konstruktorze przyjmuje macierz oraz minimalne i maksymalne numery wierszy i kolumn. Jako truncatedSVD używamy metody randomised_svd z modułu sklearn.utils.extmath.

```
class CompressTree:
    def __init__(self,matrix, row_min, row_max, col_min, col_max):
        self.matrix = matrix
        self.row_min = row_min
        self.row_max = row_max
        self.col_min = col_min
        self.col_max = col_max


        self.leaf = False
        '''
         UL | UR
        ----+----
         DL | DR

        '''
        self.childs = [[None, None], [None, None]]

    def make_leaf(self, U, Sigma, V):
            self.leaf = True
            self.u = U
            self.s = Sigma
            self.v = V

    def create_tree(self, r, epsylon):

        U, Sigma, V = randomized_svd(self.matrix[self.row_min:self.row_max, self.col_min: self.col_max],
n_components=r+1)
        if self.row_max == self.row_min + r:
            self.make_leaf(U, Sigma, V)
        elif Sigma[r] < epsylon:
            self.make_leaf(U, Sigma, V)
        else:
            rows = [self.row_min, (self.row_min + self.row_max)//2, self.row_max]
            cols = [self.col_min, (self.col_min + self.col_max)//2, self.col_max]
            for i in range(2):
                 for j in range(2):
                      self.childs[i][j] = CompressTree(self.matrix, rows[i], rows[i+1], cols[j], cols[j+1])
                      self.childs[i][j].create_tree(r, epsylon)
```

4. Analiza wykonanych pomiarów

4.1 Analiza wygenerowanych macierzy

Jako, że mieliśmy wygenerować 5 losowych macierzy wypełnionych losowymi liczbami z zakresu (0, 1) z różną procentową częścią zer. Jedyną charakterystyką tych tablic jaką możemy przedstawić jest wykres kolejnych wartości osobliwych. Nie zamieszczamy tutaj tabeli z wszystkimi wartościami osobliwymi ponieważ dla każdej z macierzy jest ich 1024 i jest to liczba dla której tabela byłaby zupełnie nieczytelna.



Wykres 1

Zauważmy, że powyższy wykres ma skalę logarytmiczną. Można więc łato dostrzec, że wartości osobliwe mają bardzo duże rozbieżności w zależności od indeksu danej wartości osobliwej, ale sam rozkład tych wartości jest dość podobny dla wszystkich macierzy. Co więcej analizując te sigmy zauważamy, że pierwsze z nich są zdecydowanie większe niż kolejne. Dlatego na początku każdego z wykresów taki pik. Potem mają one dość stale wartości, których tempo spadku wzrasta. Próbowaliśmy znaleźć funkcję która pokrywa się z wartościami osobliwymi dla macierzy, ale nie udało nam się dobrze dopasować żadnej funkcji. Najbliższa była funkcja $8.841x^{-0.3}$, ale i ta pozostawiała bardzo dużo do życzenia.

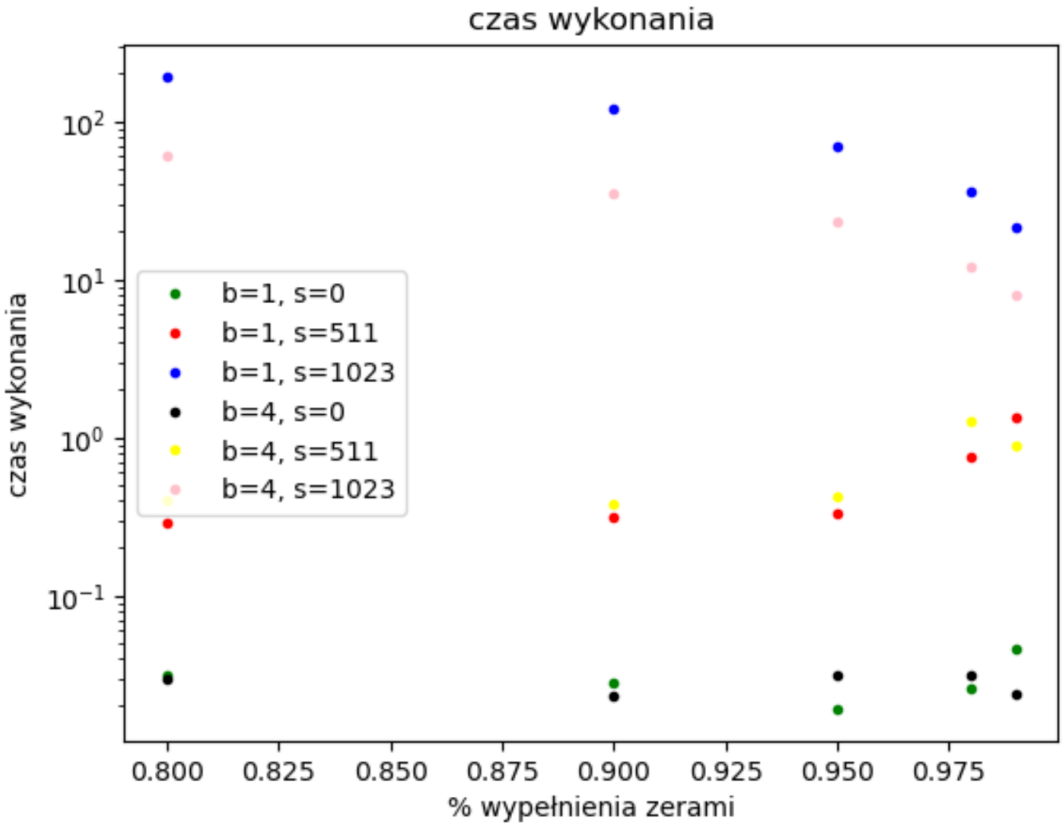
4.2 Pomiary algorytmu svd

ile zer	b	sigma index	sigma	błąd	czas wykonania [s]
0.99	1	0	5.827808	3.415954e+03	0.024728
0.99	1	511	1.410532	2.577591e+03	1.049251
0.99	1	1023	0.000114	2.148806e-29	23.434746
0.99	4	0	6.003669	3.470345e+03	0.030462
0.99	4	511	1.419443	2.125608e+03	0.844124
0.99	4	1023	0.000166	1.737049e-27	7.047108
0.98	1	0	10.917001	6.845573e+03	0.026899
0.98	1	511	2.042171	5.837295e+03	0.807798

ile zer	b	sigma index	sigma	błąd	czas wykonania [s]
0.98	1	1023	0.002861	5.902753e-29	37.640369
0.98	4	0	10.834727	6.657954e+03	0.029410
0.98	4	511	2.036892	5.524139e+03	0.368032
0.98	4	1023	0.003878	4.354615e-27	11.321288
0.95	1	0	26.193285	1.667102e+04	0.026047
0.95	1	511	3.251944	1.580979e+04	0.422981
0.95	1	1023	0.000594	2.006083e-28	81.987803
0.95	4	0	26.126504	1.637905e+04	0.034327
0.95	4	511	3.230400	1.415183e+04	0.615400
0.95	4	1023	0.001388	1.172426e-26	26.741694
0.90	1	0	51.986919	3.225271e+04	0.032814
0.90	1	511	4.527755	3.081347e+04	0.369233
0.90	1	1023	0.004046	5.176091e-28	108.832354
0.90	4	0	51.732365	3.172229e+04	0.020945
0.90	4	511	4.527537	2.782144e+04	0.393154
0.90	4	1023	0.001369	2.639185e-26	31.185451
0.80	1	0	102.828648	5.899354e+04	0.018777
0.80	1	511	6.142997	5.650888e+04	0.293375
0.80	1	1023	0.006866	1.258108e-27	178.811216
0.80	4	0	103.094506	5.854458e+04	0.032979
0.80	4	511	6.158618	5.175073e+04	0.397526
0.80	4	1023	0.001343	3.835720e-26	53.749949

Tab. 1 Pomiary algorytmu svd

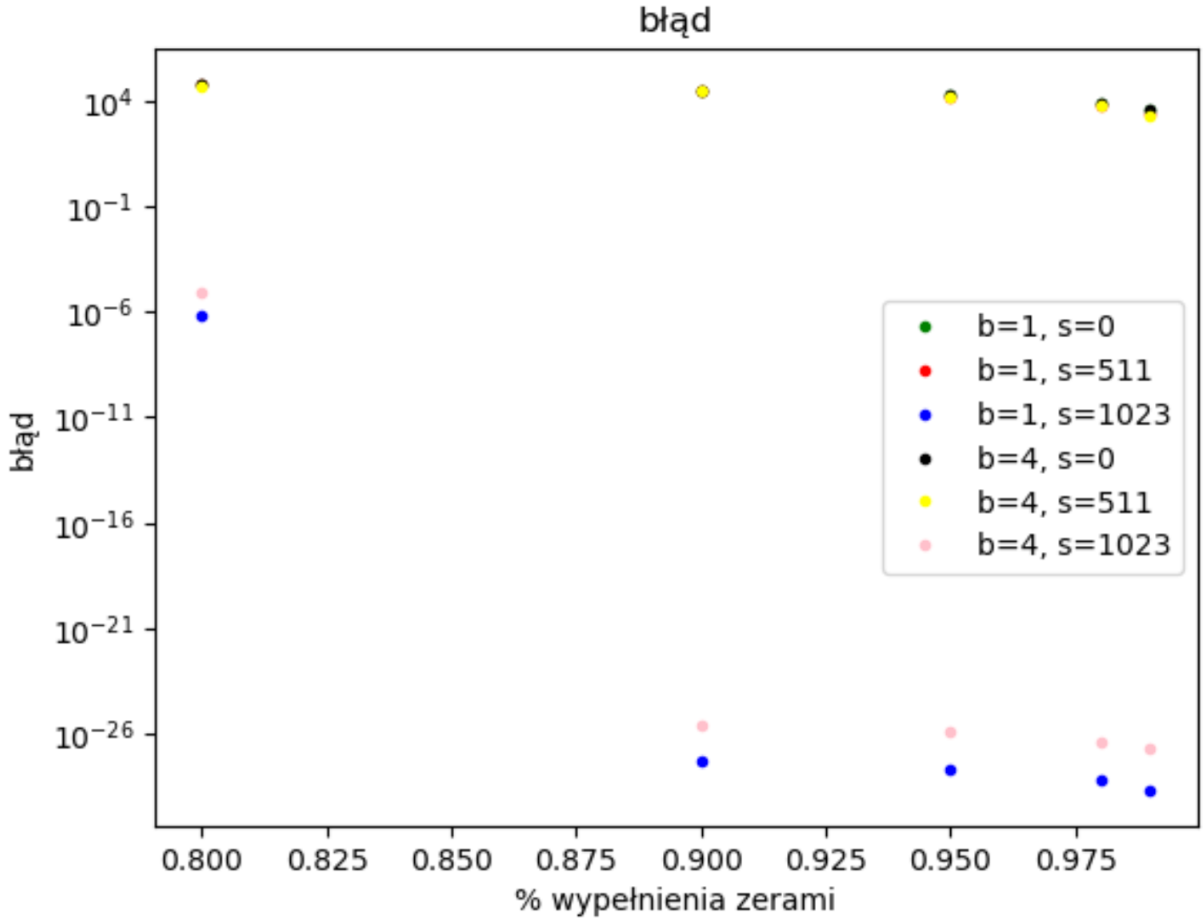
4.3 Porównanie czasów dla różnych macierzy i poziomów kompresji



Wykres 2

Na powyższym wykresie można dostrzec kilka ciekawych trendów. Po pierwsze, czasy wykonania podzieliły się na trzy podgrupy: tę która brała jako minimalną sigmę, pierwszą z nich, tę która brała 512 i tę która brała ostatnią. Im dalsza (więc i mniejsza) sigma tym dłuższy czas. Dla indeksów sigm równych 0 i 511 nie dostrzegamy jakichkolwiek prawidłowości poza tym, że zazwyczaj svd liczyło się dłużej dla b=4 niż b=1. Odwrotne obserwacje możemy natomiast otrzymać dla sigmy o indeksie 1023. Tutaj widzimy ewidentnie, że im więcej zer jest w macierzy tym szybciej możemy policzyć svd. Dodatkowo widać, że dłużej zajmowała dekompozycja dla b=1 niż dla b=4. Działo się tak ponieważ kompresja musiała zejść dużo niżej w rekurencyjnym drzewie dla b = 1.

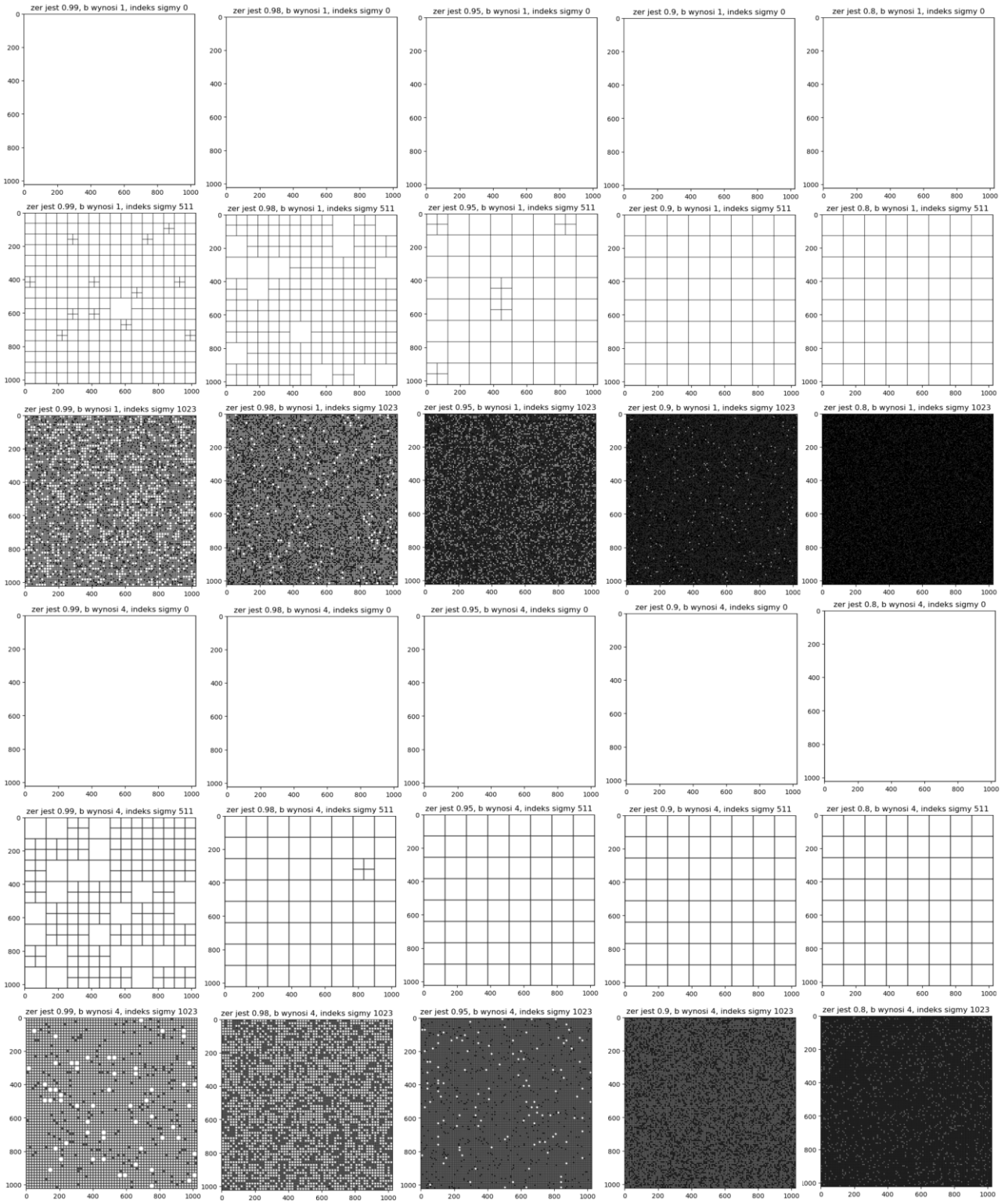
4.4 Porównanie błędów dla różnych macierzy i poziomów kompresji



Wykres 3

Pierwszym z trendów, które obserwujemy dla błędów naszych algorytmów svd jest fakt, że wraz ze wzrostem liczby zer spada stop błędu. Dużo ciekawszy jest wpływ najmniejszej sigmy na błąd. Widzimy, że korzystając ze wszystkich sigm (indeks = 1023) błąd prawie nie występuje. Tak być powinno, bo mówimy w tym wypadku o kompresji bezstratnej. W przeciwnieństwie do niej, zarówno dla indeksu sigm równego 511 jak i 0 błędy są na bardzo zbliżonym poziomie i dość dalekim od błędu kompresji bezstratnej. Można z tego wysnuć wniosek, że zarówno stosowanie większego b maksymalnego jak i mniejszej sigmy minimalnej nie poprawia znacząco dokładności kompresji. Zatem jeśli godzimy się na straty wystarczy wziąć największą z sigm jako sigmę minimalną i ograniczyć koszty obliczeniowe i czasowe kompresji.

4.5 Obserwacje narysowanych drzew



Rys.1 rysunki drzew zgodne z którymi dzieliliśmy macierze

Widać na nich, podobne zjawisko jak na wykresie nr 2, że macierze dzielą nam się w ten sam sposób na trzy podgrupy, w zależności od indeksu sigmy. Widać, że im mniejsza sigma tym dokładniejszy podział. Jest to też przyczyną obserwacji, że dla mniejszej sigmy minimalnej mamy mniejsze straty na kompresji.

5. Wnioski

- Pierwsza z wartości osobliwych jest znacznie większa niż kolejne
- Wraz ze spadkiem minimalnej wartości osobliwej rośnie czas kompresji svd, a błąd nie wydaje się maleć znacząco.
- Wraz ze wzrostem procentowej liczby zer w macierzy (spadkiem gęstości macierzy) spada błąd kompresji oraz czas jej wykonania.
- Czasy wykonania jak i stratność są mocno powiązane z głębokością drzewa podziału macierzy.