# Raport 3

## Kompresja macierzy

*Maciej Pięta, Michał Kobiera, Adam Mytnik*

## 1. Algorytm

```
t_min, t_max, s_min, s_max :
if matrix[t_min :t_max, s_min :s_max] is zero :
      create and return new zero node
U, S, V <- SVD(matrix[t_min:t_max, s_min :s_max], rank)
S <- S[S >= eps]
rank = size of S
if rank == 0 then create and return new zero Node
with m <- matrix[t_min :t_max, s_min :s_max]
if is_admissible(m, S, rank, eps) then
      create and return new Node where:
            v.U = U
            v.S = S
            v.V = V
            v.rank = rank
else
      update t, s
      create v children
create and return new Node
```

## 2. Istotne fragmenty kodu

Do generowania macierzy o podanych parametrach użyliśmy funkcji sparse z modułu scipy

```python
def random_matrix(n,m,density):
    A = scs.random(n,m,density)
    return A.toarray()
```

**Funkcje pomocnicze**

```
class Node:
    def __init__(self, t_min, t_max, s_min, s_max):
        self.t_min = t_min
        self.t_max = t_max

        self.s_min = s_min
        self.s_max = s_max

        self.rank = None

        #[A11, A12]
        #[A21, A22]

        self.U = None
        self.S = None
        self.VT = None

        self.zeros = False


        self.children = []
```

```
def is_admissible(matrix, S, r, eps):
    if min(matrix.shape) <= r:
        return True

    if S[-1] < eps:
        return True

    return False
```

```
def filter_results(results, filters):
#filters - dict {'density': wartosc/None, 'size': wartosc/None, 'rank': wartosc/None, 'sval': wartosc/None}
    filtered_results = results.copy()

    hdict = {'density' : 0, 'size' : 1, 'rank': 2, 'sval' : 3}

    for param, value in filters.items():
        if value is not None:
            filtered_results = {key: val for key, val in filtered_results.items() if key[hdict[param]] == value}

    return filtered_results
```

**Kompresja macierzy**

```
def create_tree(matrix, r, eps):
    n, m = matrix.shape


    def create_tree_r(t_min, t_max, s_min, s_max):
        nonlocal matrix, r, eps

        if not np.any(matrix[t_min:t_max, s_min:s_max]):
            v = Node(t_min, t_max, s_min, s_max)
            v.rank = 0
            v.zeros = True
            return v

        U,S,V = randomized_svd(matrix[t_min:t_max, s_min:s_max], r)

        S = S[S >= eps]
        rank = len(S)

        U = U[:, :rank]
        V = V[:rank, :]
        if rank == 0:
            v = Node(t_min, t_max, s_min, s_max)
            v.rank = 0
            v.zeros = True
            return v


        if is_admissible(matrix[t_min:t_max, s_min:s_max], S, r, eps):
            v = Node(t_min, t_max, s_min, s_max) #compress matrix
            v.rank = rank
            v.U = U
            v.S = S
            v.V = V
            return v

        else:
            v = Node(t_min, t_max, s_min, s_max)

            t_newmax = (t_min + t_max)//2
            s_newmax = (s_min + s_max)//2

            v1 = create_tree_r(t_min, t_newmax, s_min, s_newmax)
            v2 = create_tree_r(t_min, t_newmax, s_newmax, s_max)
            v3 = create_tree_r(t_newmax, t_max, s_min, s_newmax)
            v4 = create_tree_r(t_newmax, t_max, s_newmax, s_max)

            v.children = [v1, v2, v3, v4]
        return v
    return create_tree_r(0, n, 0, m)
```

### Dekompresja macierzy

```
def decompress(node):
    if node.rank is not None:
        if node.rank > 0:
            return node.U@np.diag(node.S)@node.V
        else:
            return np.zeros((node.t_max - node.t_min, node.s_max - node.s_min))
    else:
        return np.vstack(
            (
                np.hstack((decompress(node.children[0]), decompress(node.children[1]))),
                np.hstack((decompress(node.children[2]), decompress(node.children[3]))),
            )
        )
```

### Sprawdzenie poprawności:

```
s = 2**10
a = scs.random(s,s,density = 0.1)
a = a.toarray()
print(a)
```

```
 [[0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.9338018  ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]
  ...
  [0.         0.         0.36043936 ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.45906801]
  [0.         0.99780373 0.         ... 0.         0.         0.        ]]
```

```
tree = create_tree(a, 2, 10**(-8))
#print(decompress(tree))
```

```
print(np.sum(np.square(a - decompress(tree))))
```

```
 8.543763591997999e-28
```

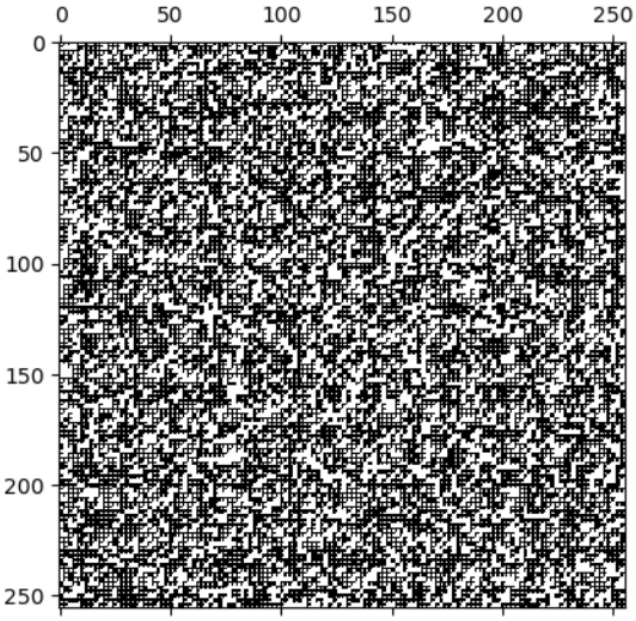
## Rysowanie wyników

```
def plot_matrix(root, matrix, title = None):
    size = matrix.shape
    tmap = np.zeros(size)

    def fill_map(node):
        nonlocal tmap
        if node.rank is not None:
            if node.rank != 0:
                tmap[node.t_min:node.t_max, node.s_min:node.s_min + node.rank] = 1
                tmap[node.t_min:node.t_min + node.rank, node.s_min:node.s_max] = 1
        else:
            for submatrix in node.children:
                fill_map(submatrix)
    fill_map(root)
    f, ax = plt.subplots(1,1)
    ax.spy(tmap)
    if title is not None:
        ax.set_title(title)
```

## Pomiary czasów

```
sizes = [2**i for i in range(8,12)]
densities = [0.01, 0.05, 0.1, 0.2]

results = {}
for density in densities:
    for size in sizes:
        matrix = random_matrix(size,size,density)

        _, S, _ = np.linalg.svd(matrix)
        singular_values = [('0', S[0]), ('n//2', S[size//2]), ('n', S[-1])]

        for b in [2,4,8]:
            for ind, singular_value in singular_values:
                start = time()
                tree = create_tree(matrix, b, singular_value)
                end = time()
                print(density, size, b, singular_value, end-start)

                ftime = f"{end - start:.5f}"
                plot_matrix(tree, matrix, title = f"{size}x{size}\n density = {density}\n rank = {b}\n singular value cut-
                x = (density, size, b, ind)
                results[x] = end - start
```

# 3. Wyniki pomiarów

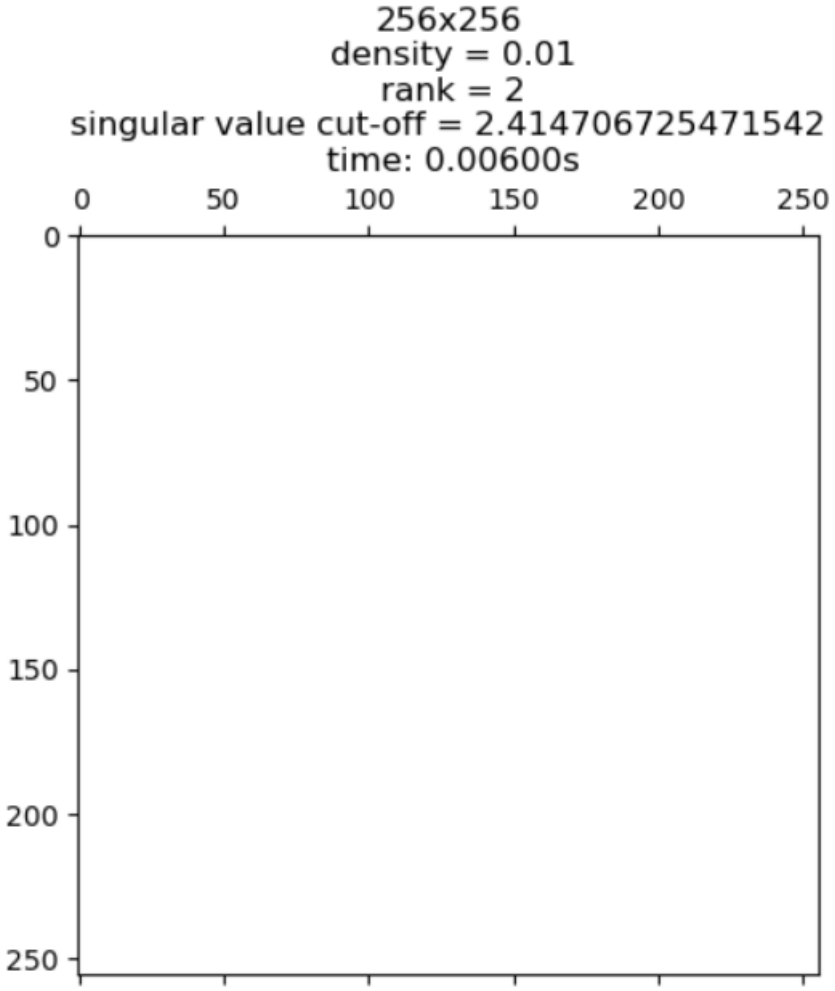


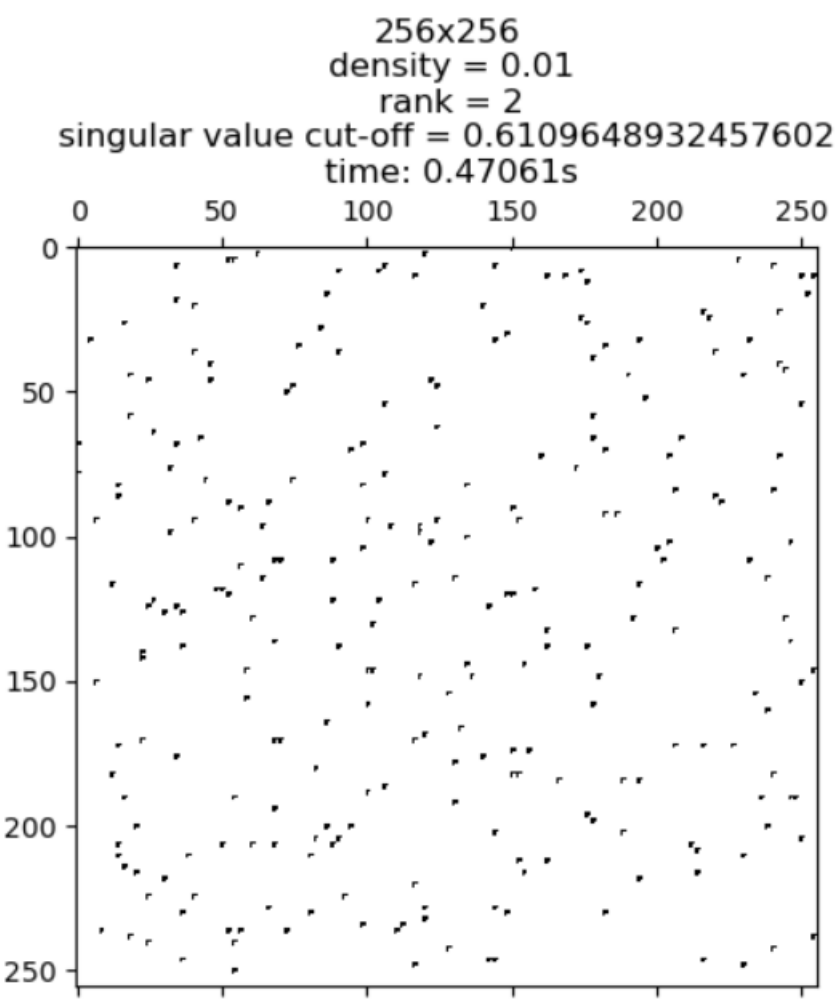

**Wyniki pomiarów czasów**

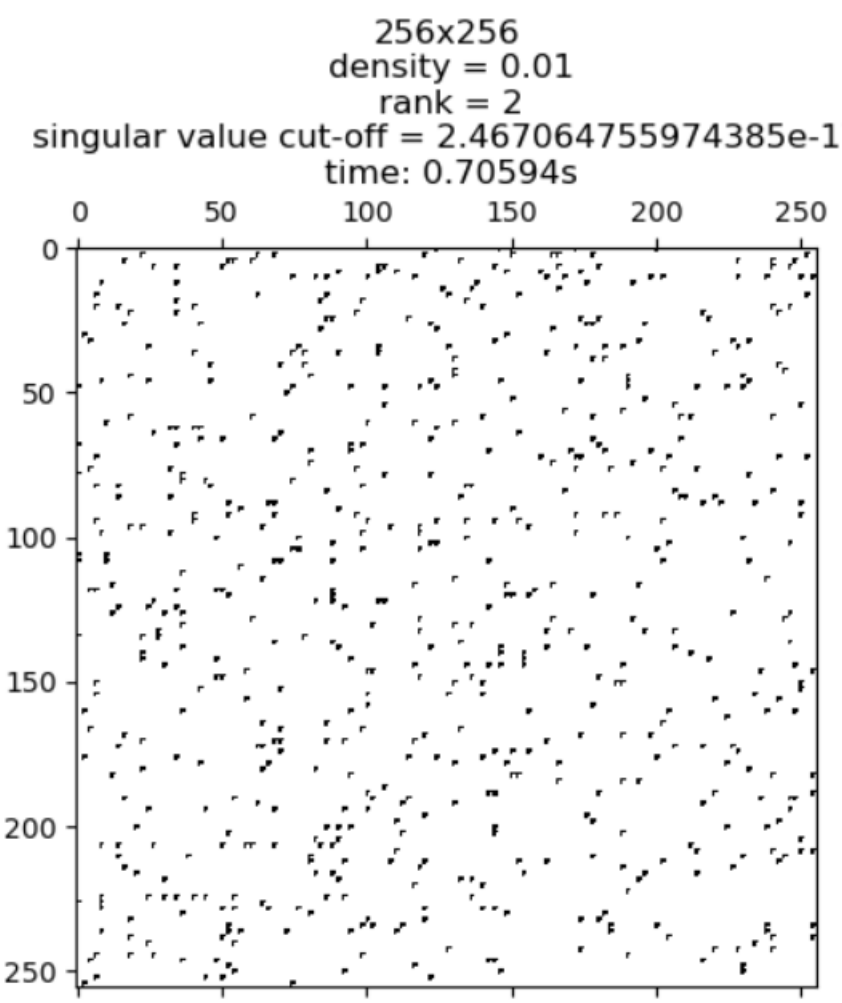
```
0.01 256 2 2.414706725471542 0.00600123405456543
0.01 256 2 0.6109648932457602 0.4706137180328369
0.01 256 2 2.467064755974385e-17 0.7059440612792969
0.01 256 4 2.414706725471542 0.0050008296966552734
0.01 256 4 0.6109648932457602 0.3907475471496582
0.01 256 4 2.467064755974385e-17 0.5111534595489502
0.01 256 8 2.414706725471542 0.00400090217590332
0.01 256 8 0.6109648932457602 0.2945842742919922
0.01 256 8 2.467064755974385e-17 0.32277560234069824
0.01 512 2 3.472690369885756 0.005001068115234375
0.01 512 2 0.9816834807422946 0.6991572380065918
0.01 512 2 1.9752413022543685e-16 3.0765061378479004
0.01 512 4 3.472690369885756 0.010055780410766602
0.01 512 4 0.9816834807422946 0.6832485198974609
0.01 512 4 1.9752413022543685e-16 2.1520371437072754
0.01 512 8 3.472690369885756 0.012002706527709961
0.01 512 8 0.9816834807422946 0.6316616535186768
0.01 512 8 1.9752413022543685e-16 1.316622257232666
0.01 1024 2 5.882121934977116 0.01227879524230957
0.01 1024 2 1.4098876561276805 0.7756803035736084
```

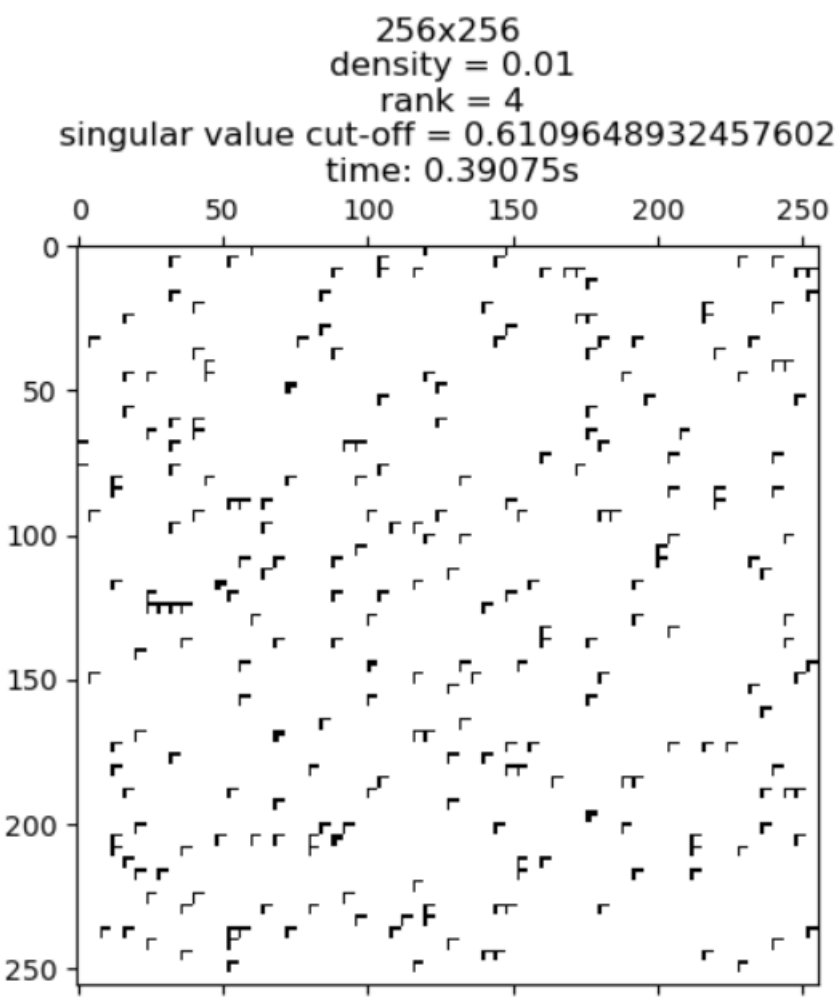
**Wykresy**

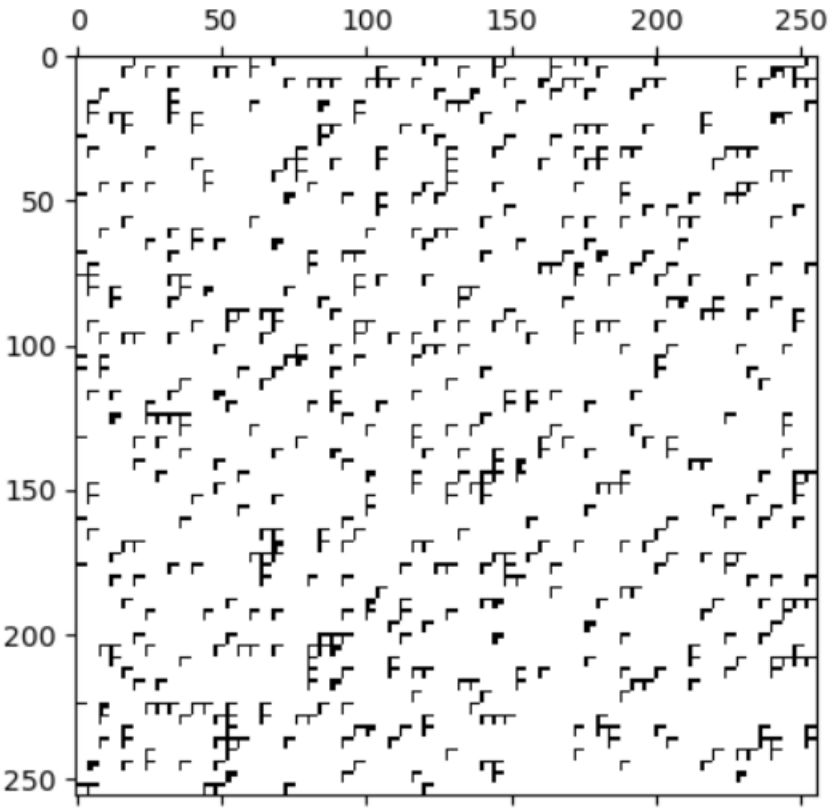
Dla gęstości {1%, 5%, 10%, 20%}, dla rozmiarów macierzy {2^8, …, 2^11}, dla b {2, 4, 8} dla $\delta \{\sigma 2, \sigma 2^k, \sigma \frac{2^k}{2}\}$ zmierzyliśmy pomiary i wyniki zaprezentowaliśmy na wykresach. W celu poprawienia czytelności sprawozdania celowo pominęliśmy „puste” wykresy, na przykład:

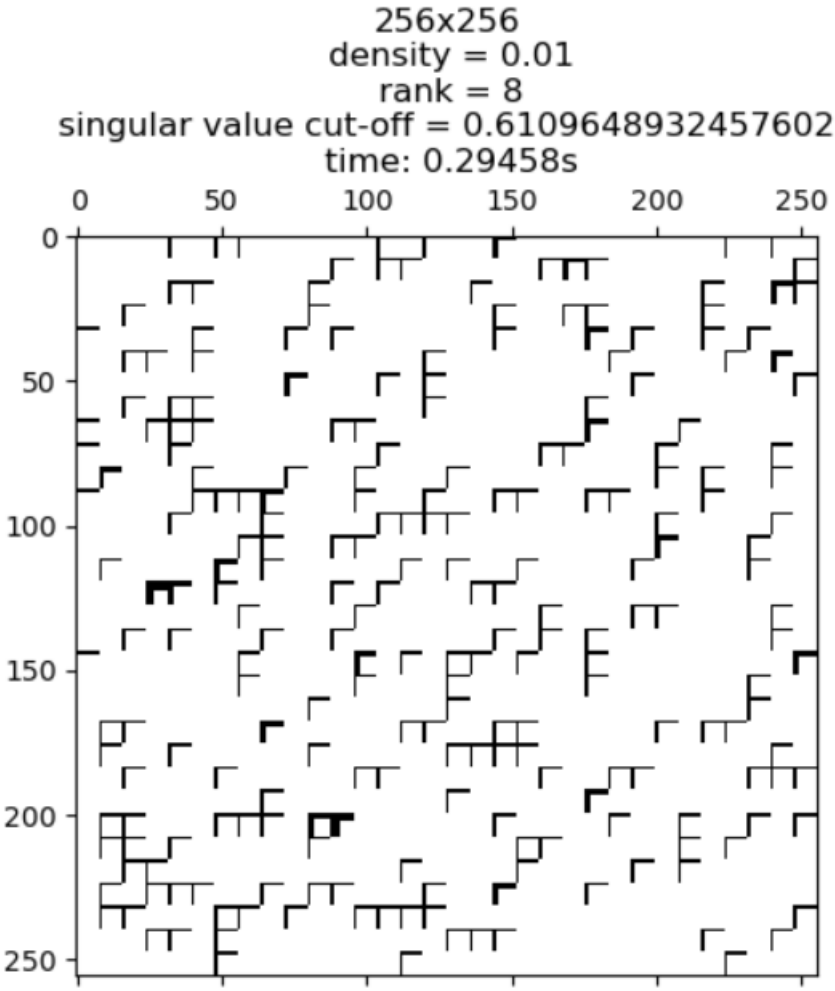


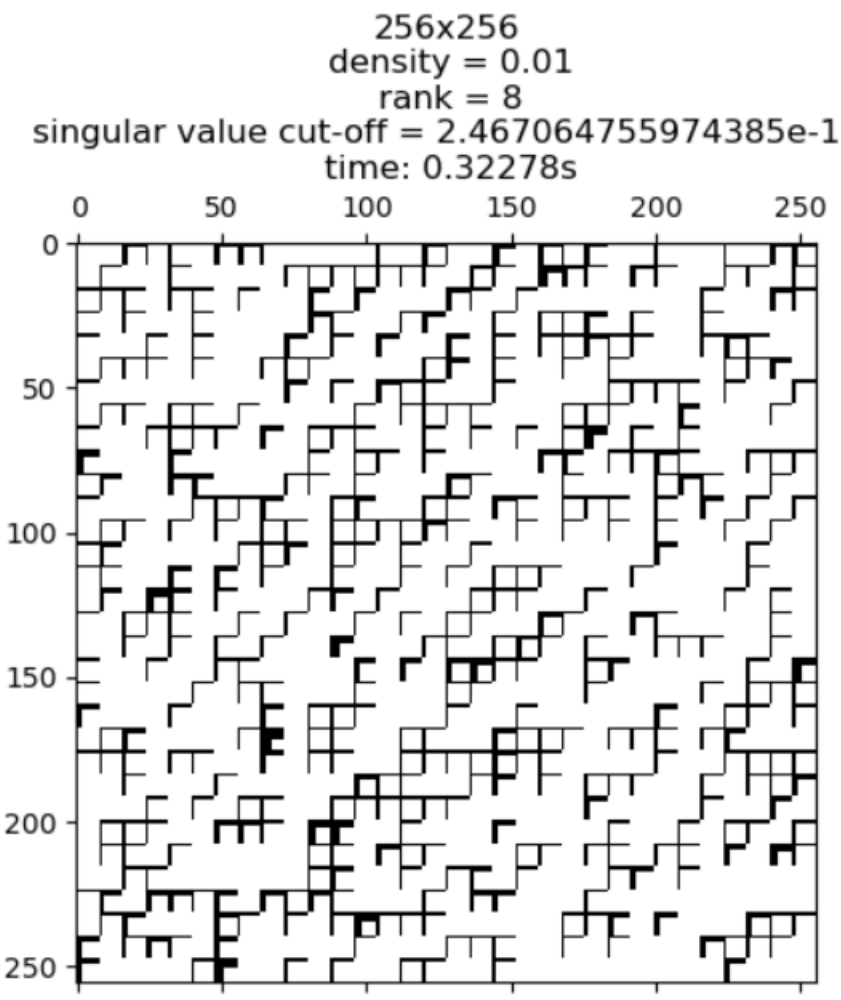

256x256
density = 0.01
rank = 2
singular value cut-off = 0.6109648932457602
time: 0.47061s
0
50
100
150
200
250
0
50
100
150
200
250

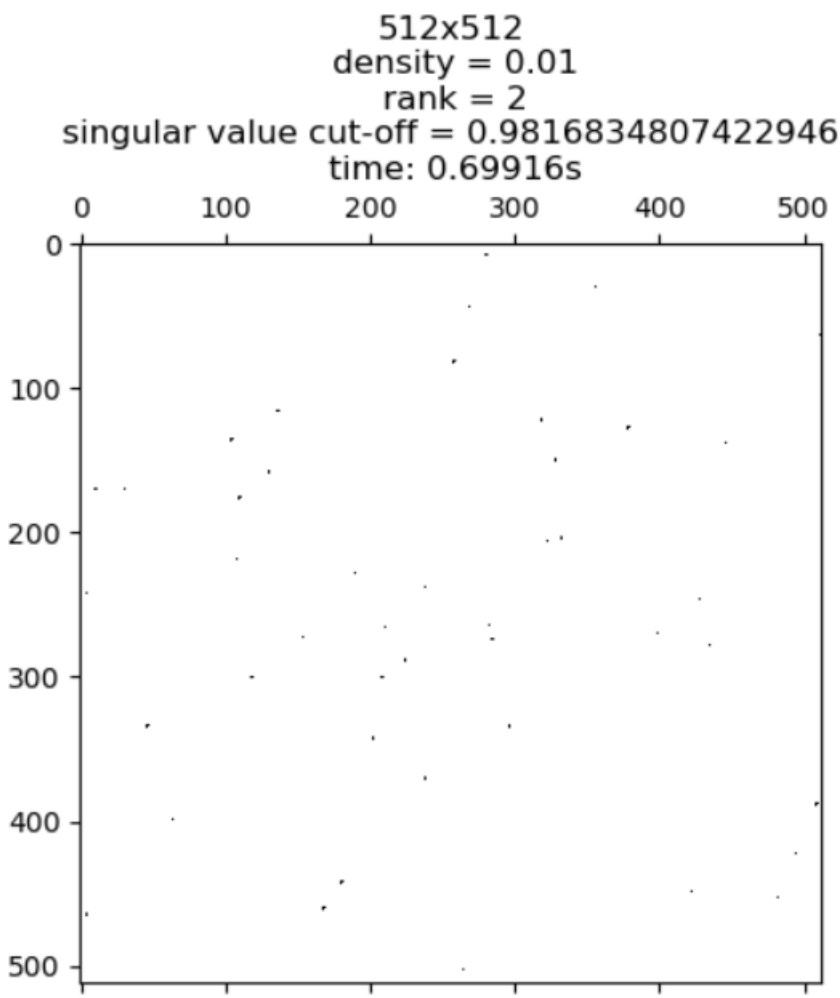

256x256
density = 0.01
rank = 2
singular value cut-off = 2.467064755974385e-17
time: 0.70594s
0
50
100
150
200
250

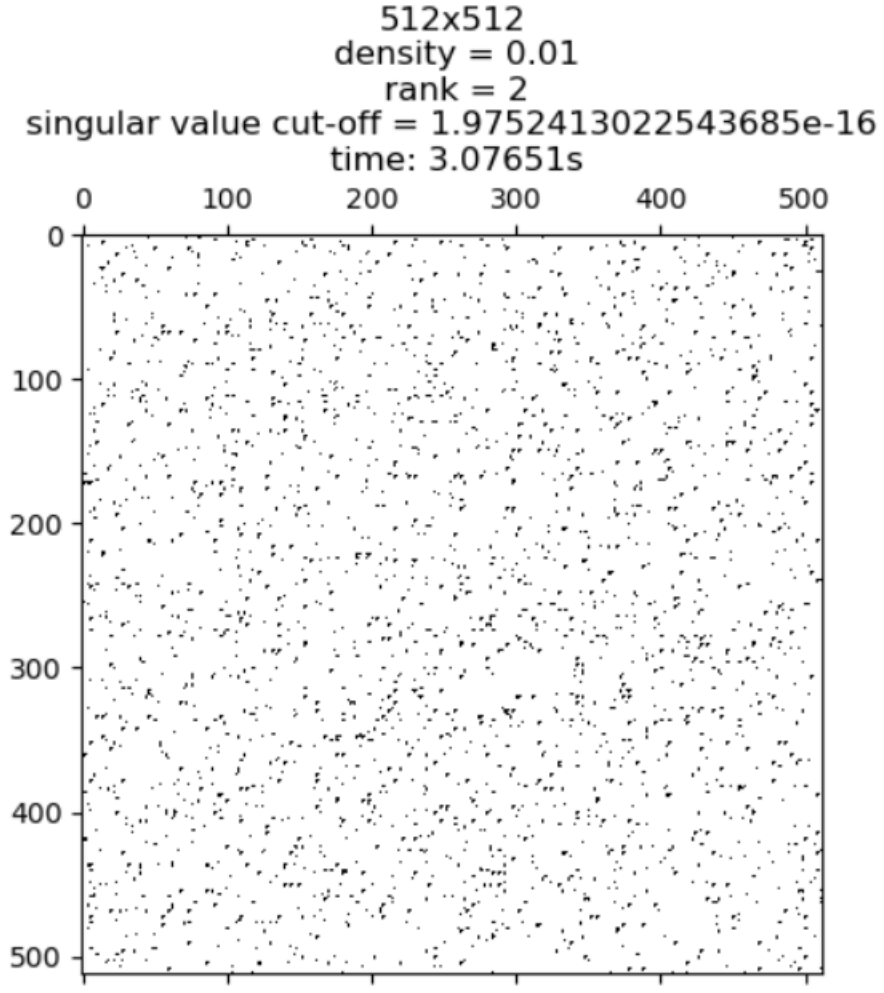

256x256
density = 0.01
rank = 4
singular value cut-off = 0.6109648932457602
time: 0.39075s
0
50
100
150
200
250

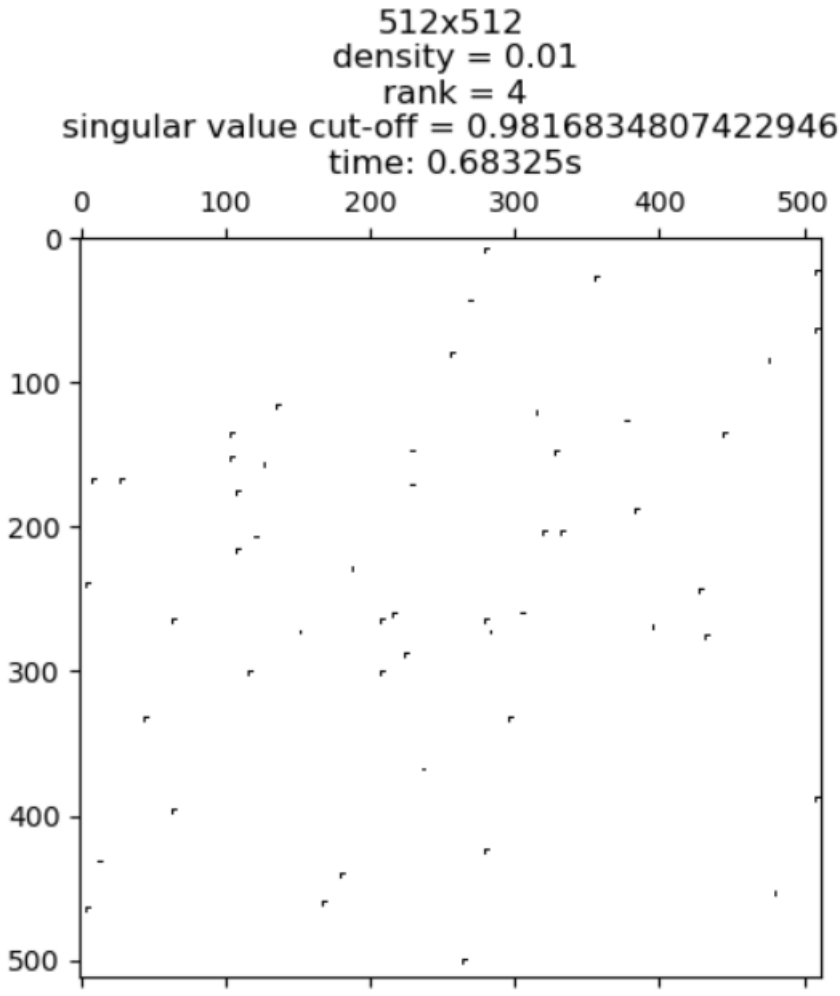

256x256
density = 0.01
rank = 4
singular value cut-off = 2.467064755974385e-17
time: 0.51115s
0
50
100
150
200
250

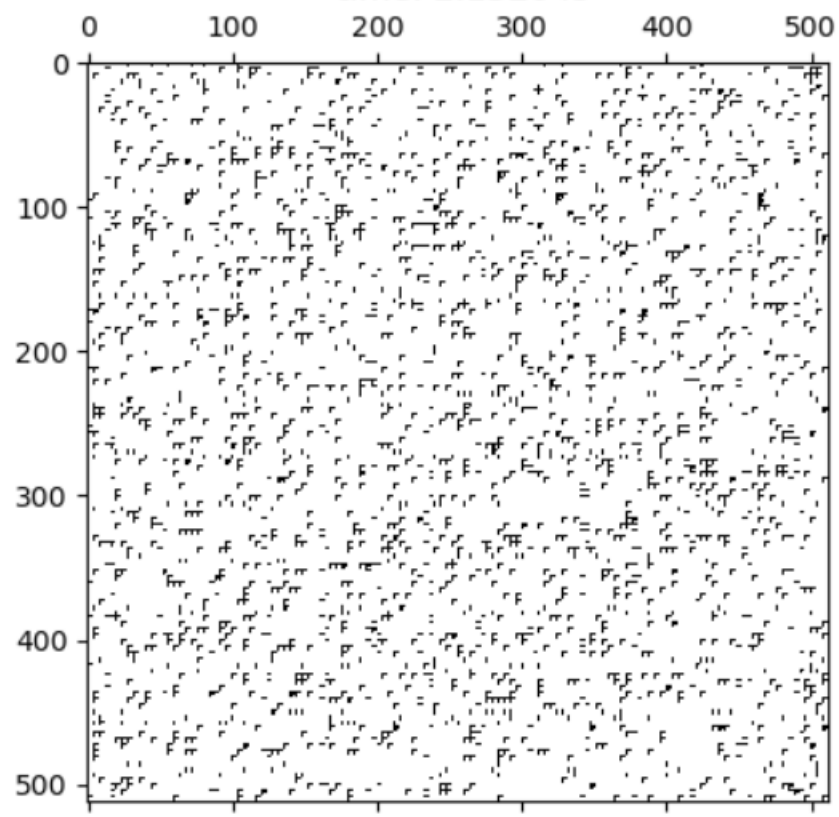

256x256
density = 0.01
rank = 8
singular value cut-off = 0.6109648932457602
time: 0.29458s
0
50
100
150
200
250

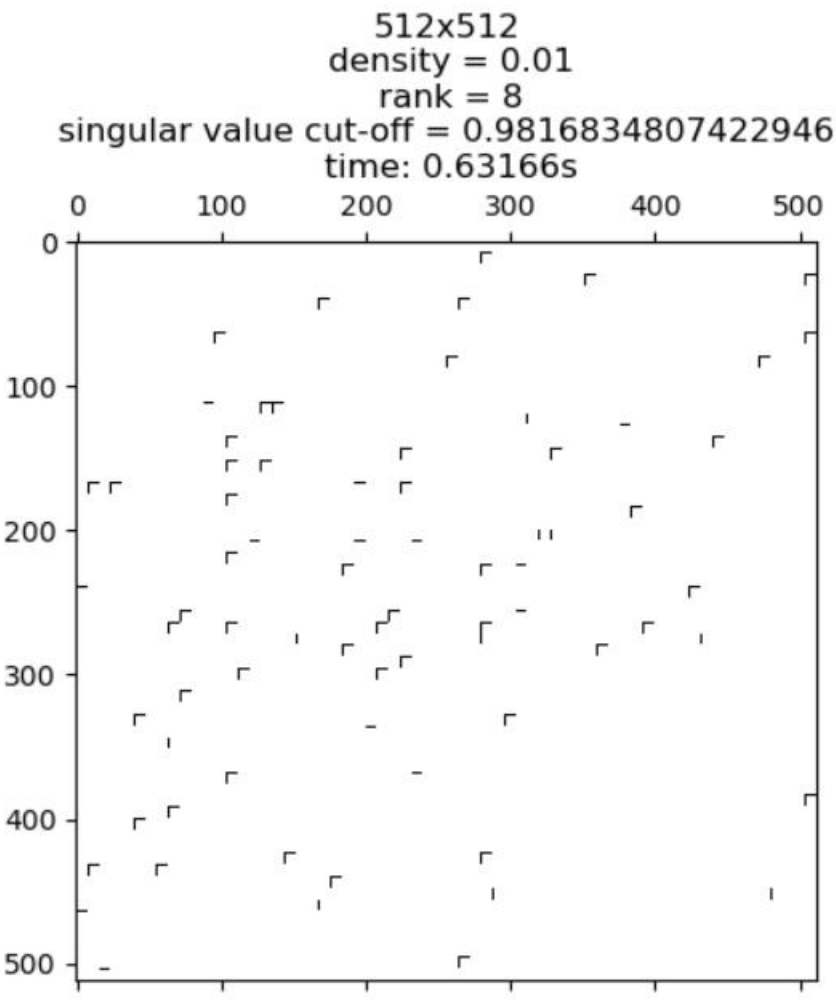

256x256
density = 0.01
rank = 8
singular value cut-off = 2.467064755974385e-17
time: 0.32278s
0
50
100
150
200
250

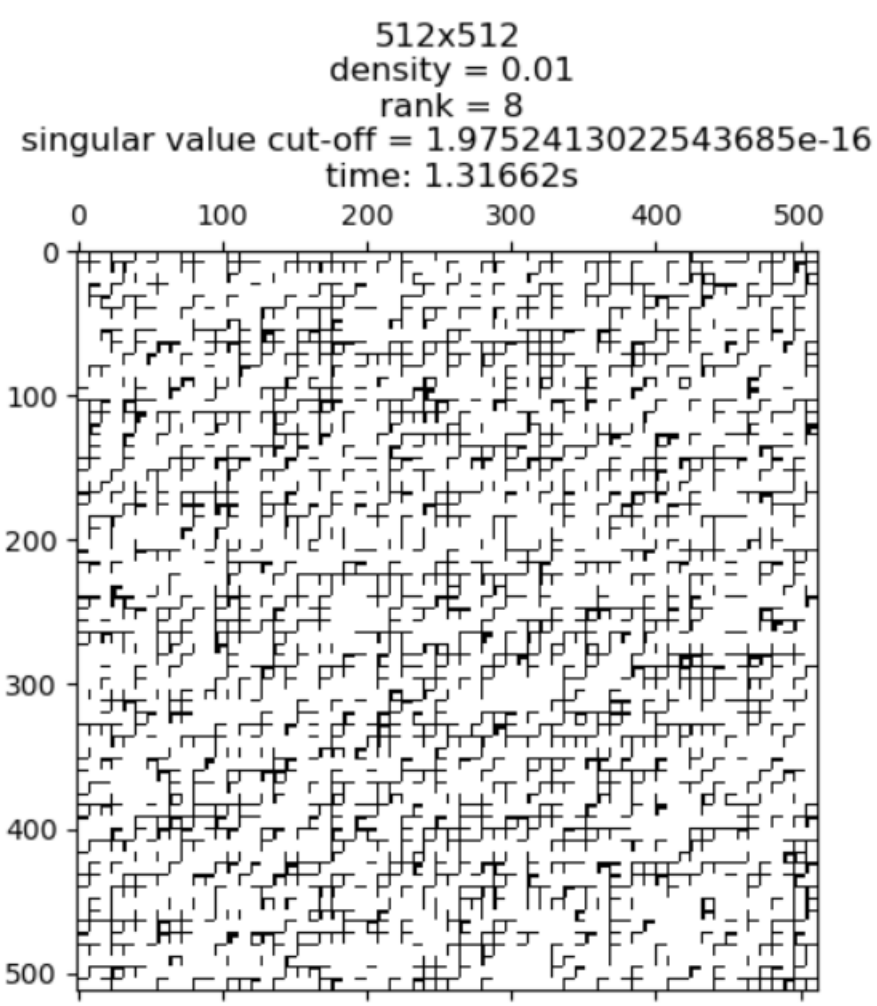

512x512
density = 0.01
rank = 2
singular value cut-off = 0.9816834807422946
time: 0.69916s
0
100
200
300
400
500
0
100
200
300
400
500

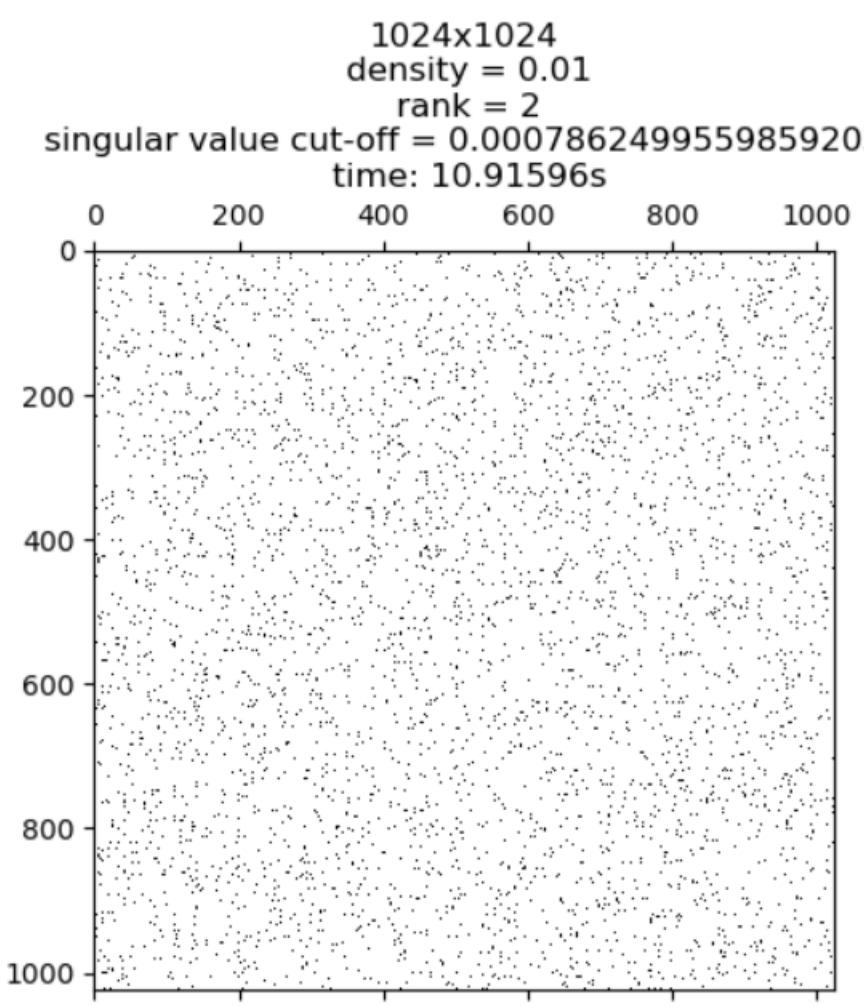

512x512
density = 0.01
rank = 2
singular value cut-off = 1.9752413022543685e-16
time: 3.07651s
0
100
200
300
400
500
0
100
200
300
400
500

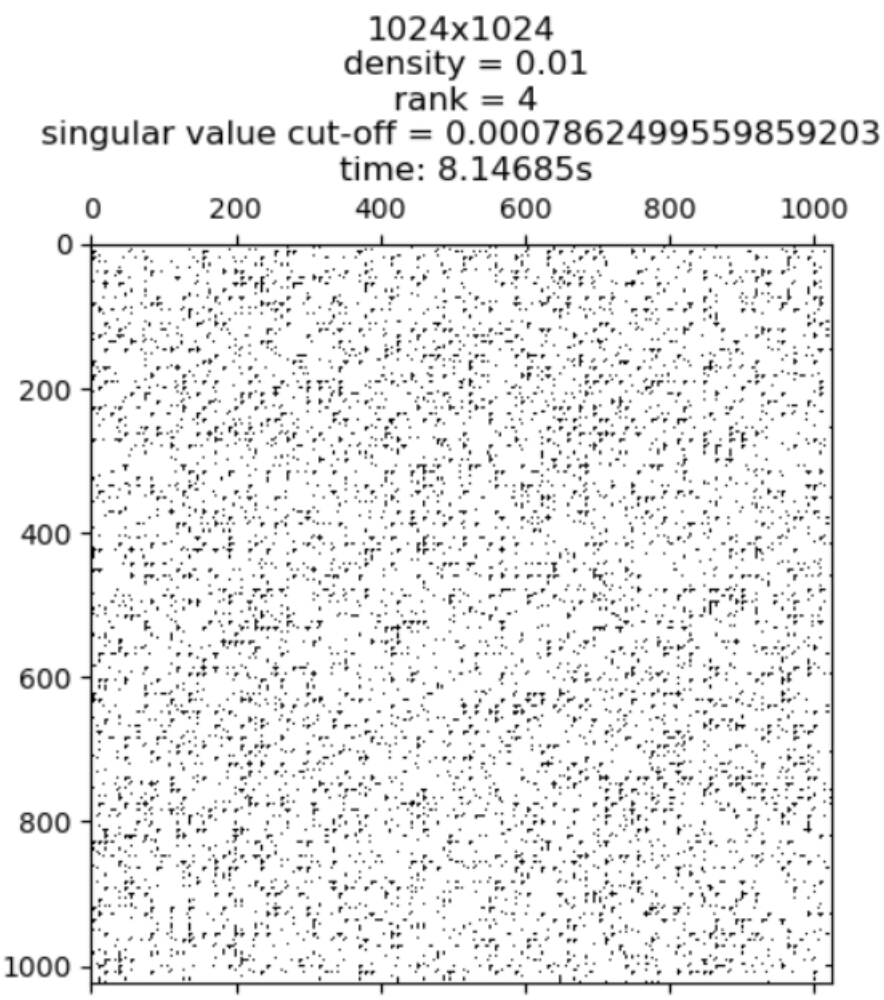

512x512
density = 0.01
rank = 4
singular value cut-off = 0.9816834807422946
time: 0.68325s
0
100
200
300
400
500
0
100
200
300
400
500

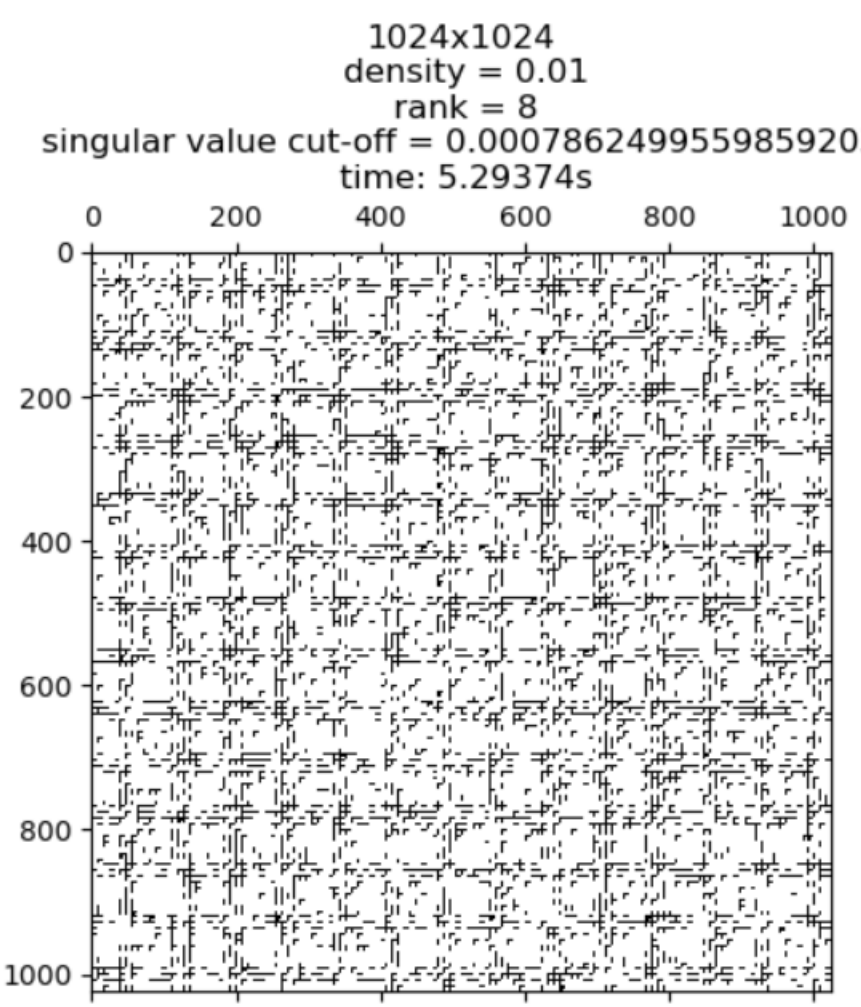

512x512
density = 0.01
rank = 4
singular value cut-off = 1.9752413022543685e-16
time: 2.15204s
0
100
200
300
400
500
0
100
200
300
400
500

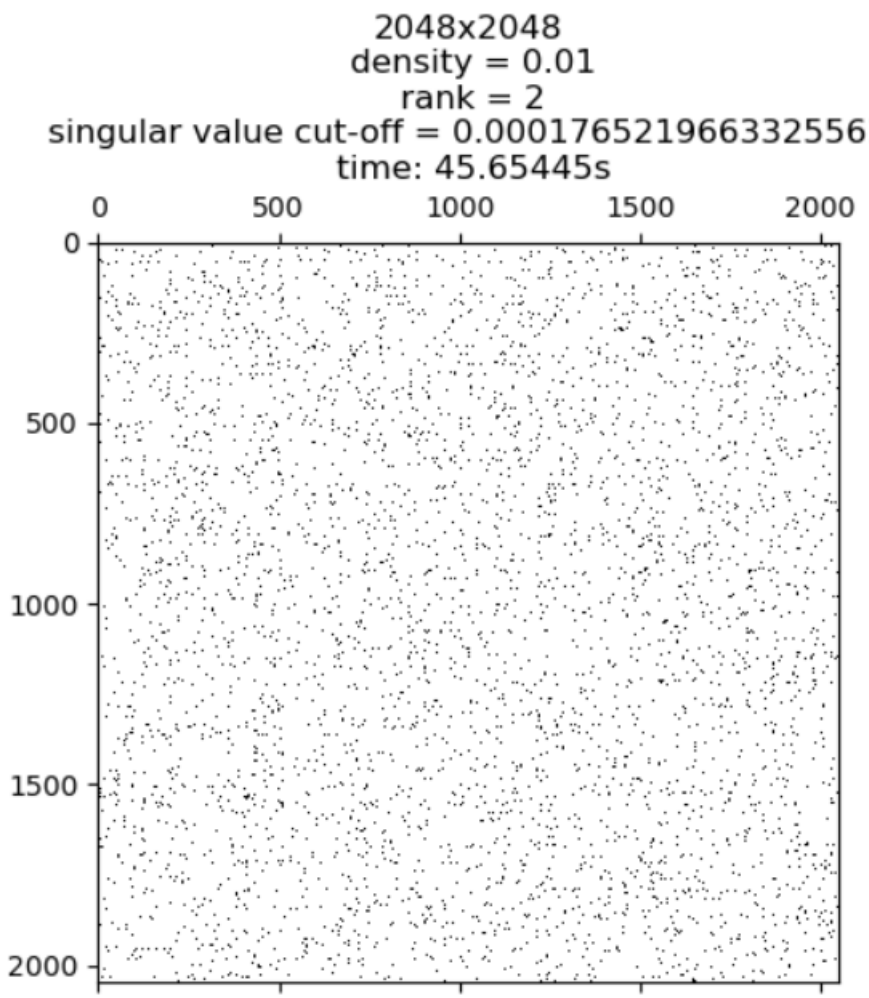

512x512
density = 0.01
rank = 8
singular value cut-off = 0.9816834807422946
time: 0.63166s
0
100
200
300
400
500

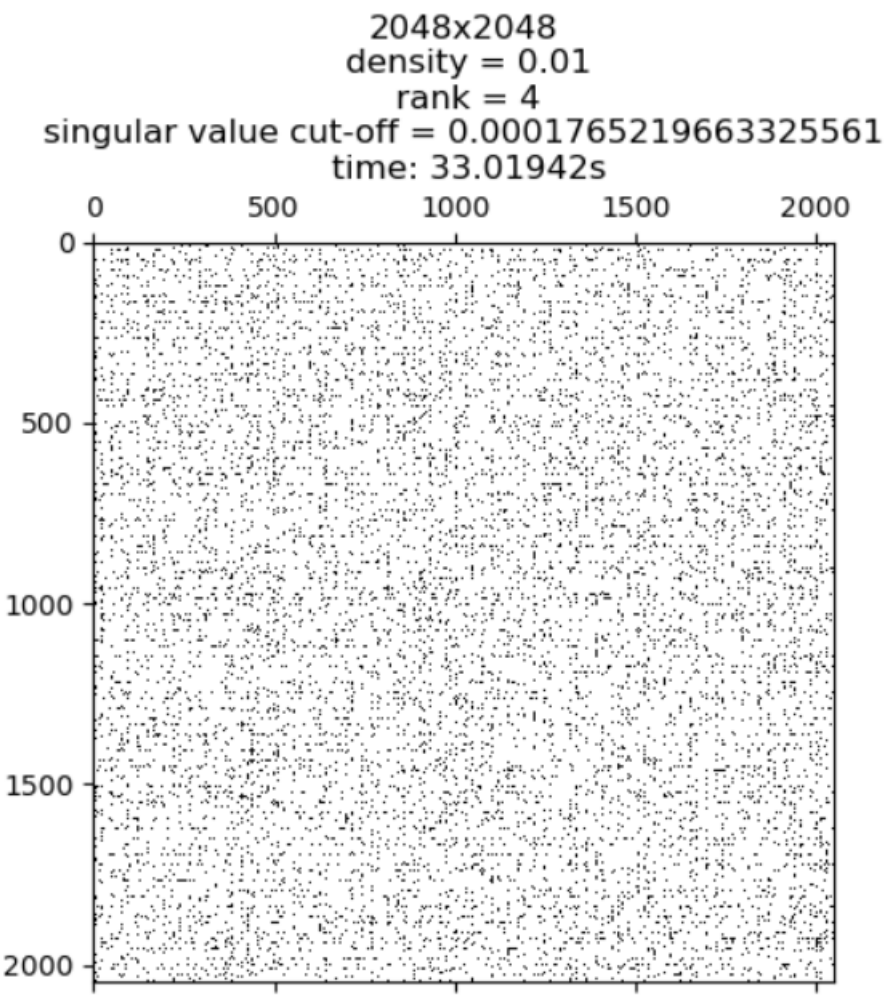

512x512
density = 0.01
rank = 8
singular value cut-off = 1.9752413022543685e-16
time: 1.31662s
0
100
200
300
400
500

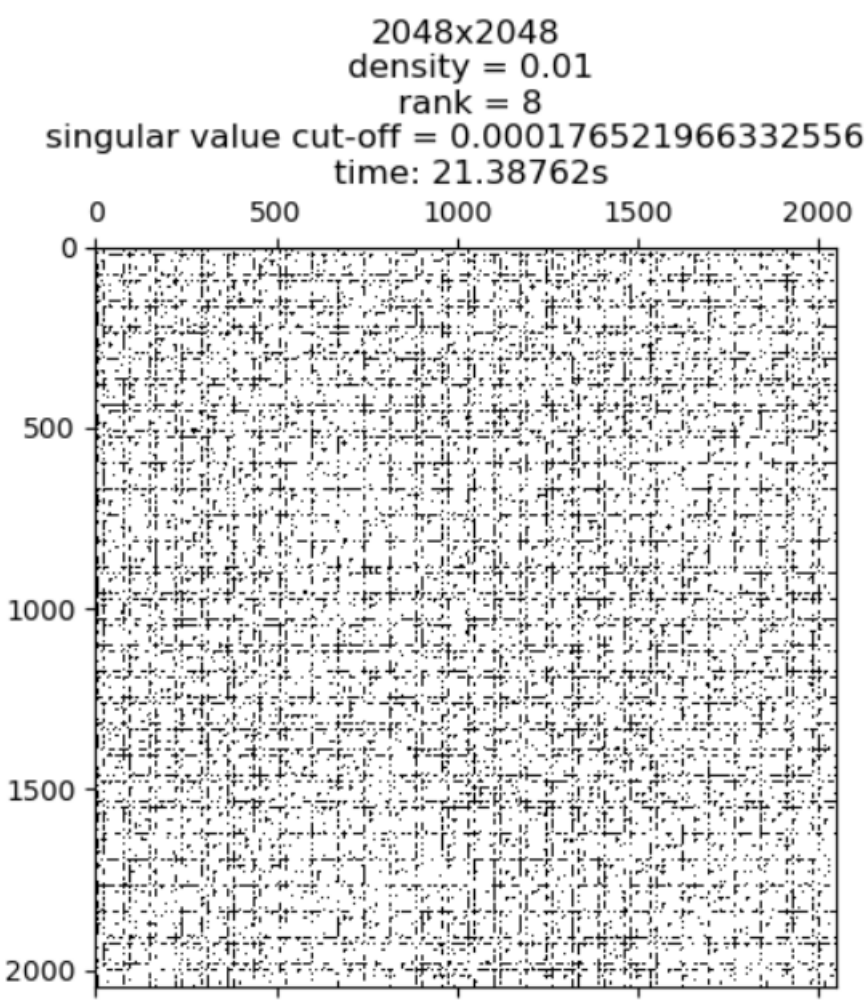

1024x1024
density = 0.01
rank = 2
singular value cut-off = 0.0007862499559859203
time: 10.91596s
0
200
400
600
800
1000
0
200
400
600
800
1000

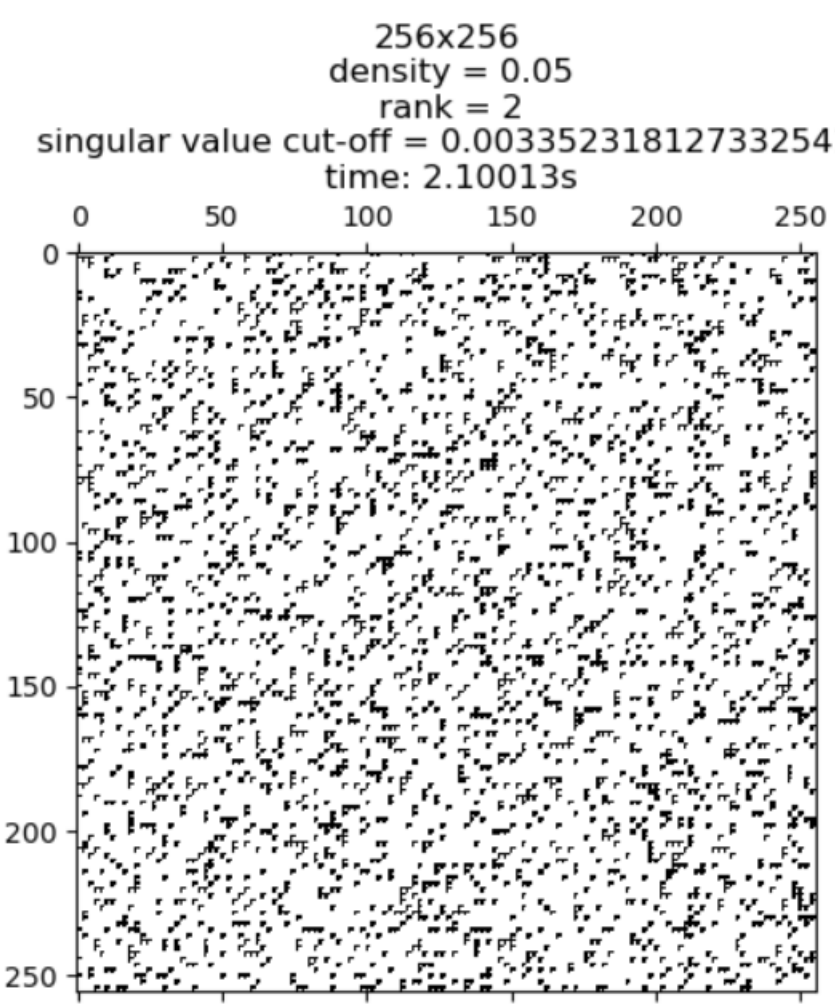

1024x1024
density = 0.01
rank = 4
singular value cut-off = 0.0007862499559859203
time: 8.14685s
0
200
400
600
800
1000
0
200
400
600
800
1000

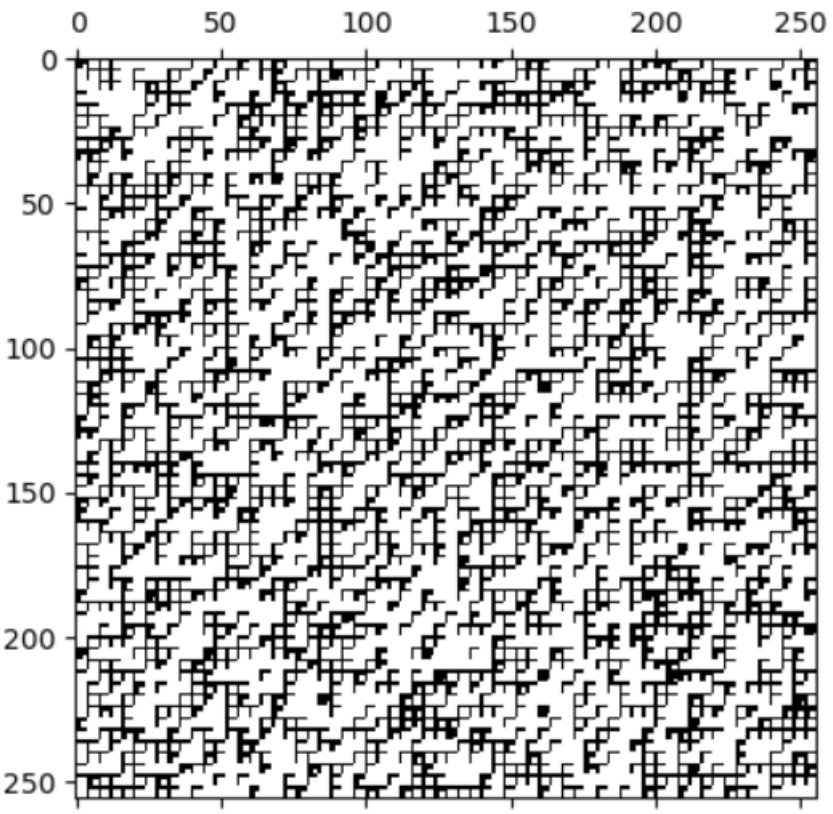

1024x1024
density = 0.01
rank = 8
singular value cut-off = 0.0007862499559859203
time: 5.29374s
0
200
400
600
800
1000

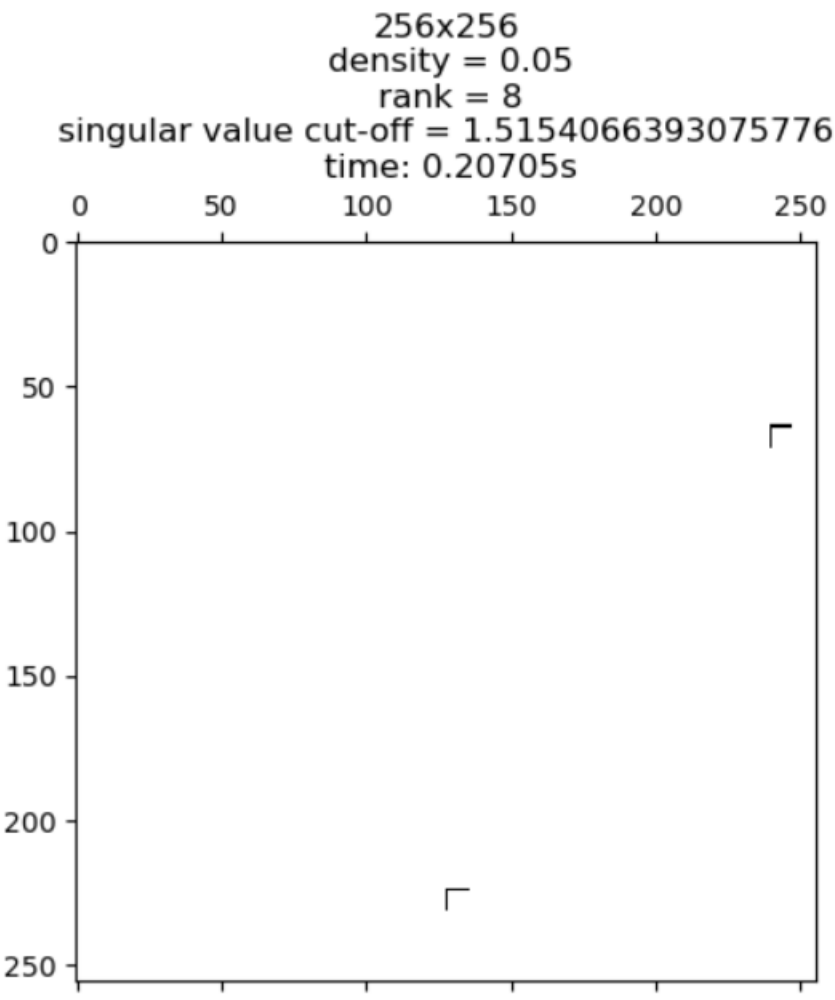

2048x2048
density = 0.01
rank = 2
singular value cut-off = 0.0001765219663325561
time: 45.65445s
0
500
1000
1500
2000
0
500
1000
1500
2000

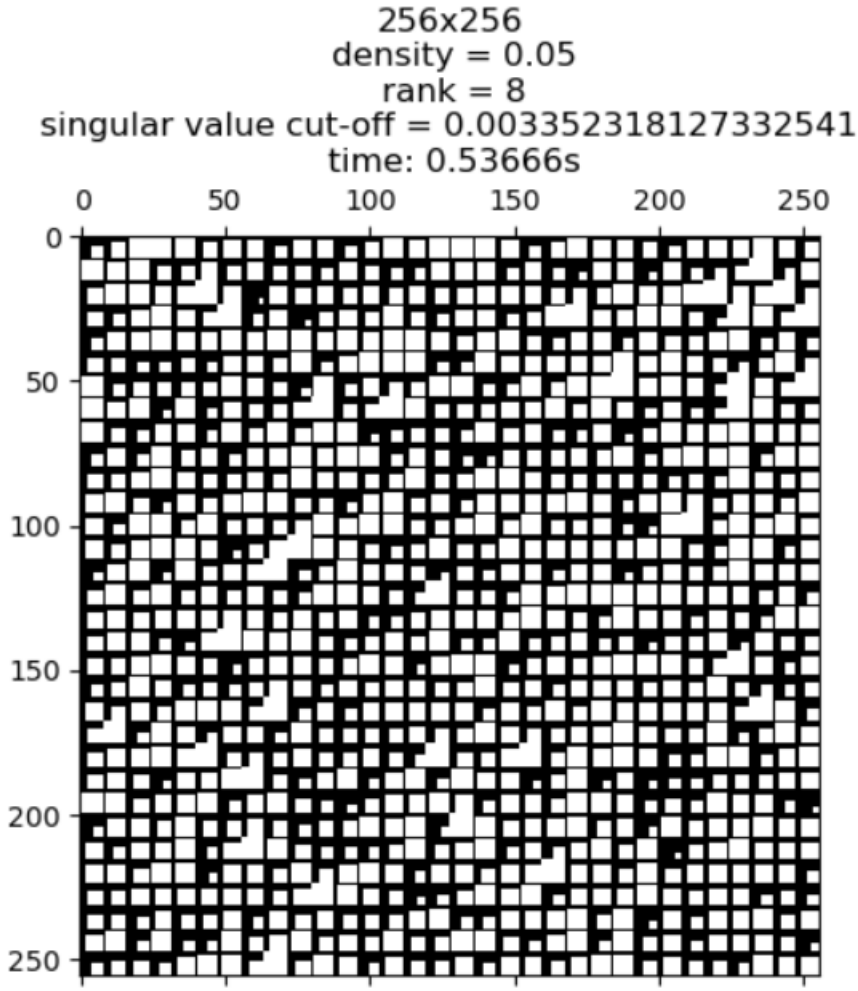

2048x2048
density = 0.01
rank = 4
singular value cut-off = 0.00017652196633255 61
time: 33.01942s
0
500
1000
1500
2000
0
500
1000
1500
2000

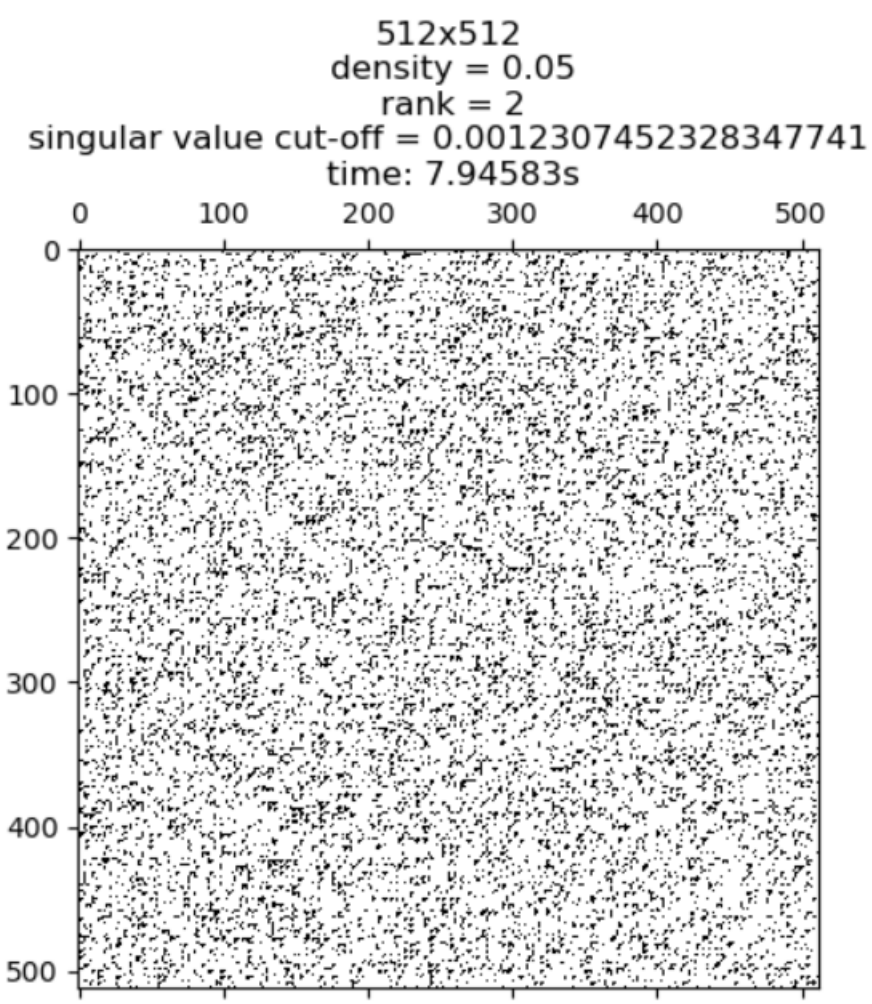

2048x2048
density = 0.01
rank = 8
singular value cut-off = 0.0001765219663325561
time: 21.38762s
0
500
1000
1500
2000
0
500
1000
1500
2000

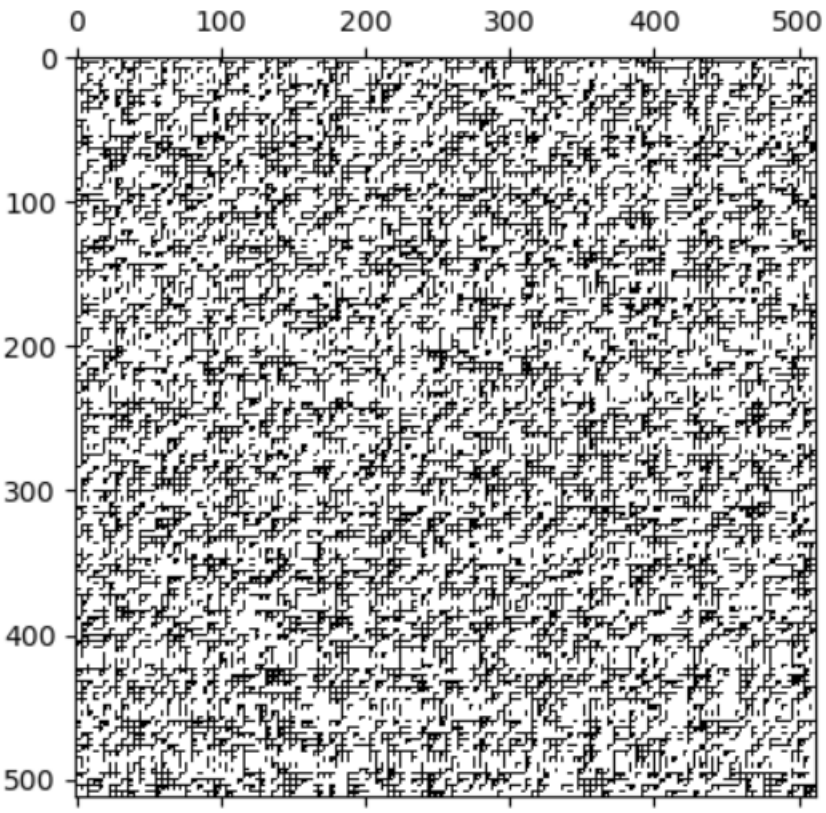

256x256
density = 0.05
rank = 2
singular value cut-off = 0.0033523181273325 41
time: 2.10013s
0
50
100
150
200
250
0
50
100
150
200
250

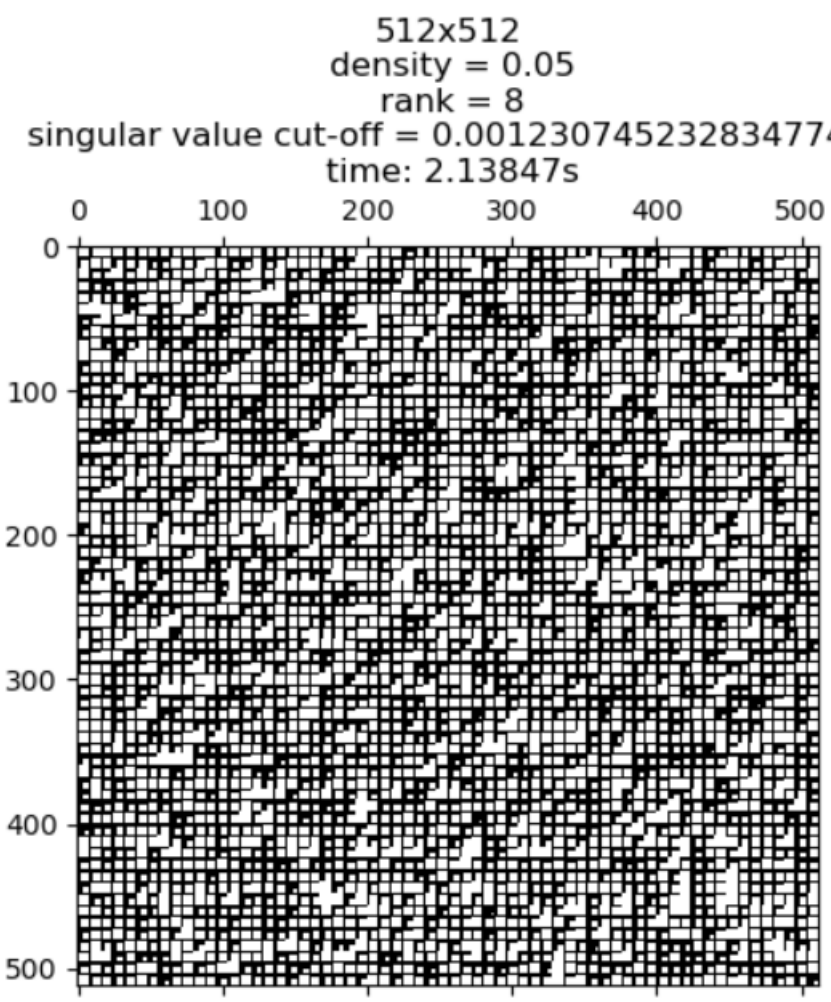

256x256
density = 0.05
rank = 4
singular value cut-off = 0.0033523181273325 41
time: 1.16442s
0
50
100
150
200
250
0
50
100
150
200
250

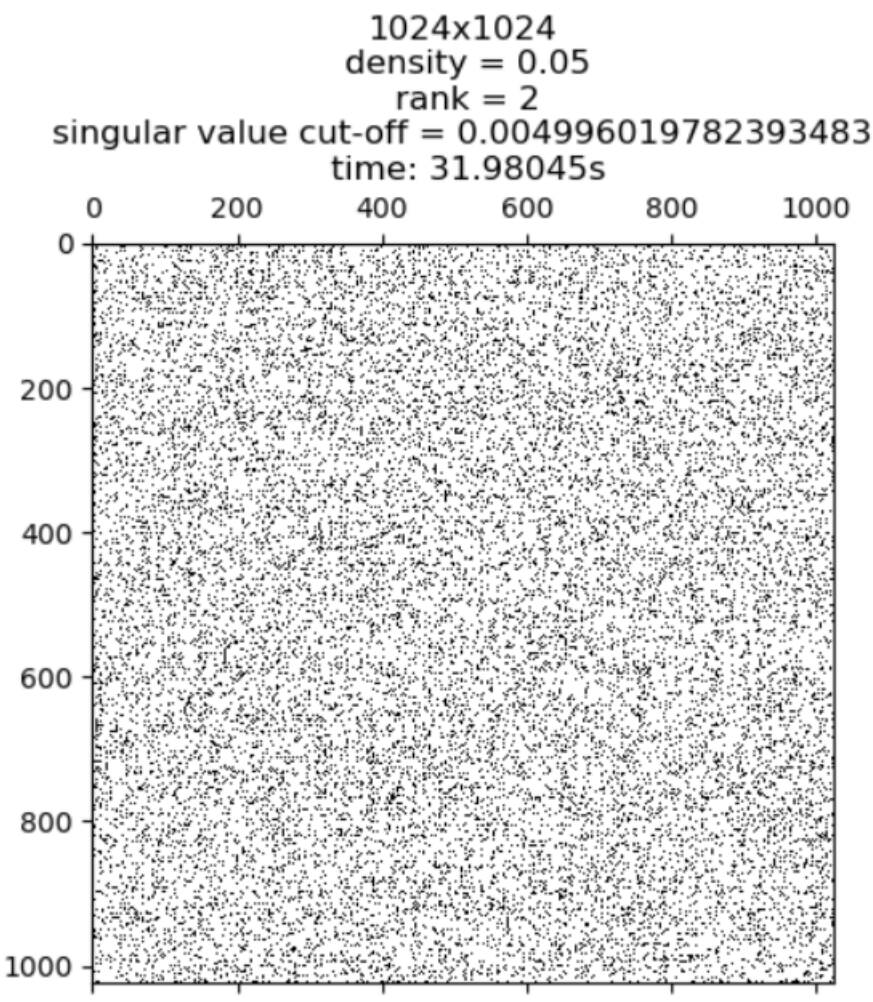

256x256
density = 0.05
rank = 8
singular value cut-off = 1.5154066393075776
time: 0.20705s
0
50
100
150
200
250
0
50
100
150
200
250

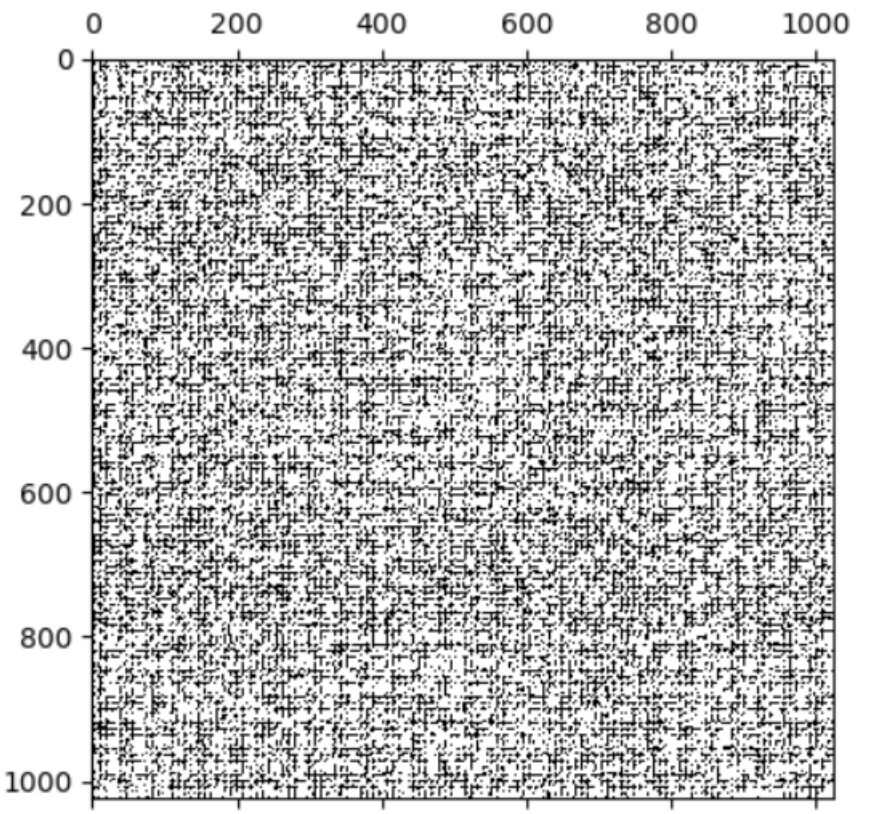

256x256
density = 0.05
rank = 8
singular value cut-off = 0.0033523181273325 41
time: 0.53666s
0
50
100
150
200
250
0
50
100
150
200
250

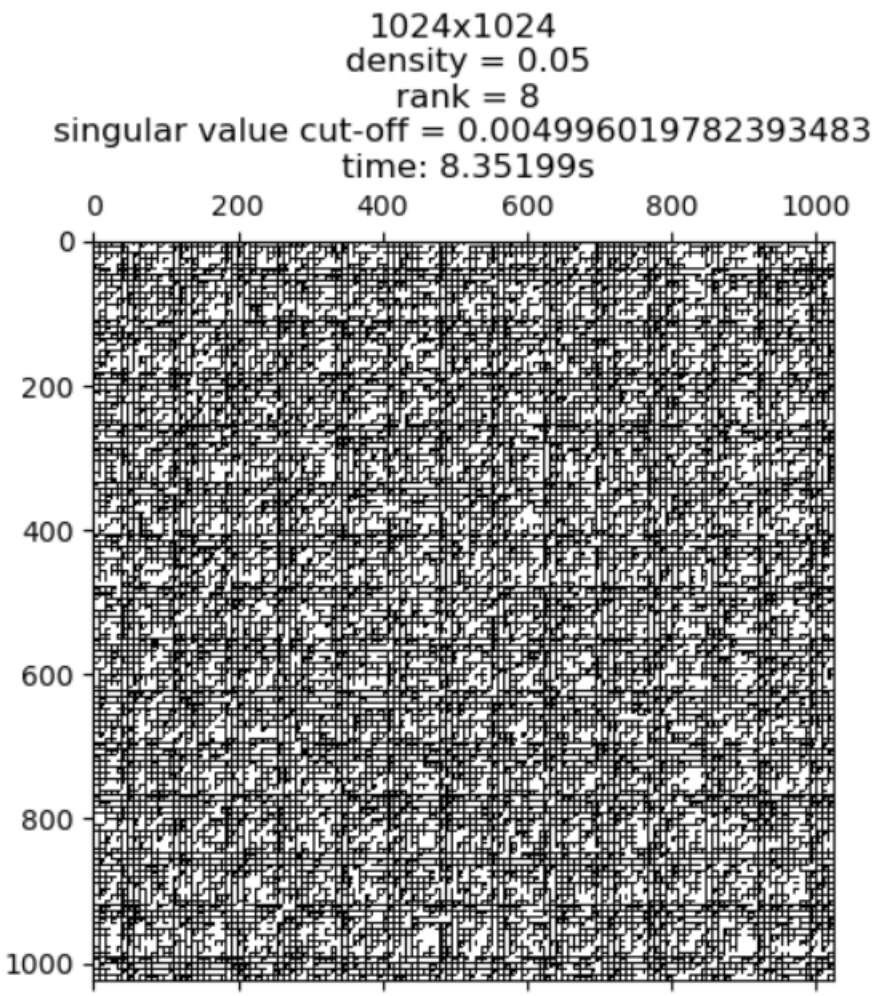

512x512
density = 0.05
rank = 2
singular value cut-off = 0.0012307452328347741
time: 7.94583s
0
100
200
300
400
500
0
100
200
300
400
500

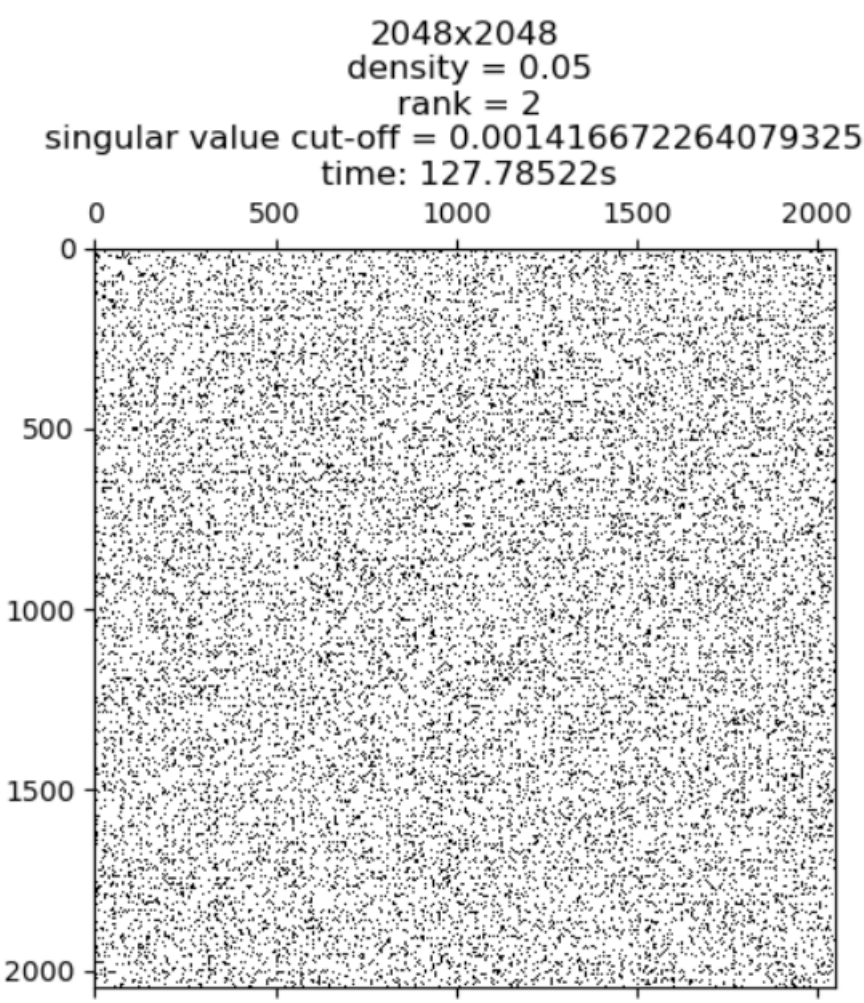

512x512
density = 0.05
rank = 4
singular value cut-off = 0.0012307452328347741
time: 4.72852s
0
100
200
300
400
500
0
100
200
300
400
500

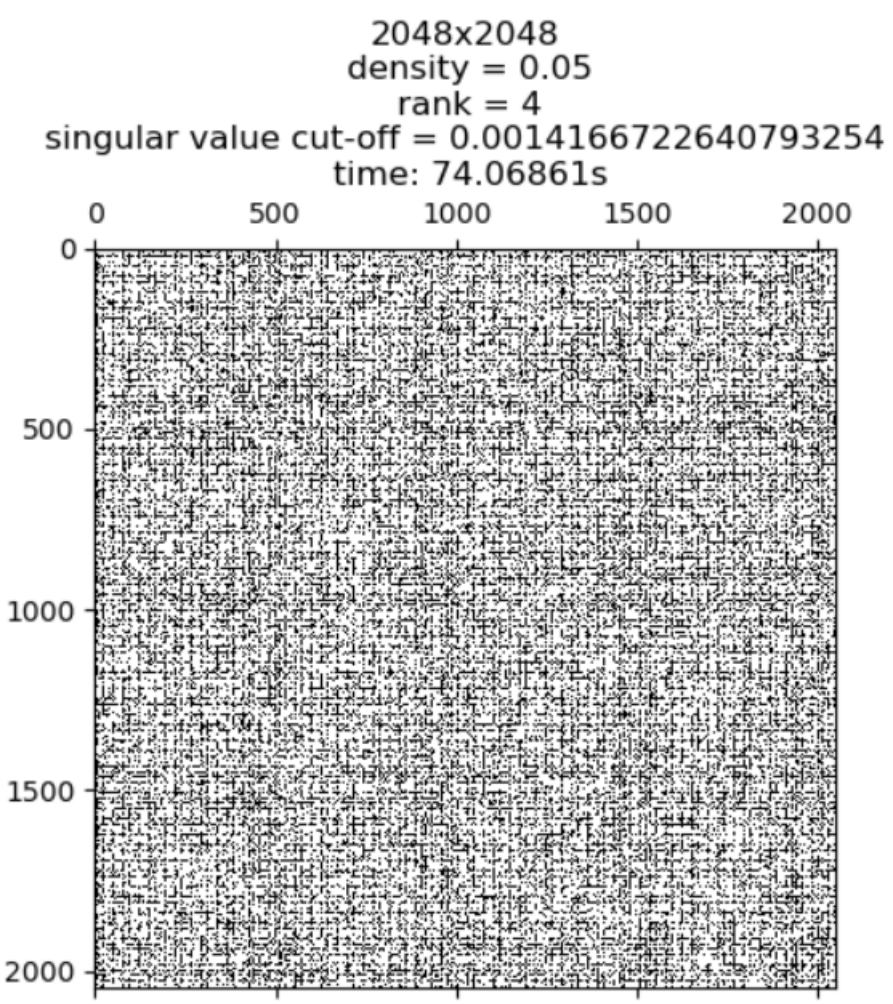

512x512
density = 0.05
rank = 8
singular value cut-off = 0.0012307452328347741
time: 2.13847s
0
100
200
300
400
500
0
100
200
300
400
500

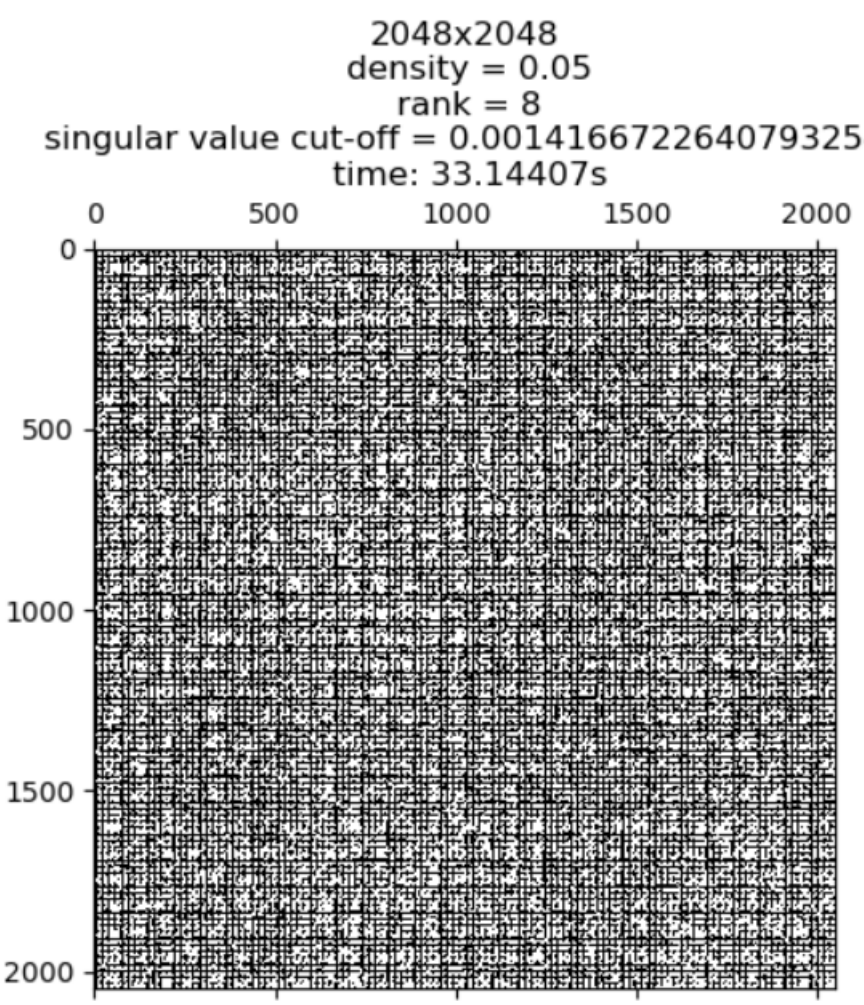

1024x1024
density = 0.05
rank = 2
singular value cut-off = 0.004996019782393483
time: 31.98045s
0
200
400
600
800
1000
0
200
400
600
800
1000

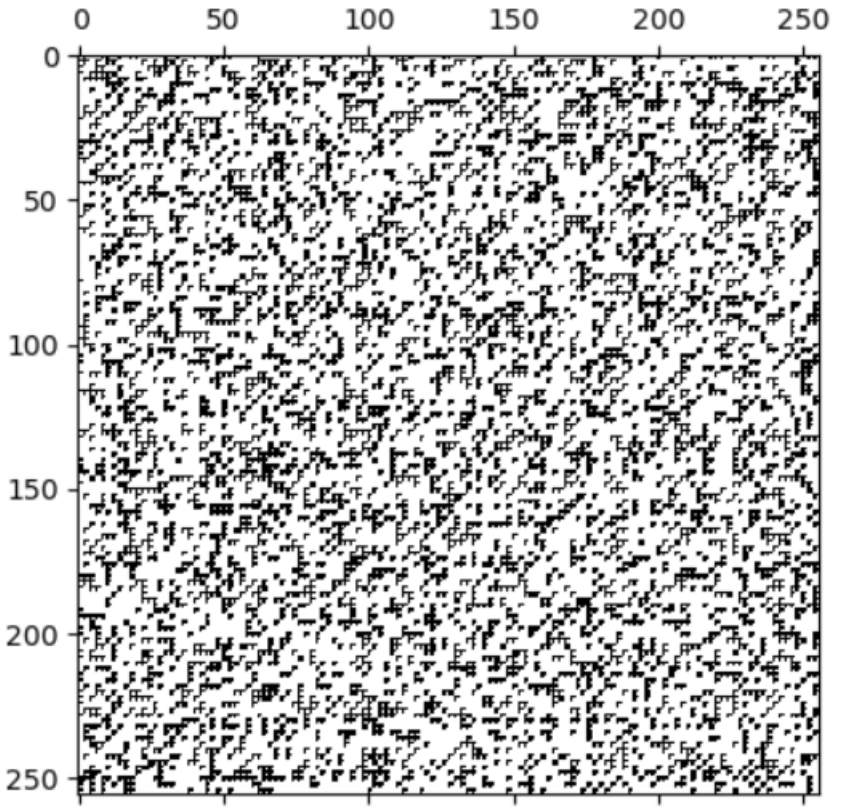

1024x1024
density = 0.05
rank = 4
singular value cut-off = 0.004996019782393483
time: 18.37380s
0
200
400
600
800
1000
0
200
400
600
800
1000

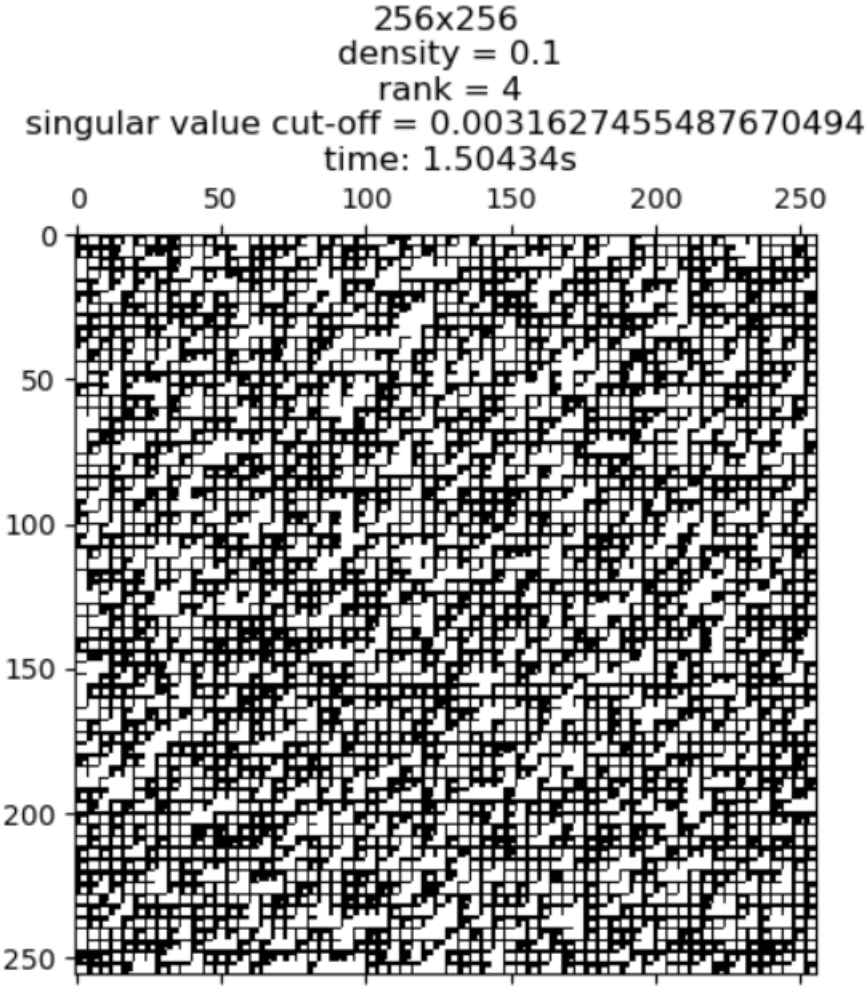

1024x1024
density = 0.05
rank = 8
singular value cut-off = 0.004996019782393483
time: 8.35199s
0
200
400
600
800
1000
0
200
400
600
800
1000

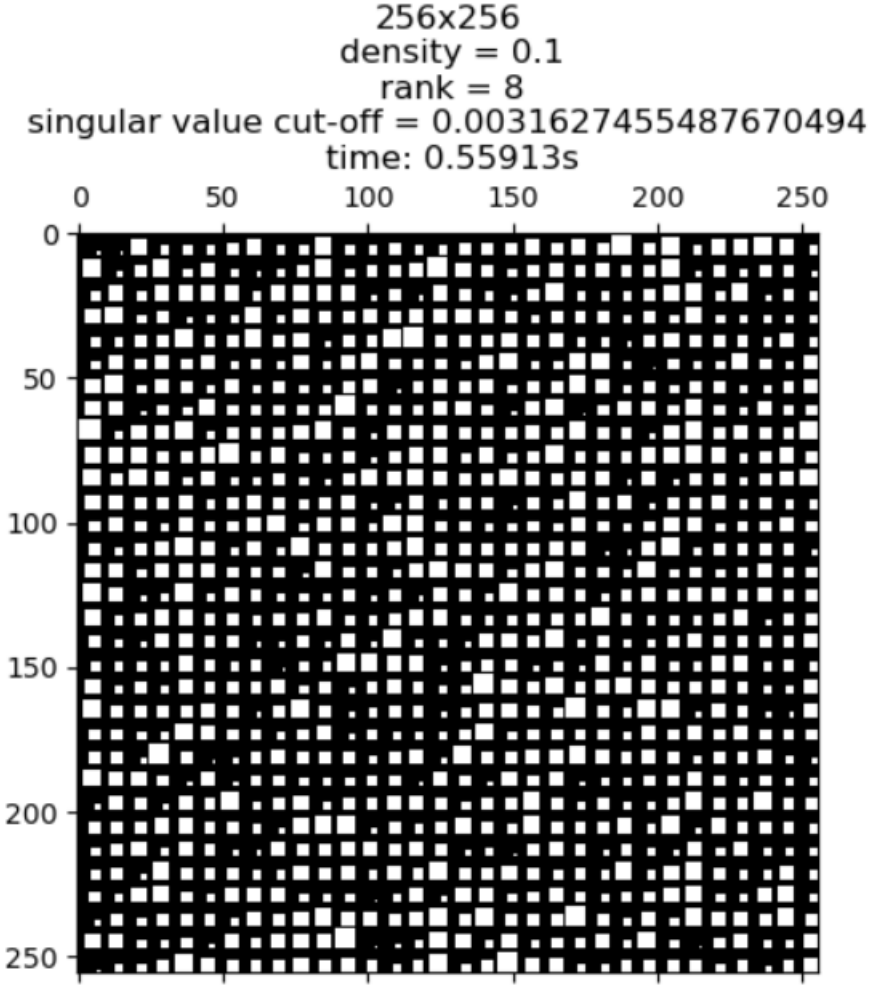

2048x2048
density = 0.05
rank = 2
singular value cut-off = 0.0014166722640793254
time: 127.78522s
0
500
1000
1500
2000
0
500
1000
1500
2000

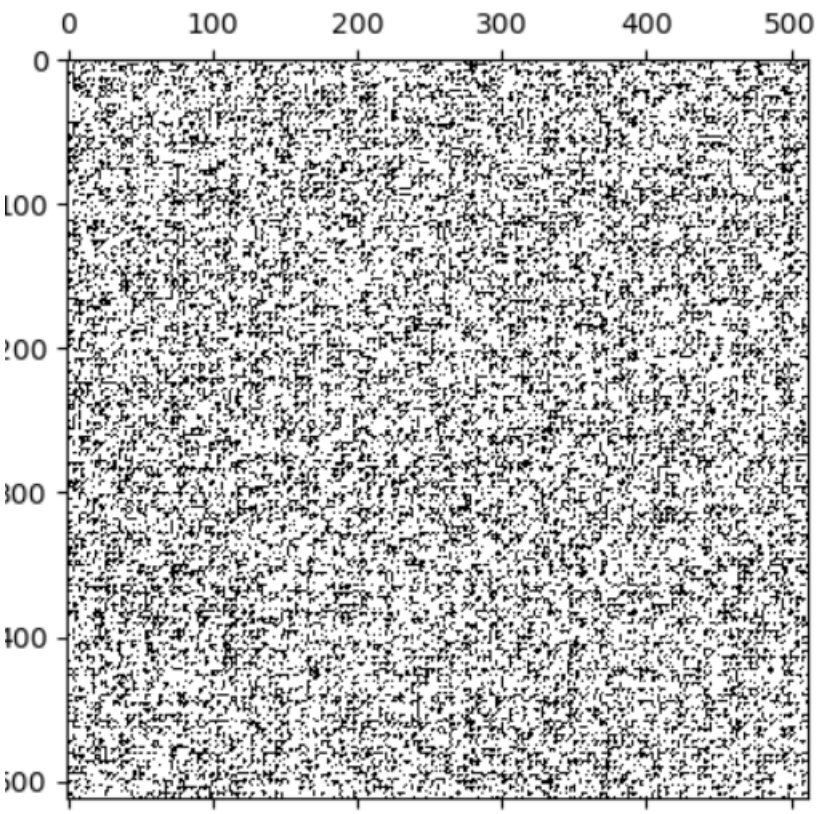

2048x2048
density = 0.05
rank = 4
singular value cut-off = 0.0014166722640793254
time: 74.06861s
0
500
1000
1500
2000
0
500
1000
1500
2000

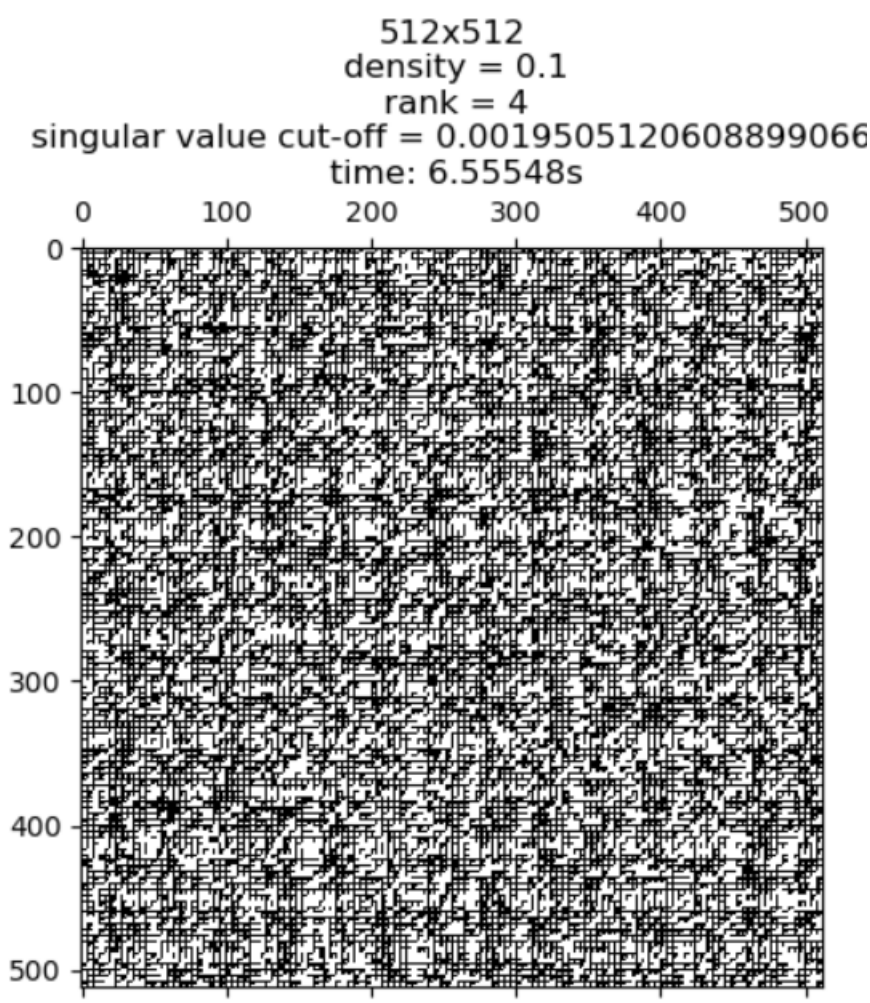

2048x2048
density = 0.05
rank = 8
singular value cut-off = 0.0014166722640793254
time: 33.14407s
0
500
1000
1500
2000
0
500
1000
1500
2000

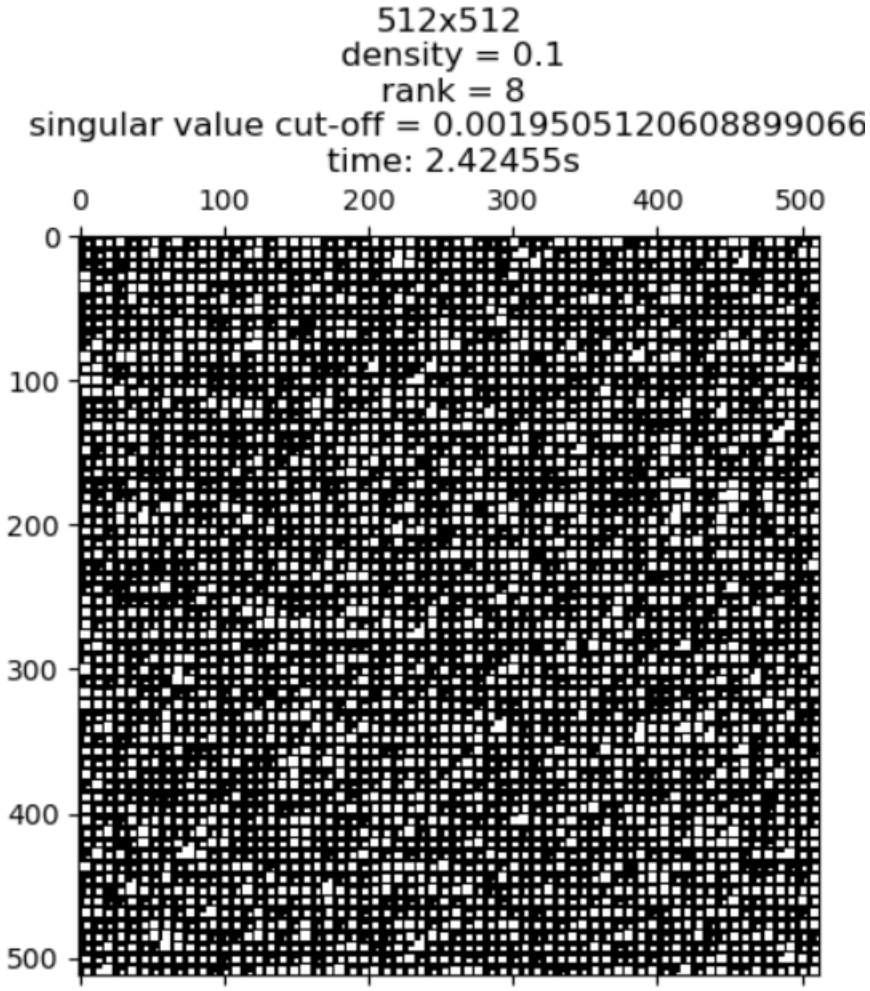

256x256
density = 0.1
rank = 2
singular value cut-off = 0.0031627455487670494
time: 3.05370s
0
50
100
150
200
250
0
50
100
150
200
250

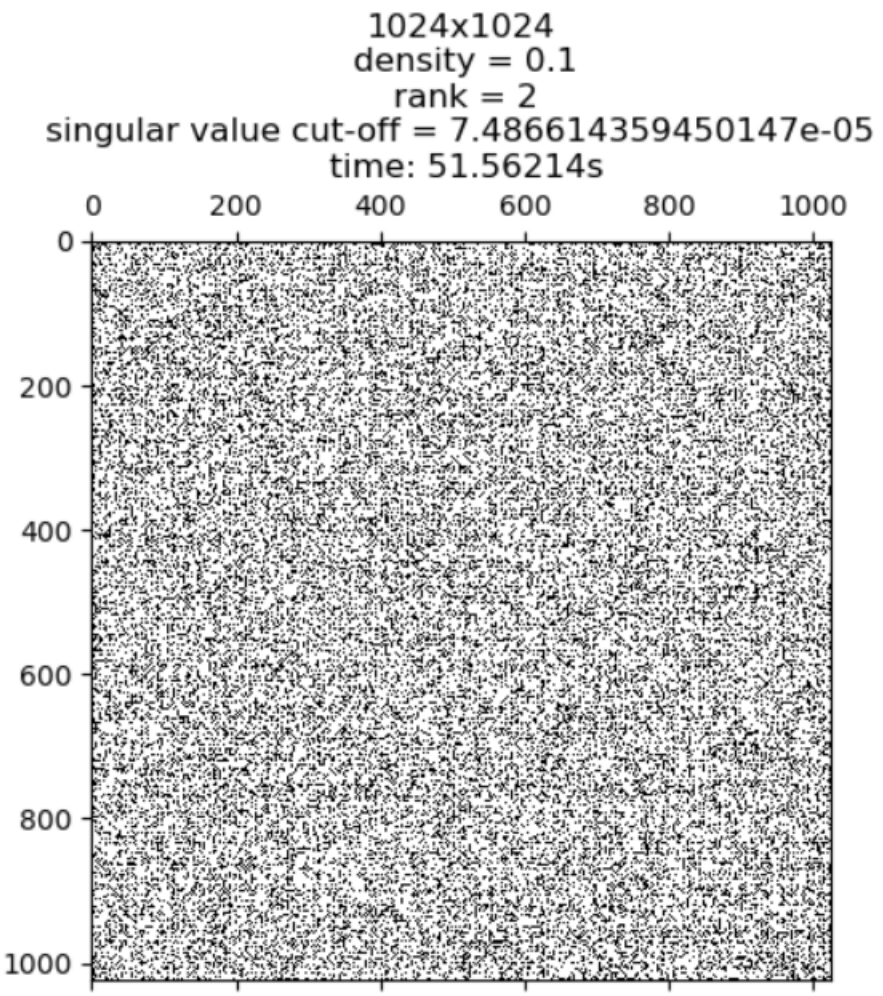

256x256
density = 0.1
rank = 4
singular value cut-off = 0.0031627455487670494
time: 1.50434s
0
50
100
150
200
250
0
50
100
150
200
250

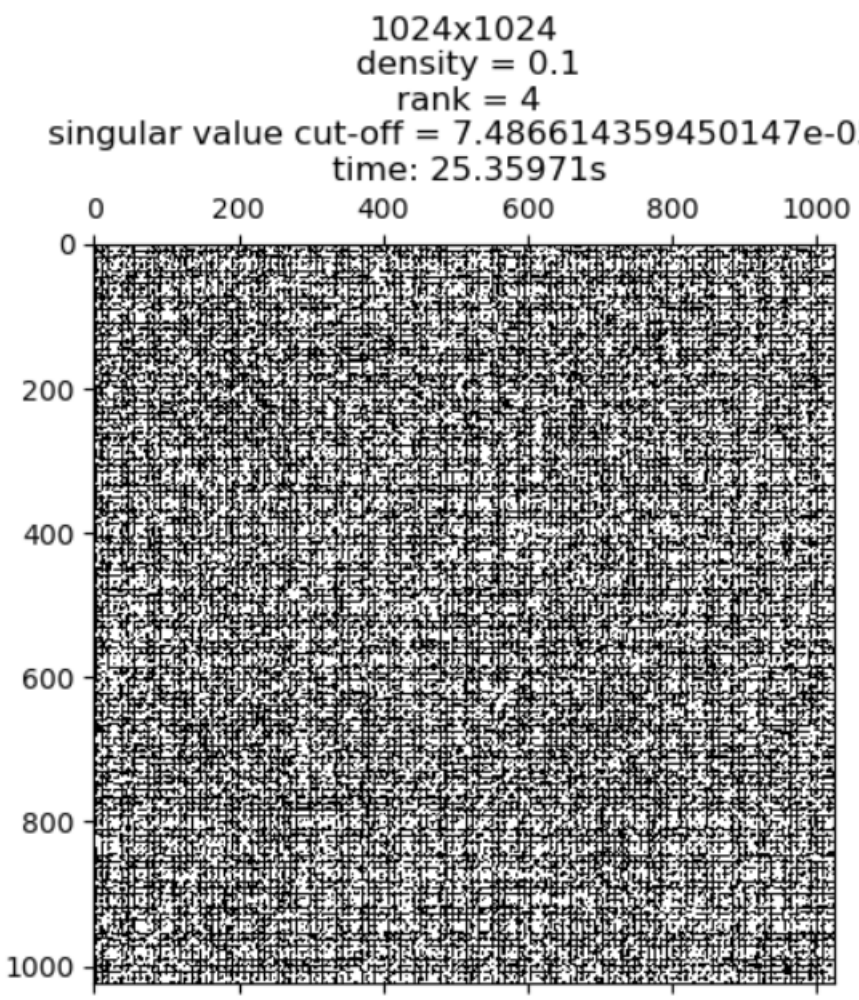

256x256
density = 0.1
rank = 8
singular value cut-off = 0.0031627455487670494
time: 0.55913s
0
50
100
150
200
250
0
50
100
150
200
250

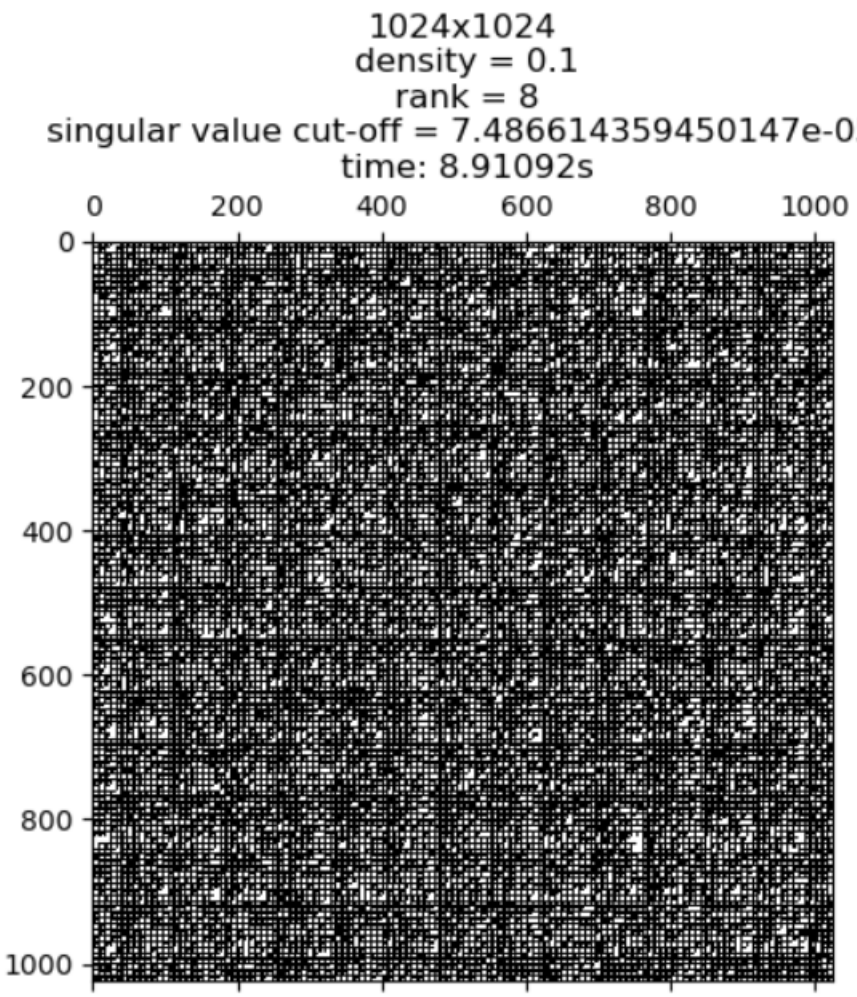

512x512
density = 0.1
rank = 2
singular value cut-off = 0.0019505120608899066
time: 12.15147s
0
100
200
300
400
500

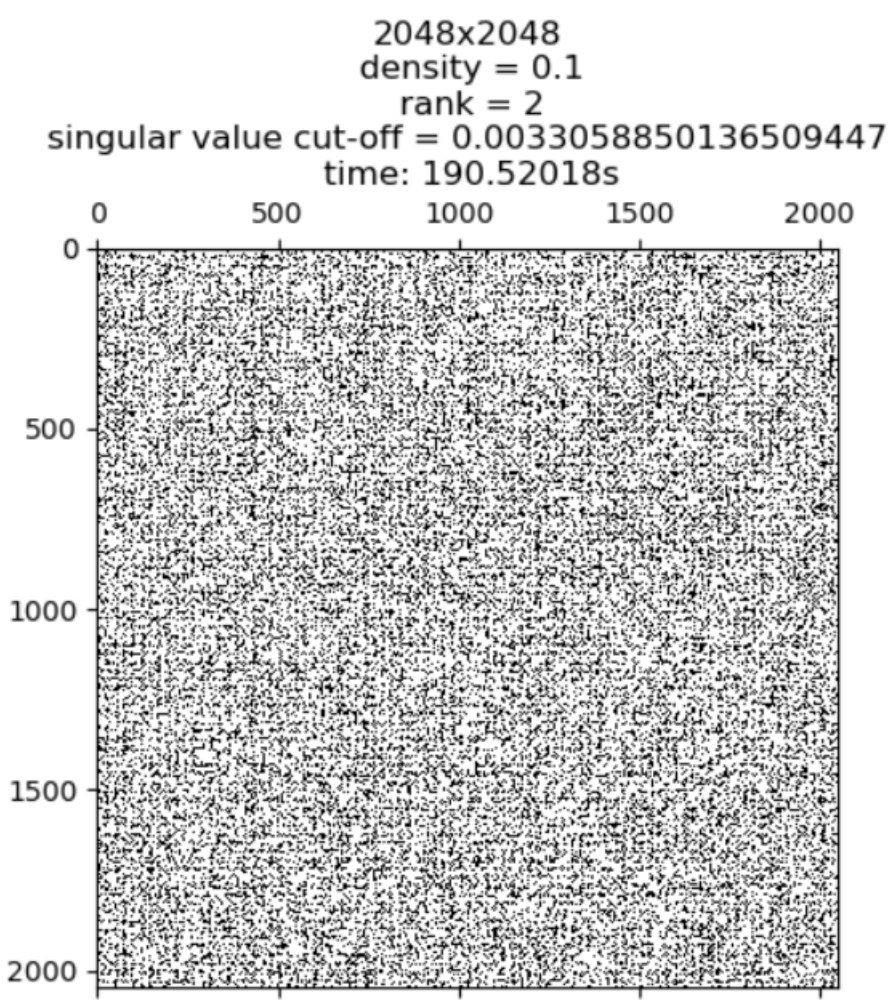

512x512
density = 0.1
rank = 4
singular value cut-off = 0.0019505120608899066
time: 6.55548s
0
100
200
300
400
500

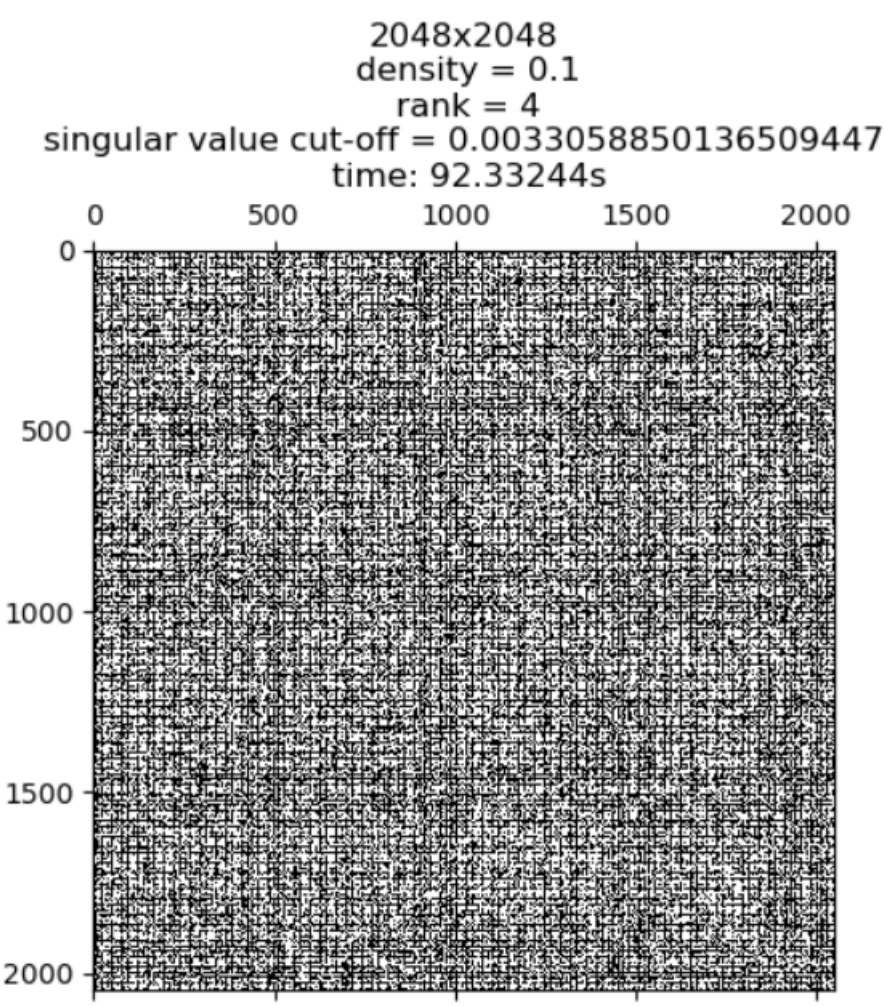

512x512
density = 0.1
rank = 8
singular value cut-off = 0.0019505120608899066
time: 2.42455s
0
100
200
300
400
500
0
100
200
300
400
500

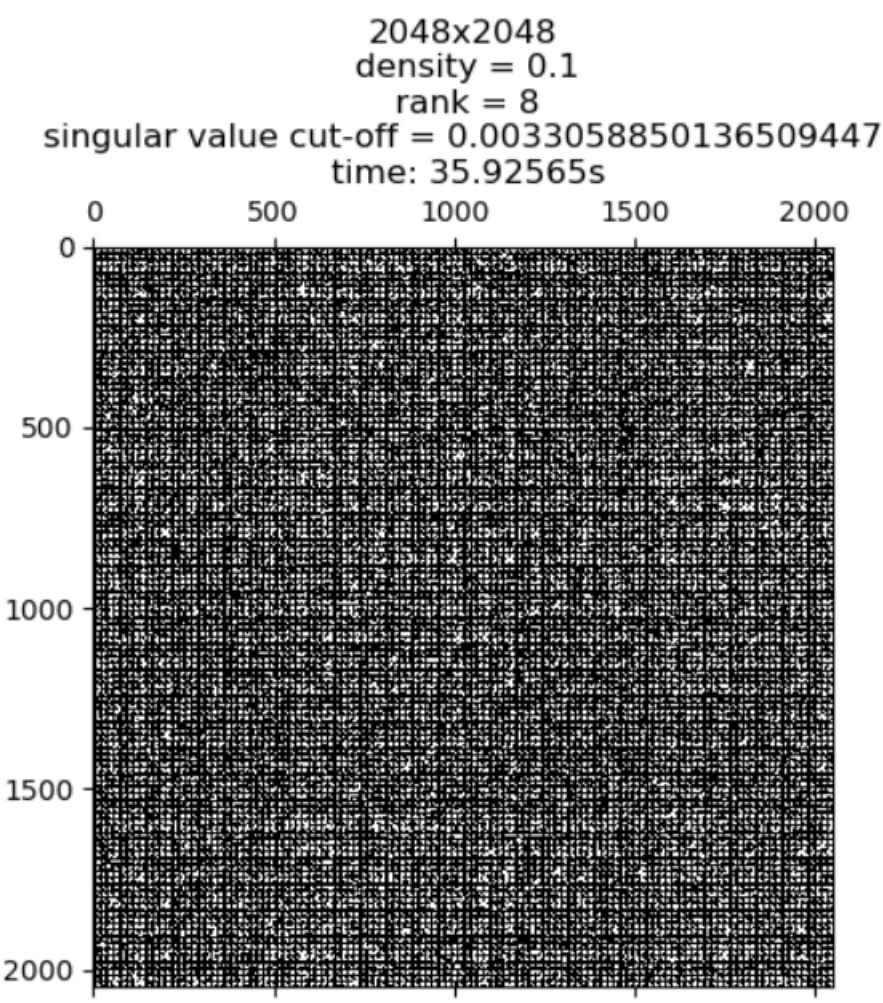

1024x1024
density = 0.1
rank = 2
singular value cut-off = 7.486614359450147e-05
time: 51.56214s
0
200
400
600
800
1000
0
200
400
600
800
1000

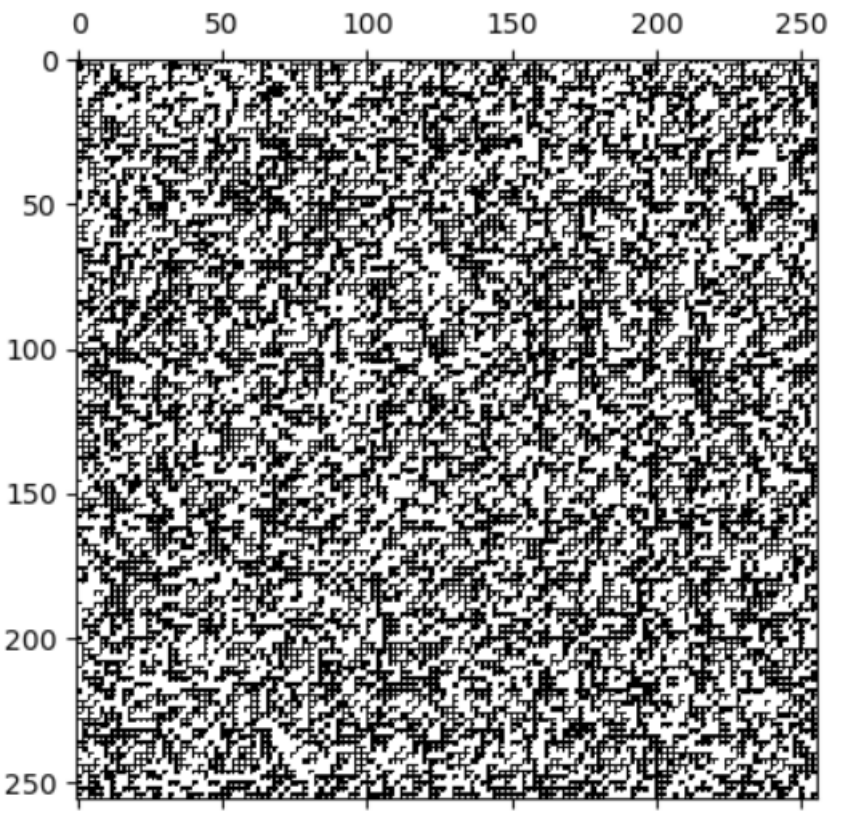

1024x1024
density = 0.1
rank = 4
singular value cut-off = 7.486614359450147e-05
time: 25.35971s
0
200
400
600
800
1000
0
200
400
600
800
1000

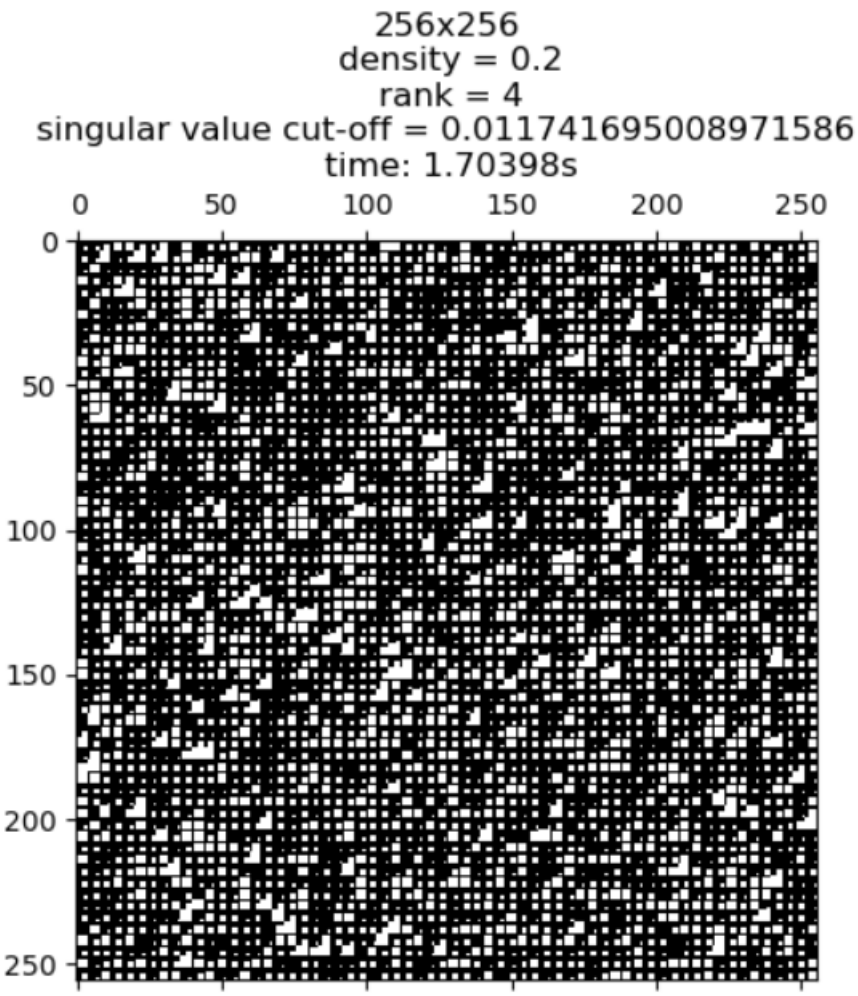

1024x1024
density = 0.1
rank = 8
singular value cut-off = 7.486614359450147e-05
time: 8.91092s
0
200
400
600
800
1000
0
200
400
600
800
1000

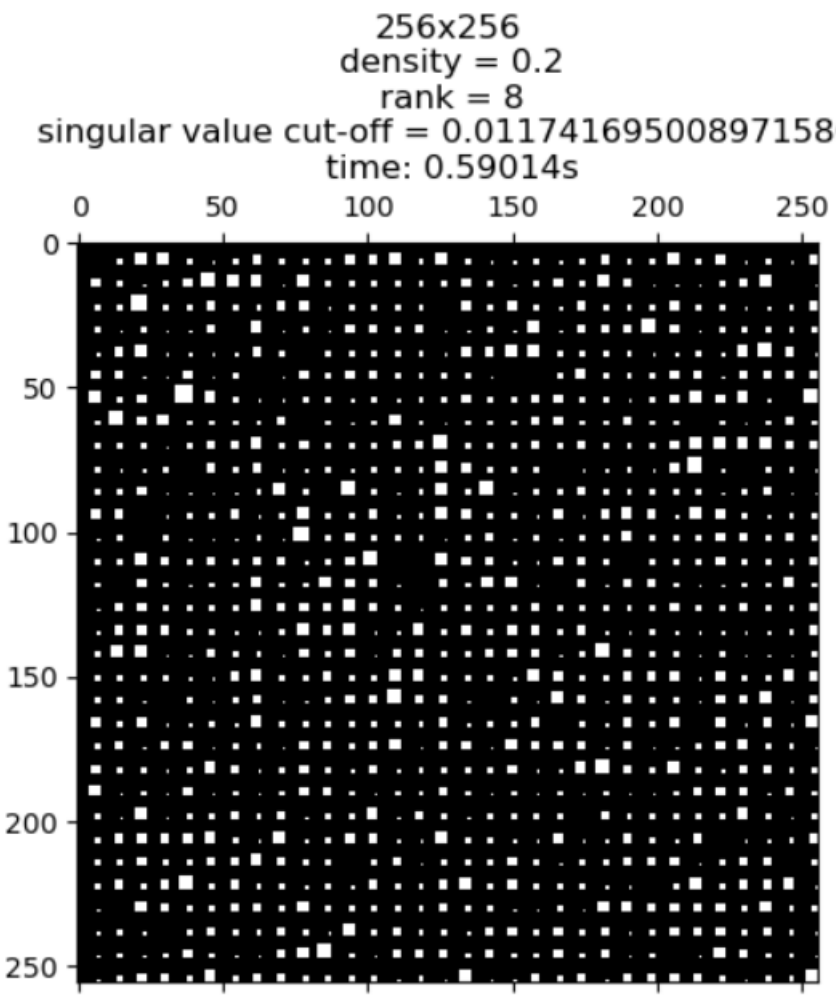

2048x2048
density = 0.1
rank = 2
singular value cut-off = 0.0033058850136509447
time: 190.52018s
0
500
1000
1500
2000
0
500
1000
1500
2000

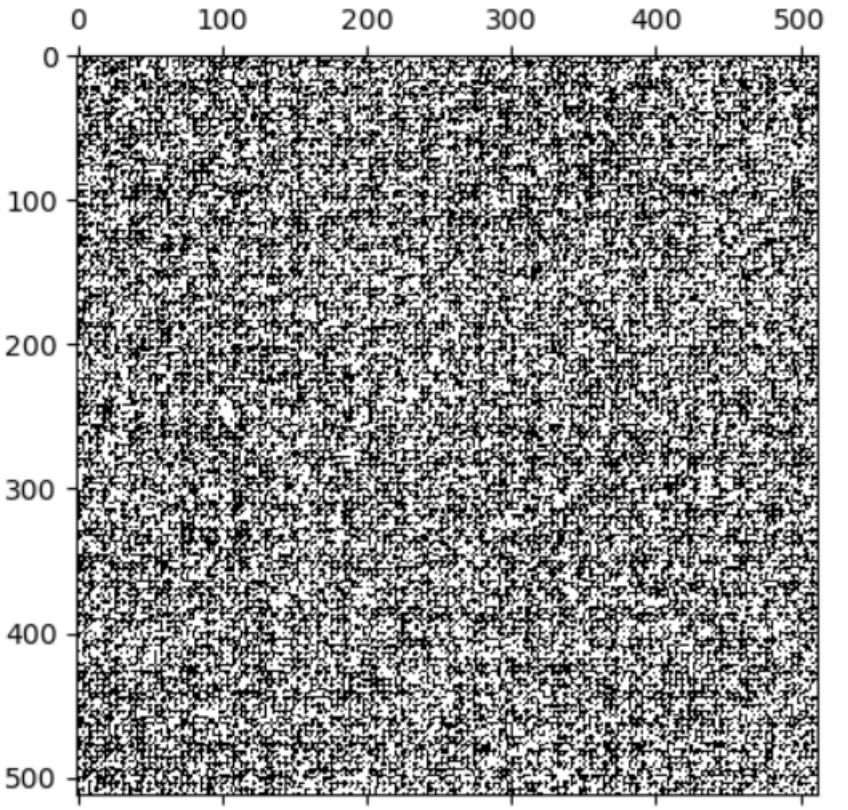

2048x2048
density = 0.1
rank = 4
singular value cut-off = 0.0033058850136509447
time: 92.33244s
0
500
1000
1500
2000
0
500
1000
1500
2000

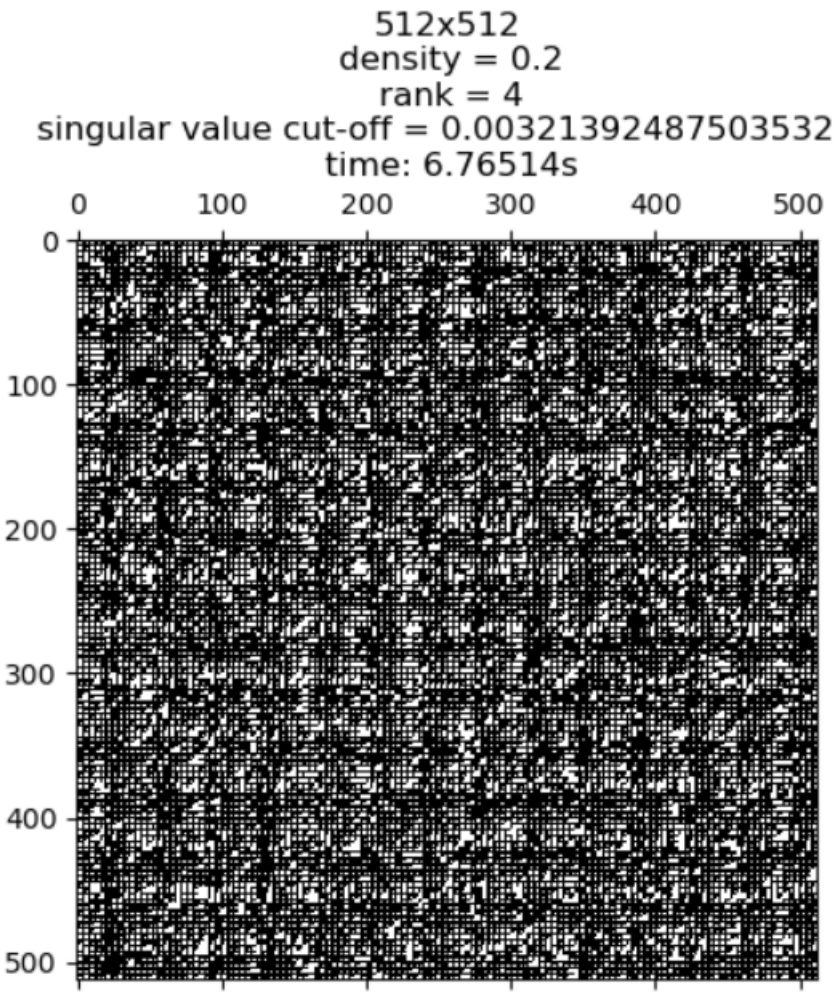

2048x2048
density = 0.1
rank = 8
singular value cut-off = 0.0033058850136509447
time: 35.92565s
0
500
1000
1500
2000
0
500
1000
1500
2000

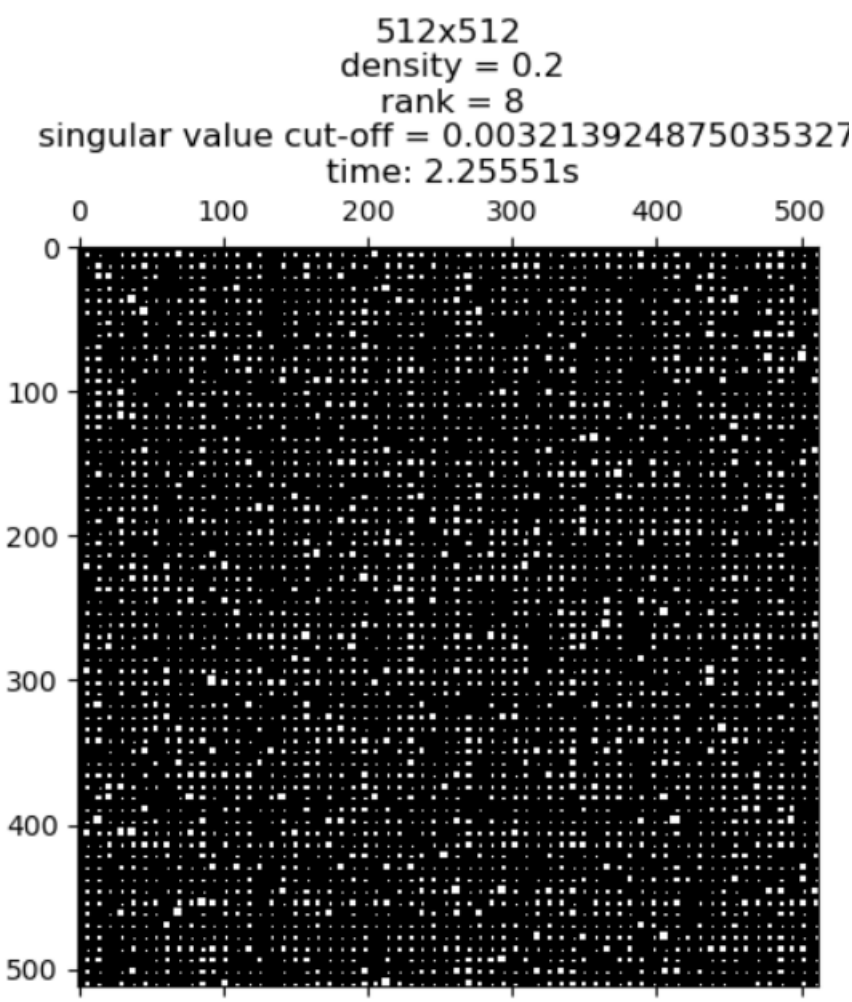

256x256
density = 0.2
rank = 2
singular value cut-off = 0.0117416950089715 86
time: 4.26998s
0
50
100
150
200
250
0
50
100
150
200
250

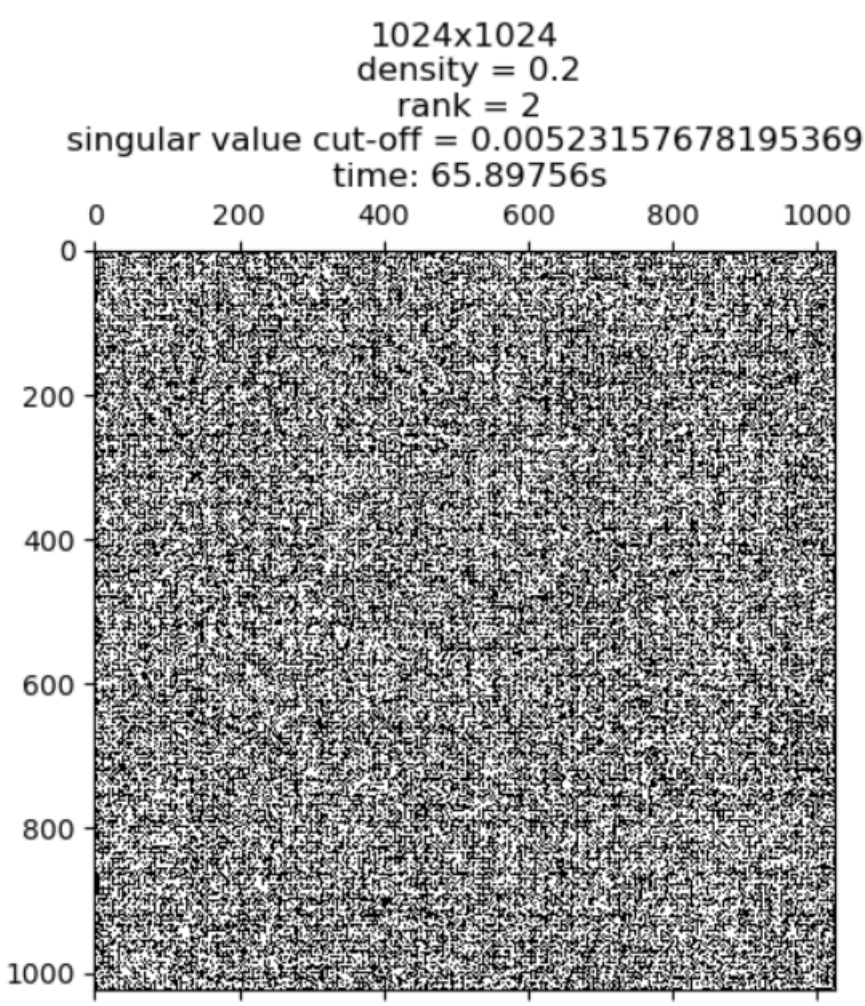

256x256
density = 0.2
rank = 4
singular value cut-off = 0.0117416950089715 86
time: 1.70398s
0
50
100
150
200
250
0
50
100
150
200
250

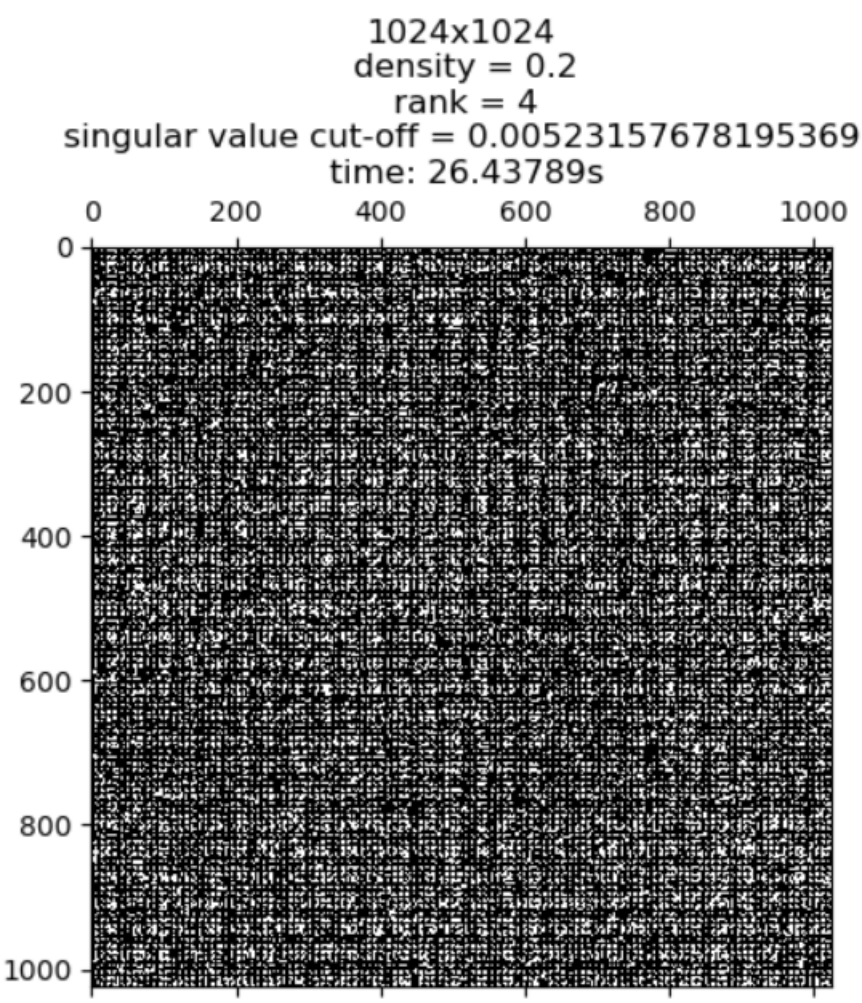

256x256
density = 0.2
rank = 8
singular value cut-off = 0.0117416950089715 86
time: 0.59014s
0
50
100
150
200
250
0
50
100
150
200
250

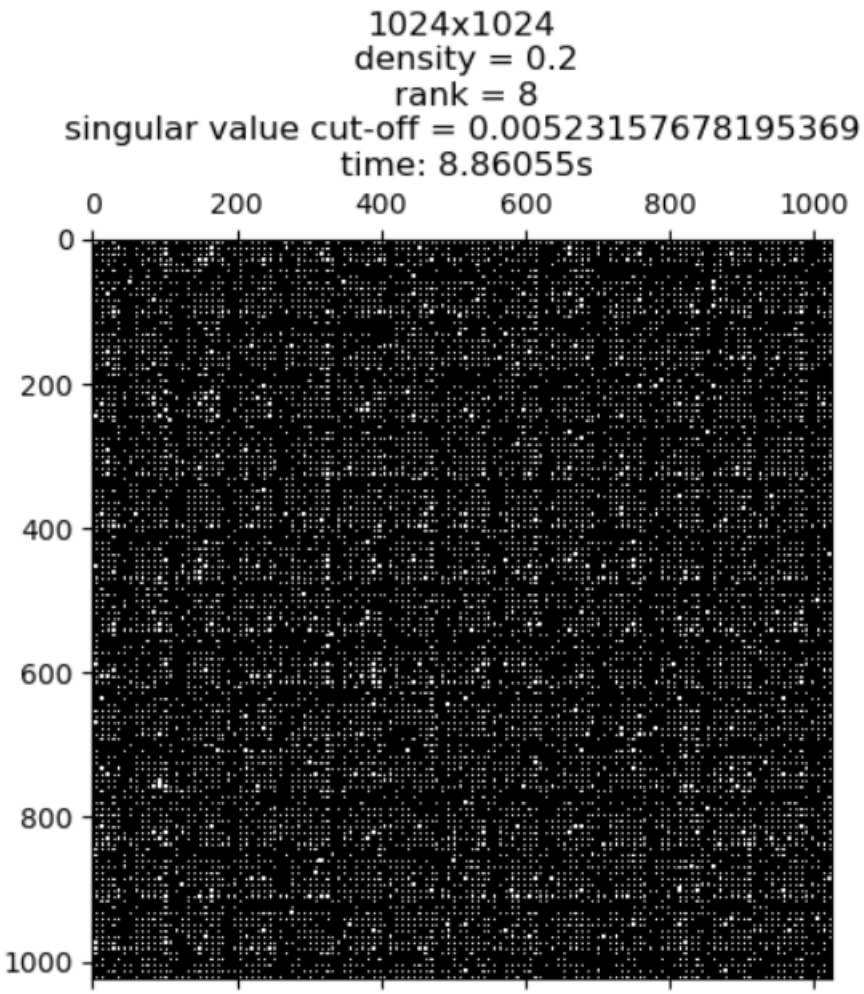

512x512
density = 0.2
rank = 2
singular value cut-off = 0.0032139248750353 27
time: 16.51118s
0
100
200
300
400
500
0
100
200
300
400
500

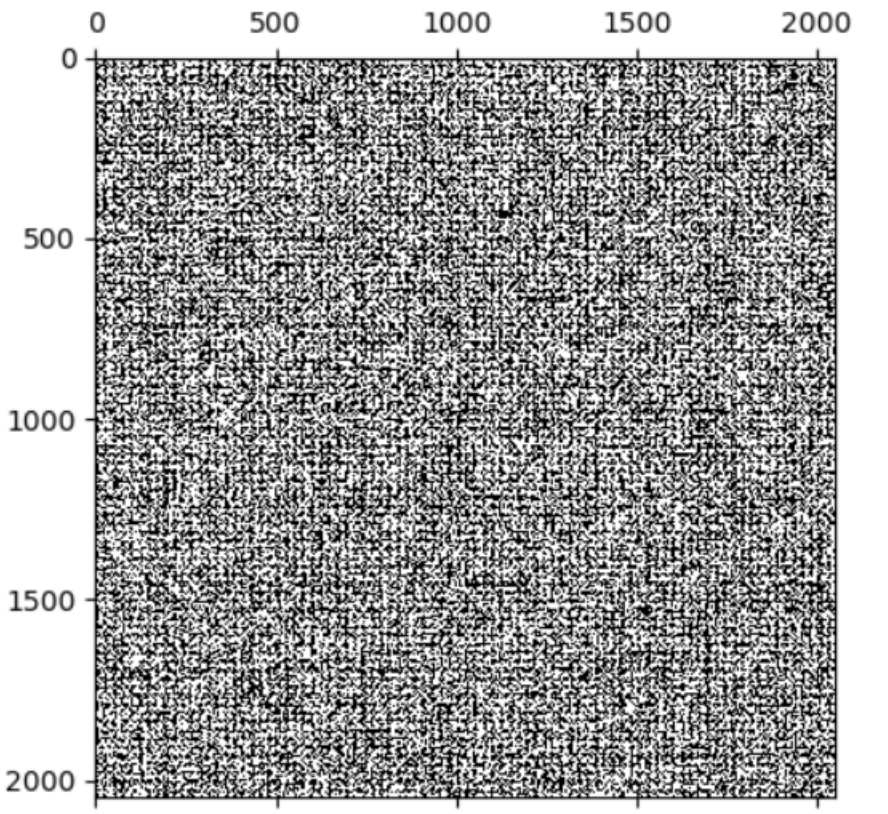

512x512
density = 0.2
rank = 4
singular value cut-off = 0.0032139248750353 27
time: 6.76514s
0
100
200
300
400
500
0
100
200
300
400
500

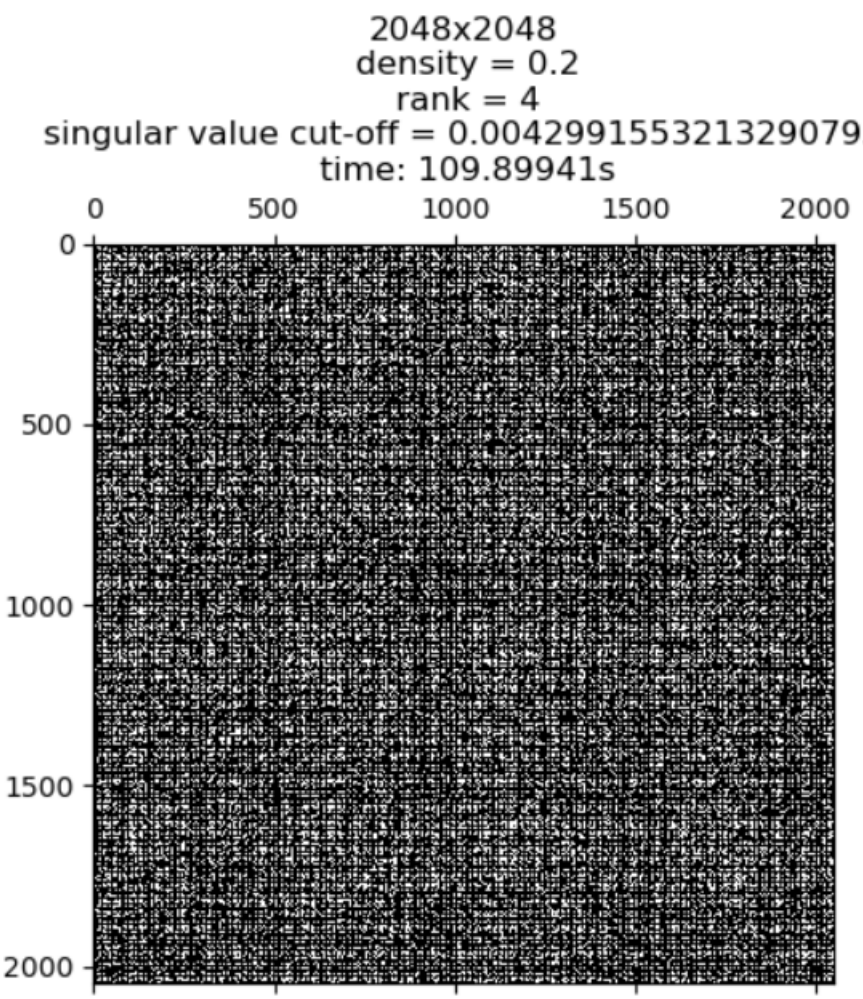

512x512
density = 0.2
rank = 8
singular value cut-off = 0.0032139248750353277
time: 2.25551s
0
100
200
300
400
500
0
100
200
300
400
500

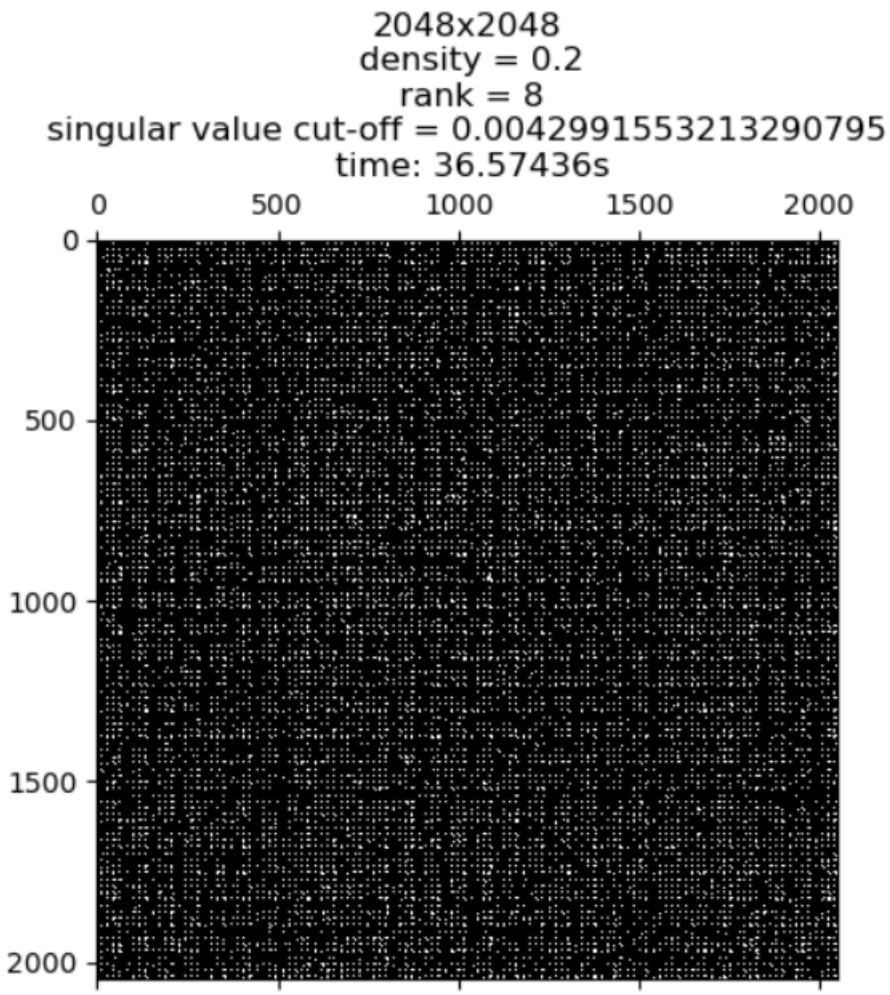

1024x1024
density = 0.2
rank = 2
singular value cut-off = 0.005231576781953 69
time: 65.89756s
0
200
400
600
800
1000
0
200
400
600
800
1000


1024x1024
density = 0.2
rank = 4
singular value cut-off = 0.005231576781953 69
time: 26.43789s
0
200
400
600
800
1000
0
200
400
600
800
1000


1024x1024
density = 0.2
rank = 8
singular value cut-off = 0.005231576781953 69
time: 8.86055s
0
200
400
600
800
1000
0
200
400
600
800
1000


2048x2048
density = 0.2
rank = 2
singular value cut-off = 0.00429915532132907 95
time: 265.89087s
0
500
1000
1500
2000
0
500
1000
1500
2000


2048x2048
density = 0.2
rank = 4
singular value cut-off = 0.0042991553213290795
time: 109.89941s
0
500
1000
1500
2000
0
500
1000
1500
2000


2048x2048
density = 0.2
rank = 8
singular value cut-off = 0.00429915532132907 95
time: 36.57436s
0
500
1000
1500
2000
0
500
1000
1500
2000