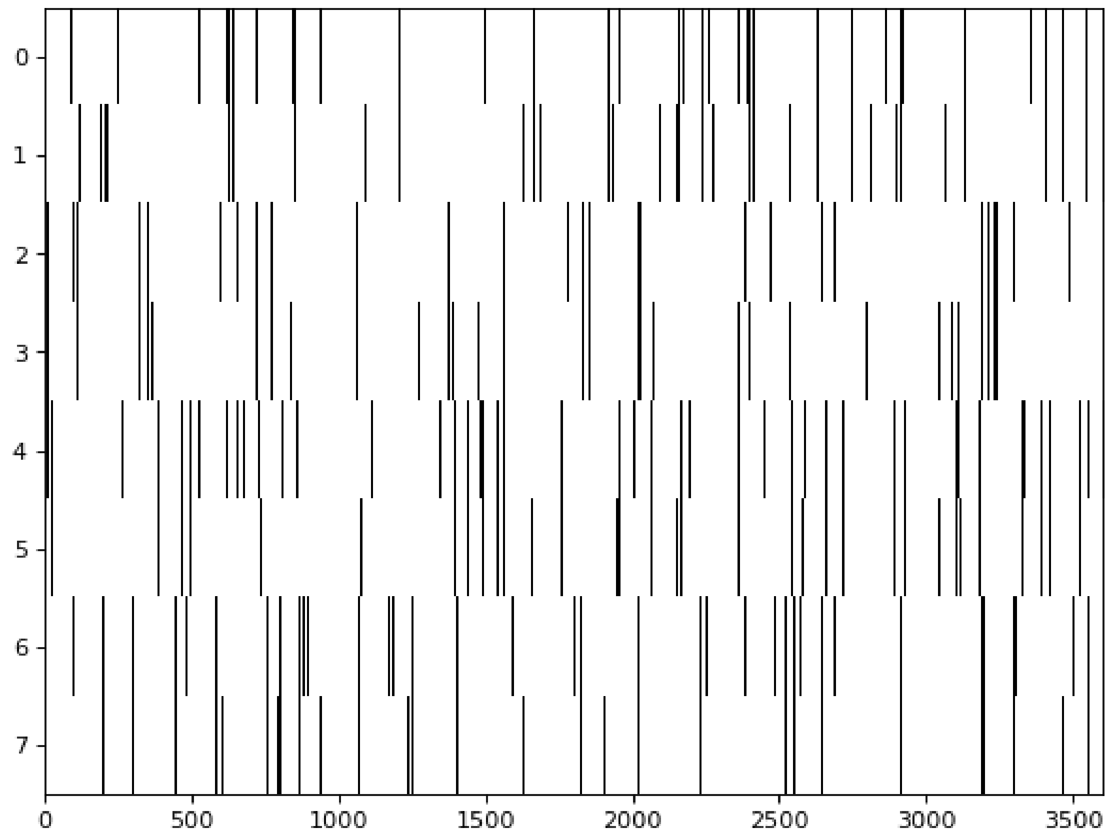
5 Model Multiplies Result by using Small Scale Dataset

August 3, 2023

[1]:
```
import warnings
#   context
with warnings.catch_warnings():
    warnings.filterwarnings("ignore")
    from coniii import *
import numpy as np
from tqdm.auto import tqdm
from scipy.spatial.distance import cdist
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
import random
import Jangenerate_assembly #In the same dir
import Jangenerate_SpikeCount #In the same dir
from scipy.stats import poisson

import itertools
import time
import math

warnings.filterwarnings("ignore")


# Define Parameters
T = 3600 # time of simul"ation
dT = 0.5 # time step
params_assembly_num =4 # number of assemblies
params_point_into_neuron_distance = 0.5

# Length of an active event as a number of timesteps
eventDur = np.random.randint(1, 10)
# Probability with which a unit is particularly active in a single timestep
eventProb = np.random.uniform(0.01, 0.05)
# Firing rate multiplier at active events
eventMult = np.random.uniform(6, 10)  # random number between 1 and 5
showPlot = True

def binaryOutput(original_list):
```

```
    # Create a new list to hold the tuples
    tuples_list = []

    # Generate all possible combinations of two elements for each sublist
    for sublist in original_list:
        combinations = itertools.combinations(sublist, 2)
        # Convert the combinations into tuples and add them to the list
        tuples_list.extend(tuple(sorted(combination)) for combination in␣
 ↪combinations)

    # Remove duplicates by converting the list to a set then back to a list
    unique_tuples = list(set(tuples_list))

    return unique_tuples

N = 8
params_assembly_density = 2 # size of neurons in each assembly

assemblies_list = []
spikeCount_list = []
binary_list = []

fire_rate_background = np.random.uniform(1, 6, N)
#assemblies = Jangenerate_assembly.generate_assembly_solve(N,␣
 ↪params_assembly_num, params_assembly_density)
assemblies = [[0, 1], [2, 3], [4, 5], [6, 7]]
# Output 0, 1 type spikes
spikeCount = Jangenerate_SpikeCount.generateSpikeCountSolve(N, T, dT,␣
 ↪assemblies, (1, 6), eventDur, eventProb, eventMult, showPlot)
# Transform to -1, 1 distribution
spikeCount[spikeCount == 0] = -1
assemblies_list.append(assemblies)
spikeCount_list.append(spikeCount)
print(assemblies)
print("_______________________________________")
binary_list.append(binaryOutput(assemblies))
```

```
[[0, 1], [2, 3], [4, 5], [6, 7]]
_______________________________________
```



[80]:
```
multipliers.shape
```

[80]: (36,)

[81]:
```
final_matrix.shape
```

[81]: (8, 8)

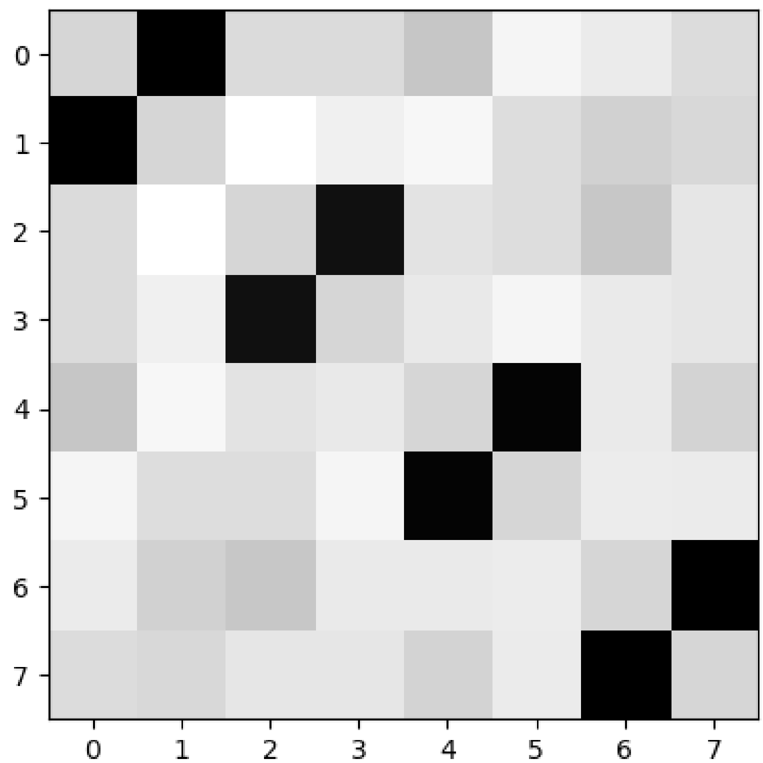
1 MCH Model

[2]:
```
solver = MCH(spikeCount, rng=np.random.
 ↪RandomState(0),n_cpus=2,sampler_kw={'boost':True})
multipliers, errflag, vstack = solver.solve(maxiter = 100, full_output=True)
mch= multipliers

matrix = np.zeros((N, N))
index = N
for i in range(N):
    for j in range(i+1, N):
```

```python
        matrix[i, j] = mch[index]
        index += 1
upper_matrix = np.triu(matrix)

lower_matrix = np.transpose(upper_matrix)
lower_matrix = np.tril(lower_matrix, -1)

final_matrix = upper_matrix + lower_matrix
#final_matrix = np.where(final_matrix < 1, 0, final_matrix)
print(final_matrix)
plt.imshow(final_matrix, cmap='gray_r')
```

```
[[ 0.          1.01303113 -0.02407972 -0.02494639  0.07435438 -0.14818347
  -0.10307371 -0.03224052]
 [ 1.01303113  0.         -0.19892855 -0.12350964 -0.15794094 -0.03631786
   0.01942806 -0.01032601]
 [-0.02407972 -0.19892855  0.          0.93407268 -0.06607223 -0.03311937
   0.06769251 -0.07662336]
 [-0.02494639 -0.12350964  0.93407268  0.         -0.09157319 -0.15035112
  -0.09925145 -0.07970615]
 [ 0.07435438 -0.15794094 -0.06607223 -0.09157319  0.          0.99581694
  -0.09822713  0.00715672]
 [-0.14818347 -0.03631786 -0.03311937 -0.15035112  0.99581694  0.
  -0.10593199 -0.10179064]
 [-0.10307371  0.01942806  0.06769251 -0.09925145 -0.09822713 -0.10593199
   0.          1.01550275]
 [-0.03224052 -0.01032601 -0.07662336 -0.07970615  0.00715672 -0.10179064
   1.01550275  0.        ]]
```

[2]: <matplotlib.image.AxesImage at 0x145180400>



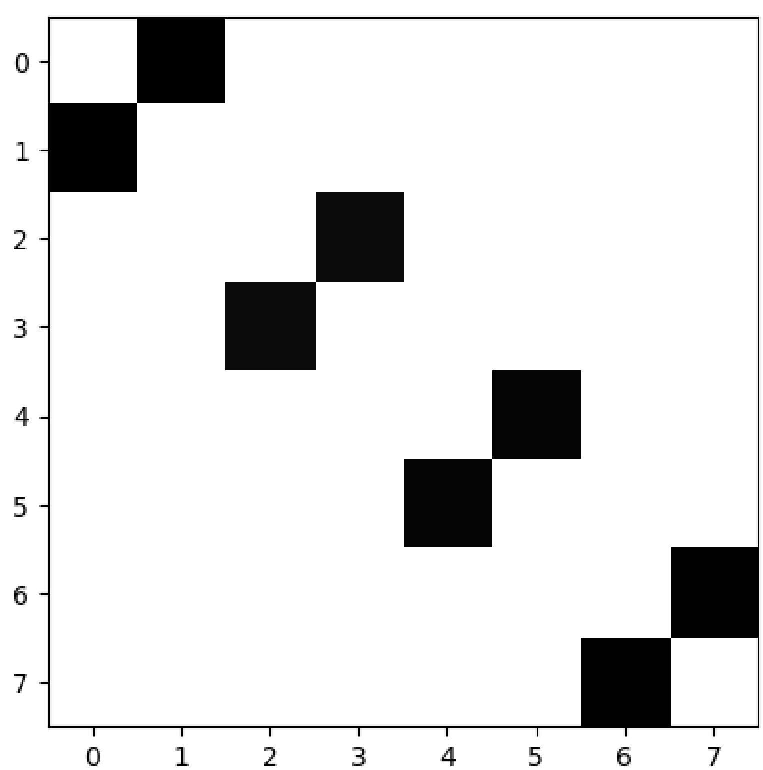
2 ACE Model

[3]:
```
solver = ClusterExpansion(spikeCount)
multipliers, ent, clusters, deltaSdict, deltaJdict= solver.solve(threshold = 0.
  ↪01, full_output=True)
ace = multipliers

matrix = np.zeros((N, N))
index = N
for i in range(N):
    for j in range(i+1, N):
        matrix[i, j] = ace[index]
        index += 1
upper_matrix = np.triu(matrix)

lower_matrix = np.transpose(upper_matrix)
lower_matrix = np.tril(lower_matrix, -1)

final_matrix = upper_matrix + lower_matrix
#final_matrix = np.where(final_matrix < 1, 0, final_matrix)
print(final_matrix)
```

```python
plt.imshow(final_matrix, cmap='gray_r')
```

```
adaptiveClusterExpansion: Clusters of size 2
adaptiveClusterExpansion: Clusters of size 3
[[0.         1.01545824 0.         0.         0.         0.
  0.         0.        ]
 [1.01545824 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.97035554 0.         0.
  0.         0.        ]
 [0.         0.         0.97035554 0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.9941811
  0.         0.        ]
 [0.         0.         0.         0.         0.9941811  0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         1.00594956]
 [0.         0.         0.         0.         0.         0.
  1.00594956 0.        ]]
```

[3]: <matplotlib.image.AxesImage at 0x144f04f40>

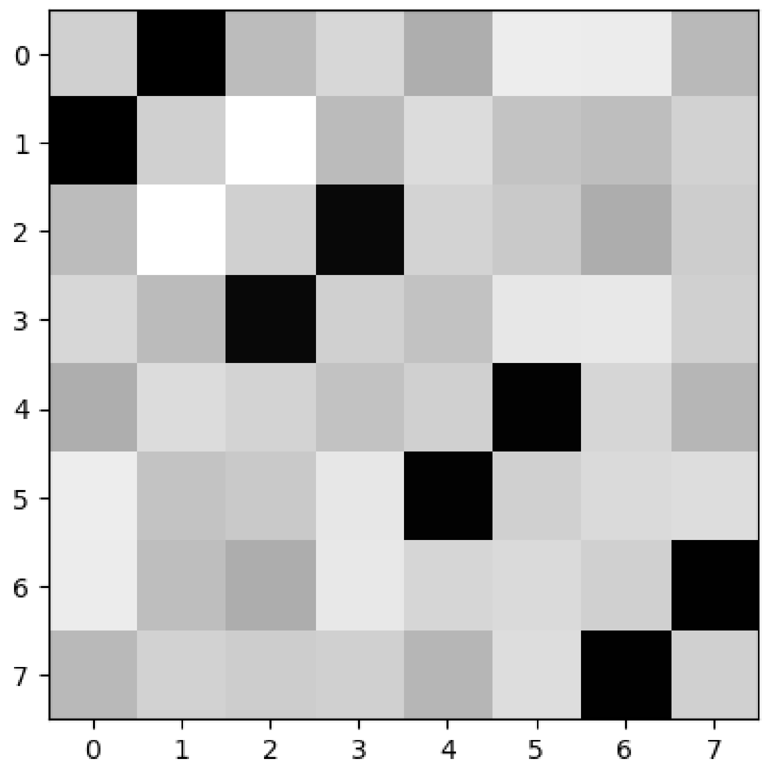


3 Pseudo Model

[4]:
```python
solver = Pseudo(spikeCount)
pse= solver.solve()

matrix = np.zeros((N, N))
index = N
for i in range(N):
    for j in range(i+1, N):
        matrix[i, j] = pse[index]
        index += 1
upper_matrix = np.triu(matrix)

lower_matrix = np.transpose(upper_matrix)
lower_matrix = np.tril(lower_matrix, -1)

final_matrix = upper_matrix + lower_matrix
#final_matrix = np.where(final_matrix < 1, 0, final_matrix)
print(final_matrix)

plt.imshow(final_matrix, cmap='gray_r')
```

```
[[ 0.00000000e+00  1.02186963e+00  1.00177565e-01 -3.15808332e-02
   1.65447549e-01 -1.39980914e-01 -1.36105633e-01  1.12089627e-01]
 [ 1.02186963e+00  0.00000000e+00 -2.31713124e-01  1.04891596e-01
  -5.68304628e-02  5.78230594e-02  9.05254518e-02 -8.95997432e-03]
 [ 1.00177565e-01 -2.31713124e-01  0.00000000e+00  9.79936502e-01
  -1.52974581e-02  3.60193953e-02  1.74557223e-01  1.79062457e-02]
 [-3.15808332e-02  1.04891596e-01  9.79936502e-01  0.00000000e+00
   7.11390400e-02 -1.12311063e-01 -1.16178363e-01 -8.21162190e-04]
 [ 1.65447549e-01 -5.68304628e-02 -1.52974581e-02  7.11390400e-02
   0.00000000e+00  1.00232449e+00 -2.84193351e-02  1.30428231e-01]
 [-1.39980914e-01  5.78230594e-02  3.60193953e-02 -1.12311063e-01
   1.00232449e+00  0.00000000e+00 -4.64647508e-02 -6.49127954e-02]
 [-1.36105633e-01  9.05254518e-02  1.74557223e-01 -1.16178363e-01
  -2.84193351e-02 -4.64647508e-02  0.00000000e+00  1.00878942e+00]
 [ 1.12089627e-01 -8.95997432e-03  1.79062457e-02 -8.21162190e-04
   1.30428231e-01 -6.49127954e-02  1.00878942e+00  0.00000000e+00]]
```

[4]:
```
<matplotlib.image.AxesImage at 0x144f42860>
```

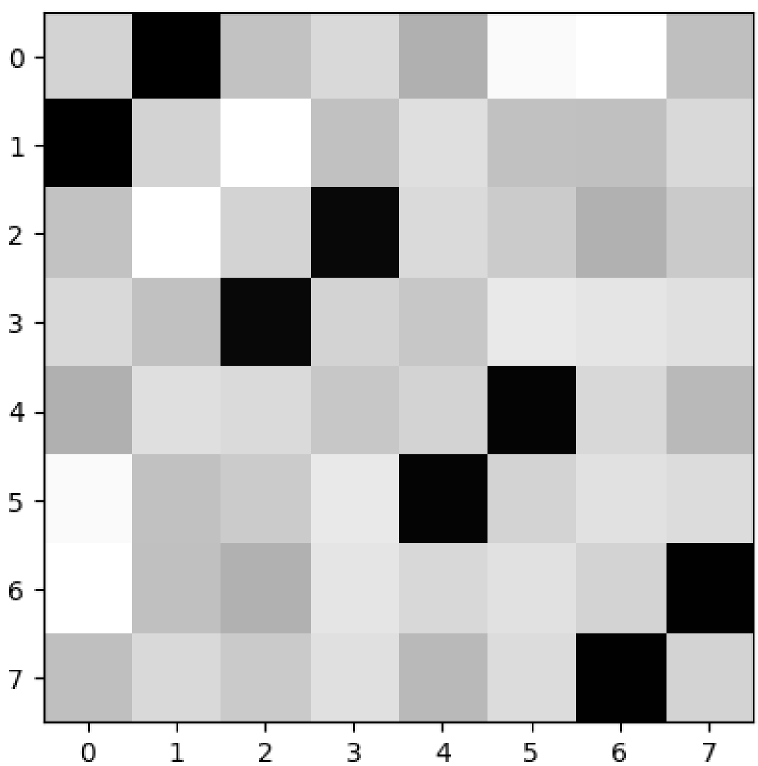


4 MPF Model

[6]:
```
solver = MPF(spikeCount)
mpf= solver.solve()

matrix = np.zeros((N, N))
index = N
for i in range(N):
    for j in range(i+1, N):
        matrix[i, j] = mpf[index]
        index += 1
upper_matrix = np.triu(matrix)

lower_matrix = np.transpose(upper_matrix)
lower_matrix = np.tril(lower_matrix, -1)

final_matrix = upper_matrix + lower_matrix
#final_matrix = np.where(final_matrix < 1, 0, final_matrix)
print(final_matrix)

plt.imshow(final_matrix, cmap='gray_r')
```

```
[[ 0.          1.02397964  0.08558408 -0.02643278  0.17544872 -0.18313193
  -0.20591132  0.10247674]
 [ 1.02397964  0.         -0.20976937  0.09081944 -0.05107257  0.09142167
   0.09810859 -0.02267283]
 [ 0.08558408 -0.20976937  0.          0.97941143 -0.02743291  0.04268213
   0.16993604  0.04699703]
 [-0.02643278  0.09081944  0.97941143  0.          0.06239857 -0.10342833
  -0.0797715  -0.05658309]
 [ 0.17544872 -0.05107257 -0.02743291  0.06239857  0.          1.00305477
  -0.01842449  0.12931304]
 [-0.18313193  0.09142167  0.04268213 -0.10342833  1.00305477  0.
  -0.06487989 -0.03863889]
 [-0.20591132  0.09810859  0.16993604 -0.0797715  -0.01842449 -0.06487989
   0.          1.00977897]
 [ 0.10247674 -0.02267283  0.04699703 -0.05658309  0.12931304 -0.03863889
   1.00977897  0.        ]]
```

[6]: <matplotlib.image.AxesImage at 0x144b0e290>

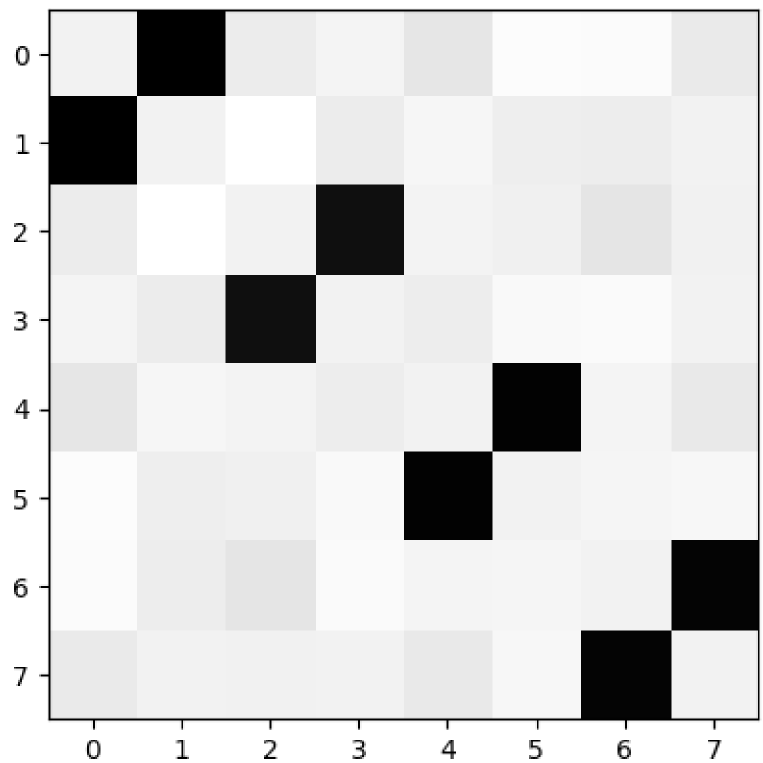


5 RMF Model

[7]:
```
solver = RegularizedMeanField(spikeCount)
rmf= solver.solve()

matrix = np.zeros((N, N))
index = N
for i in range(N):
    for j in range(i+1, N):
        matrix[i, j] = rmf[index]
        index += 1
upper_matrix = np.triu(matrix)

lower_matrix = np.transpose(upper_matrix)
lower_matrix = np.tril(lower_matrix, -1)

final_matrix = upper_matrix + lower_matrix
#final_matrix = np.where(final_matrix < 1, 0, final_matrix)
print(final_matrix)

plt.imshow(final_matrix, cmap='gray_r')
```

```
coocSampleCovariance : WARNING : using ad-hoc 'Laplace' correction
[[ 0.00000000e+00  7.32738824e+00  1.89250048e-01 -6.29562465e-02
   3.82075117e-01 -2.94658366e-01 -2.62923594e-01  2.33380823e-01]
 [ 7.32738824e+00  0.00000000e+00 -4.01543379e-01  1.96712921e-01
  -1.12281068e-01  1.20558126e-01  1.62064147e-01  7.82364845e-03]
 [ 1.89250048e-01 -4.01543379e-01  0.00000000e+00  6.86932428e+00
  -3.73725328e-02  7.45435555e-02  3.97575172e-01  4.89358045e-02]
 [-6.29562465e-02  1.96712921e-01  6.86932428e+00  0.00000000e+00
   1.49723835e-01 -2.09000852e-01 -2.37038170e-01  6.25890826e-03]
 [ 3.82075117e-01 -1.12281068e-01 -3.73725328e-02  1.49723835e-01
   0.00000000e+00  7.22428926e+00 -5.73914043e-02  2.90577883e-01]
 [-2.94658366e-01  1.20558126e-01  7.45435555e-02 -2.09000852e-01
   7.22428926e+00  0.00000000e+00 -8.51147500e-02 -1.39791116e-01]
 [-2.62923594e-01  1.62064147e-01  3.97575172e-01 -2.37038170e-01
  -5.73914043e-02 -8.51147500e-02  0.00000000e+00  7.18914652e+00]
 [ 2.33380823e-01  7.82364845e-03  4.89358045e-02  6.25890826e-03
   2.90577883e-01 -1.39791116e-01  7.18914652e+00  0.00000000e+00]]
```

[7]:
```
<matplotlib.image.AxesImage at 0x1446e34f0>
```


0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

[ ]: