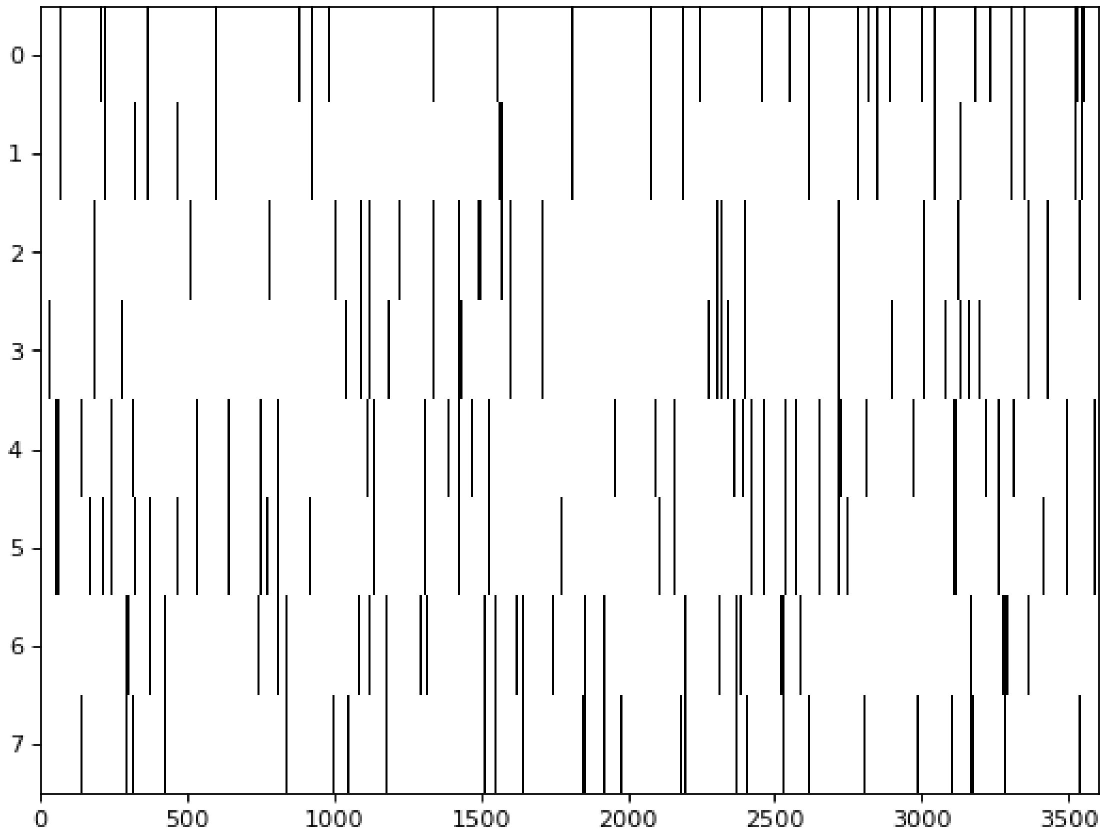
5 Model Multiplies Result by using Small Scale Dataset

August 1, 2023

[83]:

```python
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)
```

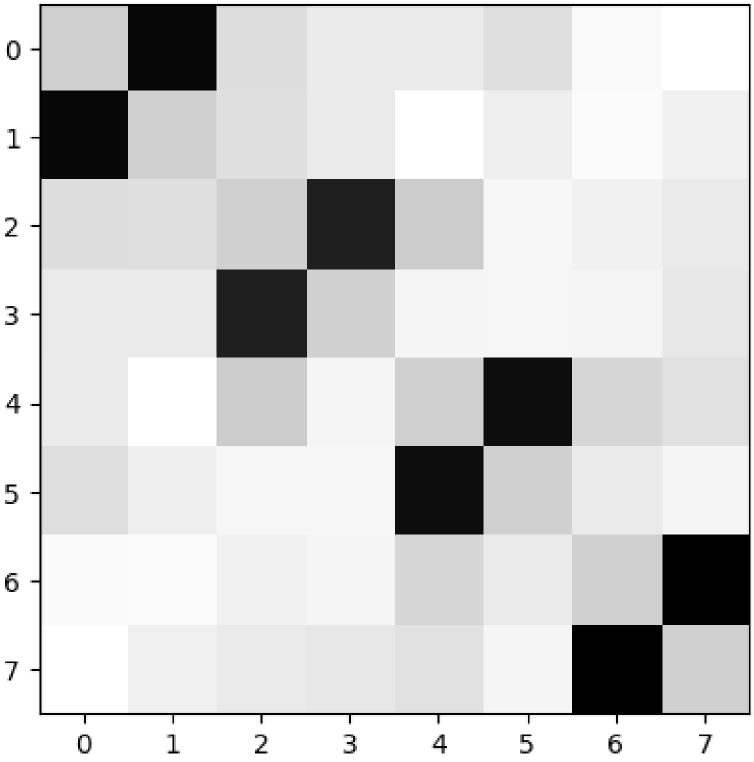
1 MCH Model

[79]:

```
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):
        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)

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

```
int8
122
```

```
[79]: <matplotlib.image.AxesImage at 0x1573ad0f0>
```



2 ACE Model

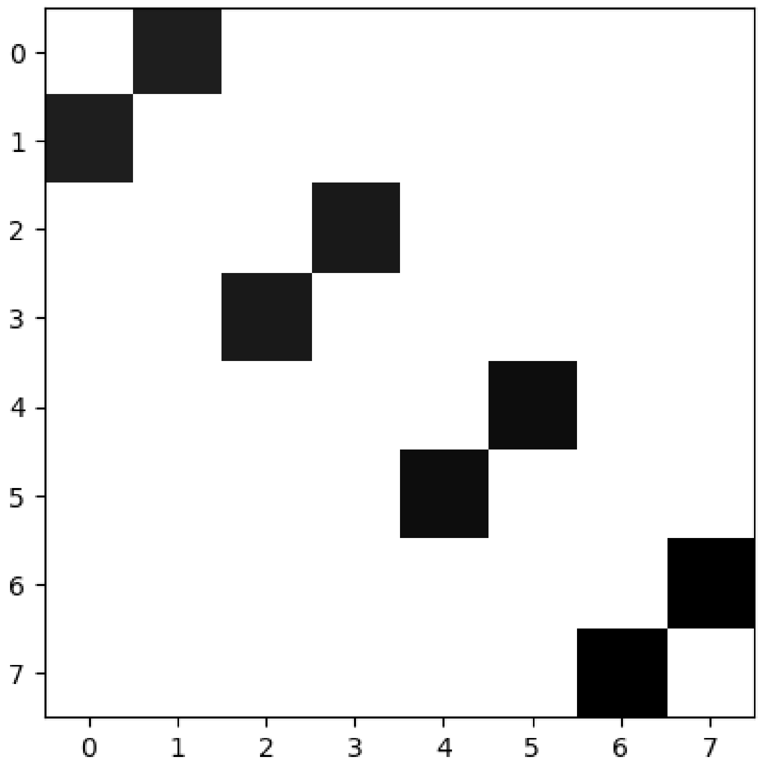
[82]:

```
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)

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

```
adaptiveClusterExpansion: Clusters of size 2
adaptiveClusterExpansion: Clusters of size 3
```

[82]:

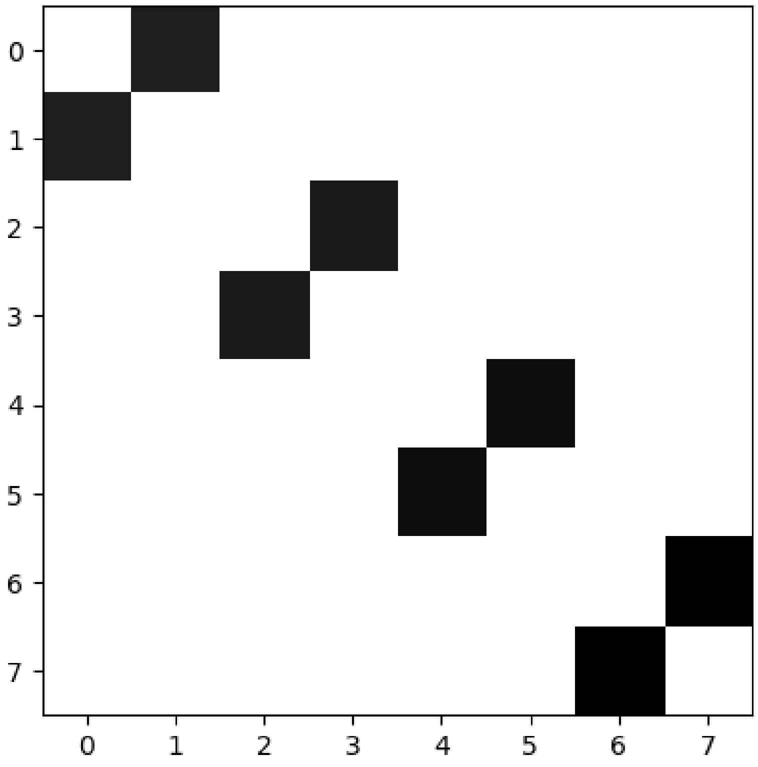
```
<matplotlib.image.AxesImage at 0x15826b490>
```



3 Pseudo Model

[84]:
```
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)

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

```
[84]: <matplotlib.image.AxesImage at 0x15b298220>
```



4 MPF Model

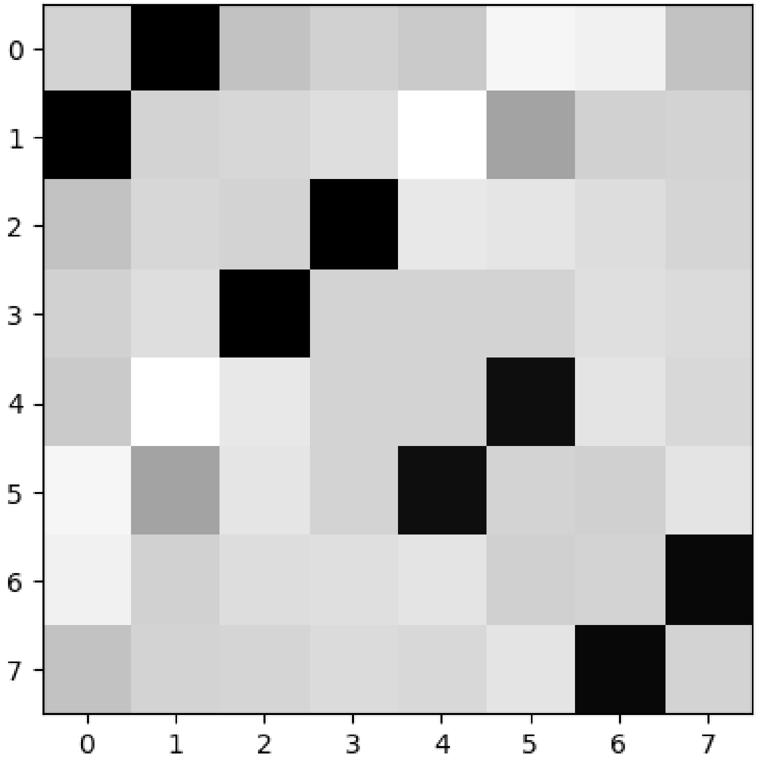
[85]:

```
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)
```

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

[85]:

```
<matplotlib.image.AxesImage at 0x15b4eedd0>
```



5 RMF Model

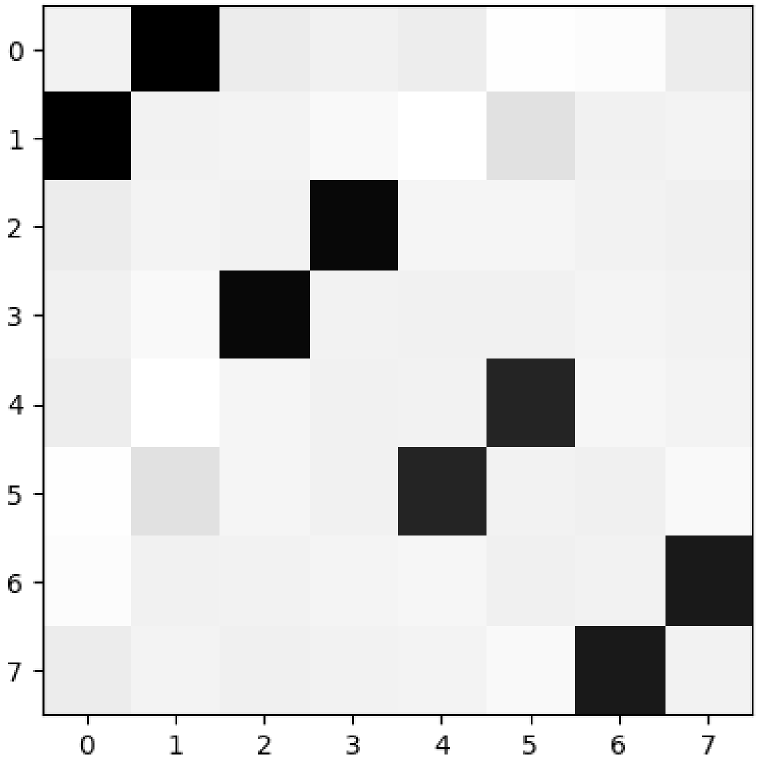
[86]:

```
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)

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

```
coocSampleCovariance : WARNING : using ad-hoc 'Laplace' correction
```

```
[86]: <matplotlib.image.AxesImage at 0x15b533a00>
```



```
[ ]:
```