1 Tutorial 01 Hi-C-Analysis

In this file is shown a basic introduction on how to use the Hi-C-Analysis package to performa a spectral analysis and visualize the results.

In this script, only to shows how the main functions work, it will be used a square and symmetric random matrix created with numpy and a fake dataframe containing the start and stop indices of the chromosomes Hi-C matrices:

[1]:
```
import numpy as np
import pandas as pd

np.random.seed(42)
adjacency = 500 * np.random.random((300, 300))
adjacency += adjacency.T

data = {'chr':['chr1', 'chr2', 'chr3', 'chr4'], 'start':[1, 100, 190, 270],␣
 ↪'end':[99, 189, 269, 299 ]}

metadata = pd.DataFrame(data)
metadata
```

[1]:
```
    chr  start  end
0  chr1      1   99
1  chr2    100  189
2  chr3    190  269
3  chr4    270  299
```

Now that the adjacecy matrix is created let's import the other packages needed for the process.

[12]:
```
from pathlib import Path
import hicanalysis.preprocessing as pre
import hicanalysis.visualizegraph as vg
from scipy.stats import pearsonr
```

To normalize the adjacency matrix with the Observed Over Expected normalization:

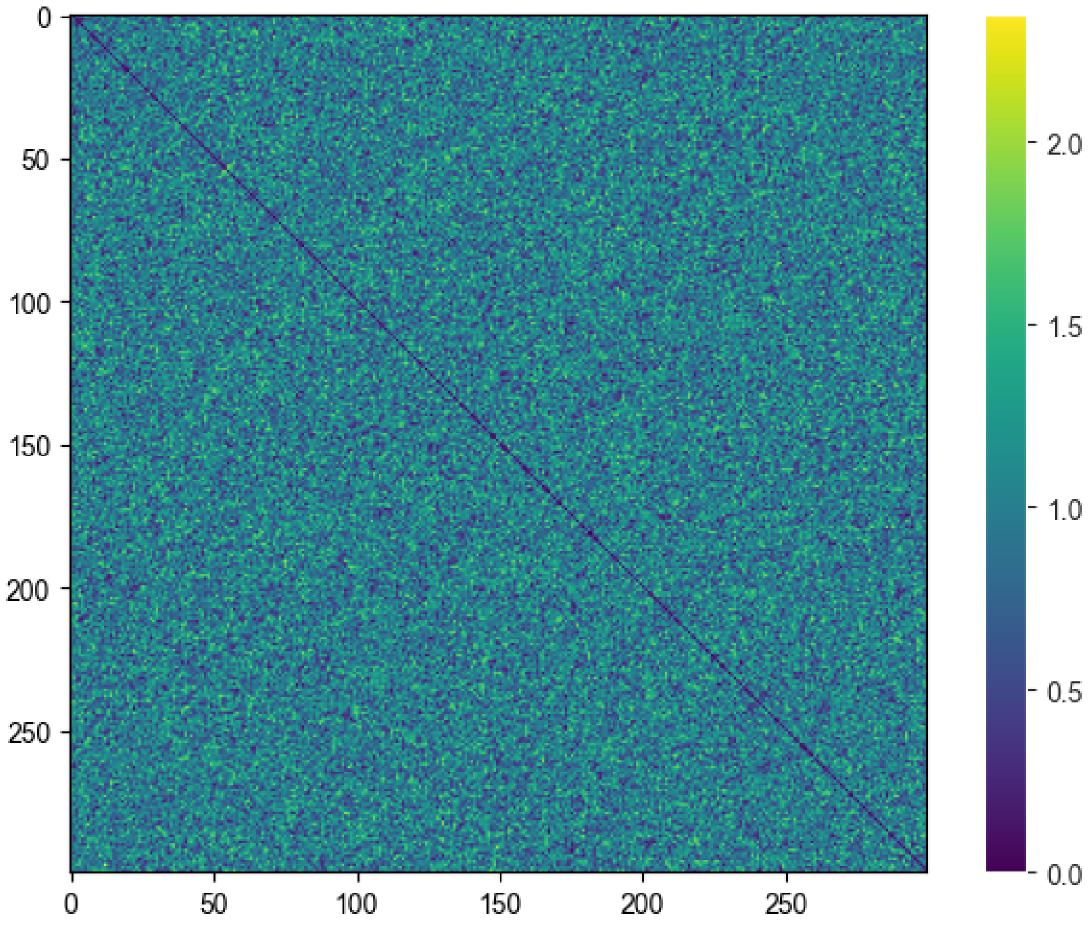
[3]:
```
normalized_adj = pre.ooe_normalization(adjacency)
```

In order to extract the blocks of the chromsomes from the diagonal of the normalized adjacency it is enough to apply the function get_chromosome_list and to provide the list to extract_diagonal_blocks:

[4]:
```
list = pre.get_chromosome_list(metadata)

blocks = pre.extract_diagonal_blocks(normalized_adj, list)
```

in order to visualize them it's possible to use the visualizegraph module:

[ ]:
```
vg.plot_matrix(normalized_adj)
```



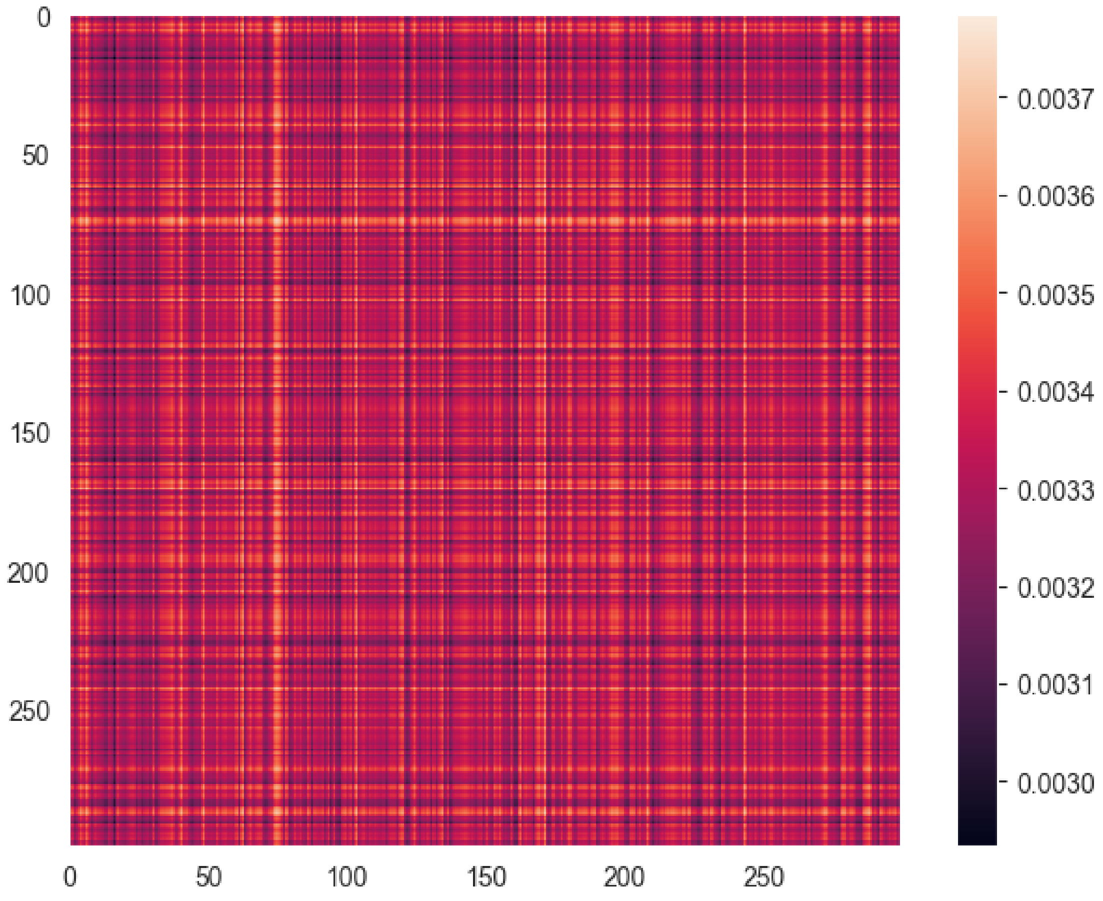
Then it is possible to use the numpy library to find the eigenvalues and eigenvectors of the matrix, and order them from the greater to the smaller eigenvalues

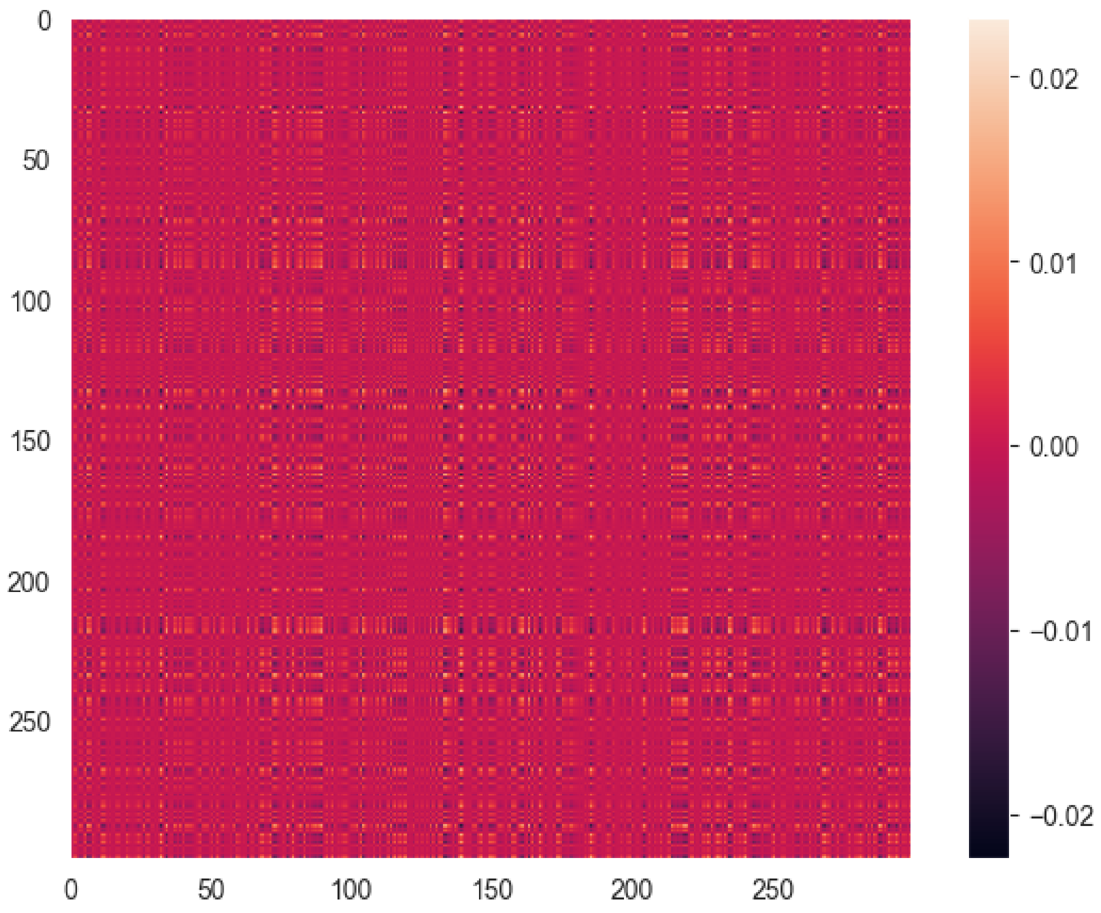
[5]:
```
eigenvalues, eigenvectors = np.linalg.eig(normalized_adj)

idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]
```

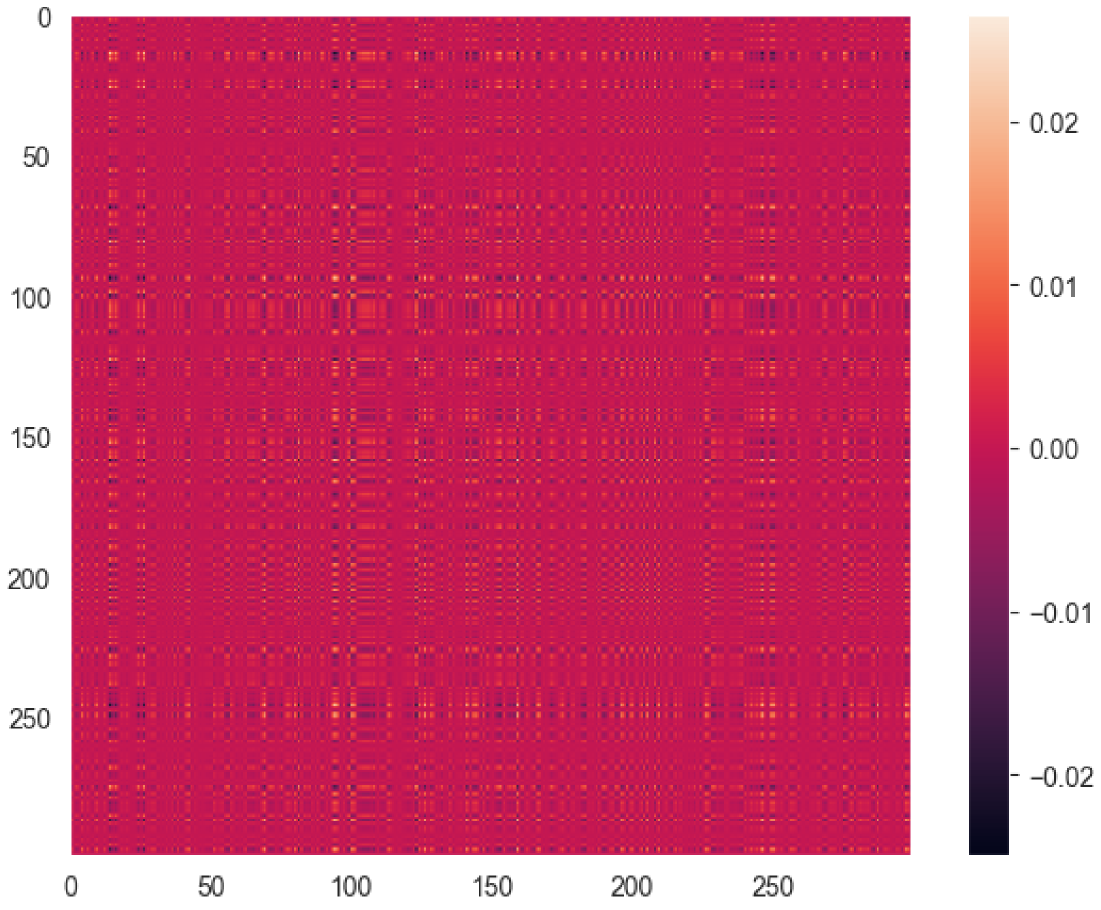
The obtained results can be passed to the build_projection function, that, from the eigenvectors, create the corresponding projectors:

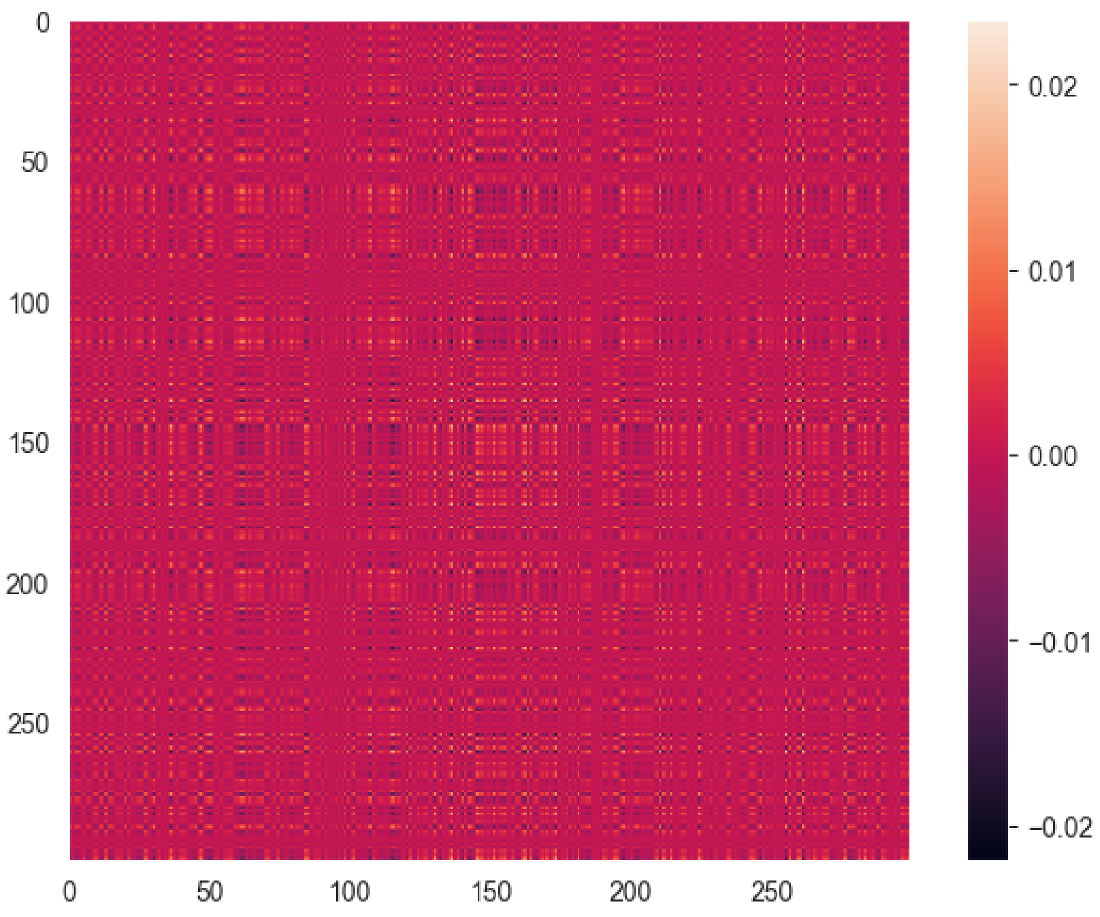
[7]:
```
projectors = pre.build_projectors(eigenvectors, 4)
```

[8]:
```
for projector in projectors:
    vg.plot_matrix(projector)
```


0
50
100
150
200
250
0
50
100
150
200
250
0.0037
0.0036
0.0035
0.0034
0.0033
0.0032
0.0031
0.0030

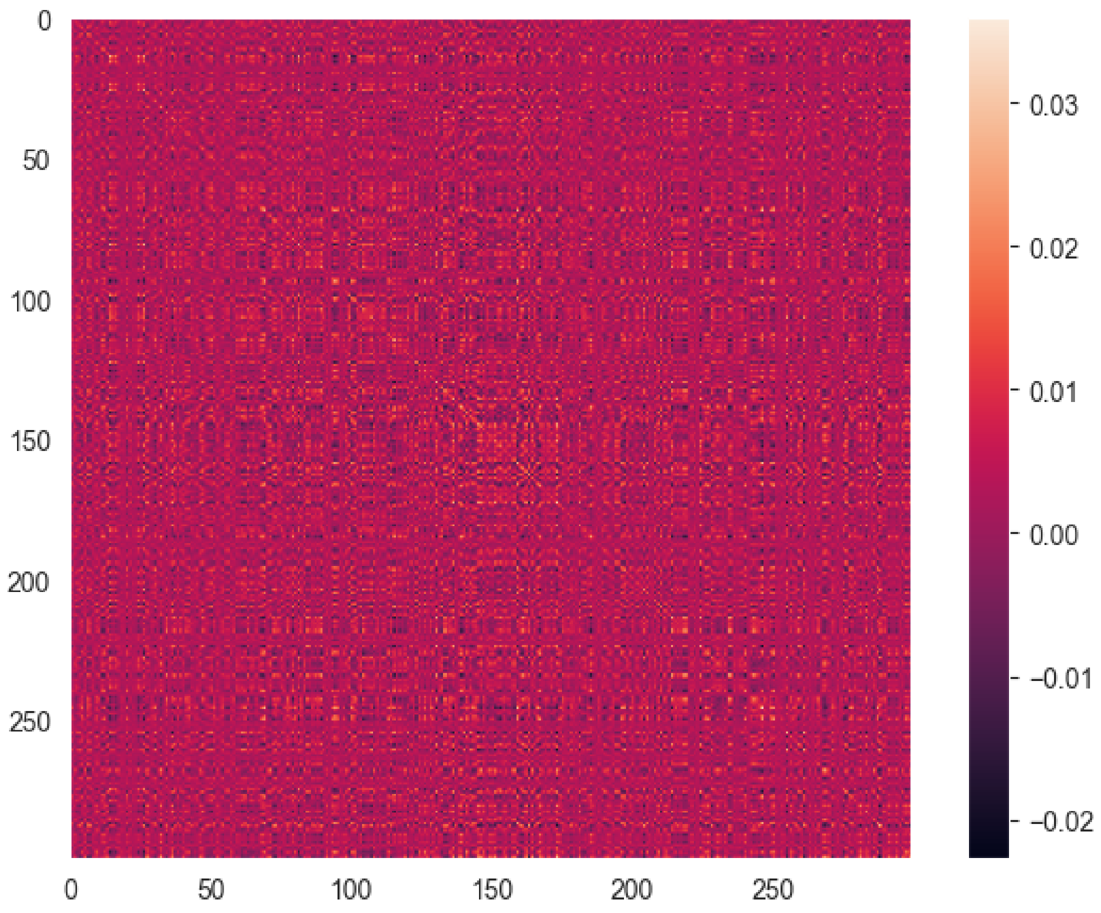

0
50
100
150
200
250
0
50
100
150
200
250
0.02
0.01
0.00
−0.01
−0.02


0
50
100
150
200
250
0
50
100
150
200
250
0.02
0.01
0.00
−0.01
−0.02



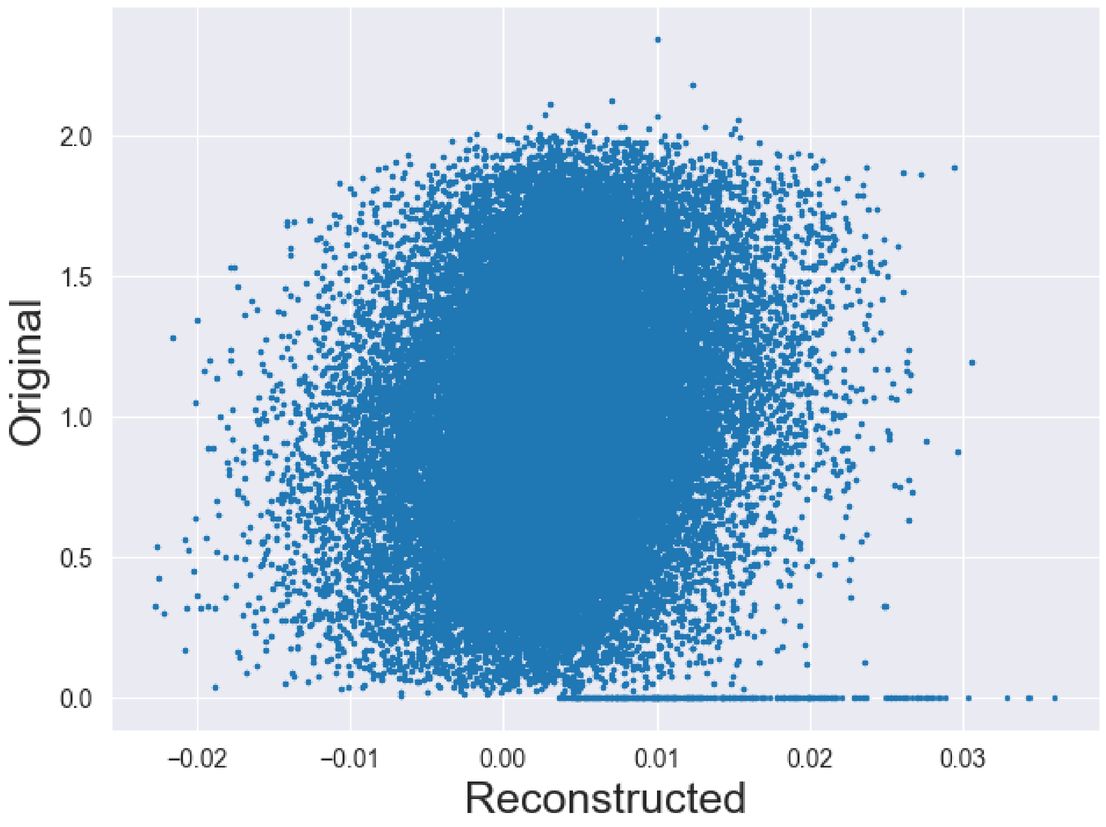
Finally it is possible to reconstruct the original matrix by adding together the projectors, that should be similar to the original one (in this case we used a random matrix so the behaviour should be different when using real data):

[9]:
```
reconstructed_matrix = pre.reconstruct_matrix(projectors, 4)
vg.plot_matrix(reconstructed_matrix)
```



It is also possible to compare the reconstructed and original matrix with a scatter plot and computing the pearson coefficient:

```
[14]: vg.scatter_plot(reconstructed_matrix,
                      normalized_adj,
                      'Reconstructed',
                      'Original',
                      )
      corr, _ = pearsonr(reconstructed_matrix.flatten(), normalized_adj.flatten())
      print(f'Pearson correlation coeffient: {corr}')
```



```
Pearson correlation coeffient: 0.177039794231876
```

```
[ ]:
```