Perform Network Analysis

- Create a covariance matrix
- Find an ideal noise-removing threshold filter with Random Matrix Theory
- Analyze properties of the network

Questions to answer with alpha and beta diversity analysis:

- Does the 3G and 3M look like 1C or 2C
- How do 1G and 1M compare to 1C
- How does 2G compare to 3G, 2M compare to 3M
- How does 1C compare to 2C

In [1]:
```python
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
```

Import Data

In [2]:
```python
# load data
df_counts_rel = pd.read_csv("data/FCF_relative_counts.csv", index_col=0)
df_annotations = pd.read_csv("data/FCF_annotations_corrected.csv", index_col=0)

print("df_counts_rel.shape", df_counts_rel.shape)
print("df_annotations.shape", df_annotations.shape)
```

```
df_counts_rel.shape (86, 141)
df_annotations.shape (86, 5)
```

In [3]:
```python
df_counts_rel
```

Out[3]:

	opitutus spp.	paludibacter propionicigenes	magnetospirillum sp.	rhodopseudomonas palustris	acetobacter spp.	bacteroides spp.	pleomorphomonas oryzae	afipia sp.	rhodoblast acidophil
C0C	0.207686	0.186672	0.111388	0.105284	0.074637	0.072618	0.061373	0.051297	0.0443
1C3A	0.266982	0.056341	0.189617	0.127545	0.012517	0.026242	0.050610	0.038828	0.0161
1C3B	0.290486	0.052059	0.163243	0.117691	0.011864	0.026510	0.041062	0.032812	0.0115
1C5A	0.194118	0.023516	0.294578	0.097890	0.008608	0.018172	0.031574	0.027593	0.0244
1C5B	0.187871	0.048505	0.132181	0.130452	0.011161	0.036060	0.049378	0.037943	0.0250
...	...	...	...	...	...	...	...	...	
3G7A	0.144459	0.001801	0.018378	0.065465	0.057839	0.393686	0.159824	0.005029	0.0430
3G7B	0.247657	0.000743	0.023841	0.051781	0.062985	0.275727	0.206768	0.004794	0.0267
3G7C	0.228160	0.000888	0.026917	0.059782	0.080850	0.262612	0.185294	0.004707	0.0299
3G10A	0.093492	0.001327	0.096390	0.089828	0.046889	0.383666	0.146778	0.008510	0.0512
3G10B	0.120237	0.000638	0.240036	0.053780	0.061816	0.286654	0.121156	0.005679	0.0169

86 rows × 141 columns

Split Samples Into Groups

In [4]:
```python
groups = set(df_annotations["group"])
groups
```

Out[4]:
```
{'1C', '1G', '1M', '2C', '2G', '2M', '3G', '3M', 'C0C'}
```

In [5]:
```python
df_counts_rel_1C = df_counts_rel.loc[df_annotations["group"] == "1C", :]
df_counts_rel_1G = df_counts_rel.loc[df_annotations["group"] == "1G", :]
df_counts_rel_1M = df_counts_rel.loc[df_annotations["group"] == "1M", :]
df_counts_rel_2C = df_counts_rel.loc[df_annotations["group"] == "2C", :]
df_counts_rel_2G = df_counts_rel.loc[df_annotations["group"] == "2G", :]
df_counts_rel_2M = df_counts_rel.loc[df_annotations["group"] == "2M", :]
df_counts_rel_3G = df_counts_rel.loc[df_annotations["group"] == "3G", :]
df_counts_rel_3M = df_counts_rel.loc[df_annotations["group"] == "3M", :]
```

In [6]:
```python
df_counts_rel_3M
```

Out[6]:

	opitutus spp.	paludibacter propionicigenes	magnetospirillum sp.	rhodopseudomonas palustris	acetobacter spp.	bacteroides spp.	pleomorphomonas oryzae	afipia sp.	rhodoblast acidophil
3M3A	0.174045	0.000404	0.033946	0.021195	0.000757	0.013949	0.061011	0.005201	0.6204
3M3B	0.135105	0.000885	0.407362	0.021515	0.001188	0.005499	0.099053	0.007420	0.1617
3M3C	0.084566	0.000922	0.508812	0.014735	0.001243	0.003717	0.044183	0.004214	0.1289
3M5B	0.171456	0.003128	0.024658	0.024319	0.012068	0.090322	0.199926	0.007409	0.1384
3M5C	0.149788	0.001990	0.057485	0.027828	0.001339	0.087199	0.072974	0.006518	0.4014
3M5A	0.157334	0.001036	0.061791	0.020911	0.001099	0.034594	0.130031	0.005369	0.3593
3M7A	0.203516	0.001522	0.034876	0.049155	0.009756	0.078834	0.123492	0.008863	0.2616
3M7B	0.094730	0.007476	0.024598	0.038245	0.029740	0.163735	0.192327	0.011687	0.0984
3M7C	0.195999	0.002399	0.069706	0.023421	0.023294	0.124625	0.112692	0.005227	0.1062
3M10A	0.121134	0.016020	0.119332	0.049653	0.090298	0.093914	0.127554	0.011773	0.1497
3M10B	0.125448	0.007026	0.265430	0.030463	0.030561	0.184910	0.113892	0.009670	0.0715
3M10C	0.149036	0.002733	0.201424	0.037578	0.031801	0.144593	0.084645	0.008691	0.1933

12 rows × 141 columns

Standardize Data For Correlation Analysis

Standardize relative abundance counts by OTU by subtracting the mean across all samples and dividing by variance across all samples. Use this standardized data matrix for all subsequent correlation analysis.

Source: Deng Paper

In [7]:
```python
df_counts_rel_1C_stan = df_counts_rel_1C.sub(df_counts_rel_1C.mean(0), axis=1).div(df_counts_rel_1C.std(0), axi
df_counts_rel_1G_stan = df_counts_rel_1G.sub(df_counts_rel_1G.mean(0), axis=1).div(df_counts_rel_1G.std(0), axi
df_counts_rel_1M_stan = df_counts_rel_1M.sub(df_counts_rel_1M.mean(0), axis=1).div(df_counts_rel_1M.std(0), axi
df_counts_rel_2C_stan = df_counts_rel_2C.sub(df_counts_rel_2C.mean(0), axis=1).div(df_counts_rel_2C.std(0), axi
df_counts_rel_2G_stan = df_counts_rel_2G.sub(df_counts_rel_2G.mean(0), axis=1).div(df_counts_rel_2G.std(0), axi
df_counts_rel_2M_stan = df_counts_rel_2M.sub(df_counts_rel_2M.mean(0), axis=1).div(df_counts_rel_2M.std(0), axi
df_counts_rel_3G_stan = df_counts_rel_3G.sub(df_counts_rel_3G.mean(0), axis=1).div(df_counts_rel_3G.std(0), axi
df_counts_rel_3M_stan = df_counts_rel_3M.sub(df_counts_rel_3M.mean(0), axis=1).div(df_counts_rel_3M.std(0), axi
```

In [8]:
```python
df_counts_rel_3M_stan
```

Out[8]:

	opitutus spp.	paludibacter propionicigenes	magnetospirillum sp.	rhodopseudomonas palustris	acetobacter spp.	bacteroides spp.	pleomorphomonas oryzae	afipia sp.	rhodoblas acidoph
3M3A	0.737828	-0.756187	-0.716109	-0.766586	-0.727594	-1.158250	-1.111845	-0.977458	2.448
3M3B	-0.318516	-0.648953	1.572562	-0.738538	-0.710806	-1.295065	-0.305736	-0.098996	-0.386
3M3C	-1.689498	-0.640769	2.194347	-1.334413	-0.708690	-1.323907	-1.468437	-1.368070	-0.589
3M5B	0.667591	-0.148796	-0.773035	-0.492098	-0.286830	0.078213	1.831747	-0.103407	-0.530
3M5C	0.079791	-0.402432	-0.571835	-0.183721	-0.704912	0.027657	-0.858348	-0.456020	1.094
3M5A	0.284504	-0.615221	-0.545445	-0.791596	-0.714287	-0.824016	0.350675	-0.910960	0.834
3M7A	1.537270	-0.506780	-0.710407	1.690675	-0.376939	-0.107779	0.212116	0.472312	0.230
3M7B	-1.413770	0.820874	-0.773399	0.731798	0.401823	1.266768	1.670731	1.590231	-0.777
3M7C	1.333359	-0.311347	-0.496935	-0.571038	0.150613	0.633580	-0.016741	-0.967293	-0.729
3M10A	-0.697500	2.725996	-0.192773	1.734422	2.761679	0.136367	0.298188	1.623975	-0.460
3M10B	-0.580466	0.720491	0.702663	0.047891	0.433822	1.609582	0.008699	0.791451	-0.943
3M10C	0.059406	-0.236876	0.310365	0.673204	0.482123	0.956849	-0.611050	0.404234	-0.191

12 rows × 141 columns

In [9]:
```python
df_counts_rel_1C_stan.mean(0)
```

Out[9]:
```
opitutus spp.                    2.886580e-16
paludibacter propionicigenes     1.665335e-16
magnetospirillum sp.            -1.110223e-17
rhodopseudomonas palustris       0.000000e+00
acetobacter spp.                -1.665335e-16
                                     ...
prevotella nanceiensis                    NaN
lactobacillus helveticus                  NaN
bacillus thermoamylovorans       0.000000e+00
bosea spp.                                NaN
corynebacterium durum            3.330669e-17
Length: 141, dtype: float64
```

In [10]:
```python
df_counts_rel_1C_stan.var(0)
```

Out[10]:
```
opitutus spp.                   1.0
paludibacter propionicigenes    1.0
magnetospirillum sp.            1.0
rhodopseudomonas palustris      1.0
acetobacter spp.                1.0
                               ...
prevotella nanceiensis          NaN
lactobacillus helveticus        NaN
bacillus thermoamylovorans      1.0
bosea spp.                      NaN
corynebacterium durum           1.0
Length: 141, dtype: float64
```

Network Construction

- create a pair-wise similarity (Pearson) of abundance across different samples
- determine adjacency matrix by RMT-based approach

In [11]:
```python
import network_analysis_thesholding as nat
```

Use Random Matrix Theory To Threshold

Good overview of Random Matrix Theory by Torsten Scholak

TODO: figure out if my Poisson distribution is very off... my chi_squared value seems unreal...

In [12]:
```python
# nat.find_RMT_threshold(df_counts_rel_1G_stan, s_tb=0.3, alpha=0.5)
```

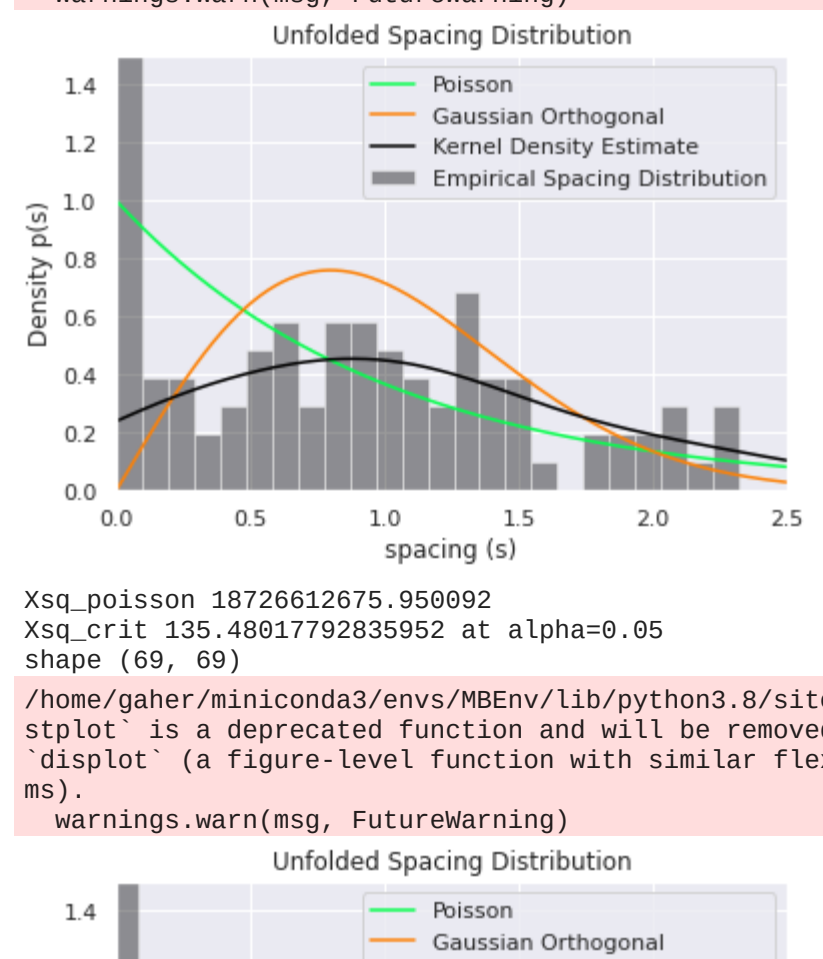
Data Is Too Messy

Since the data is too messy, this code lets you eyeball the threshold where the data fits the Poisson distribution more than it fits the GOE distribution.

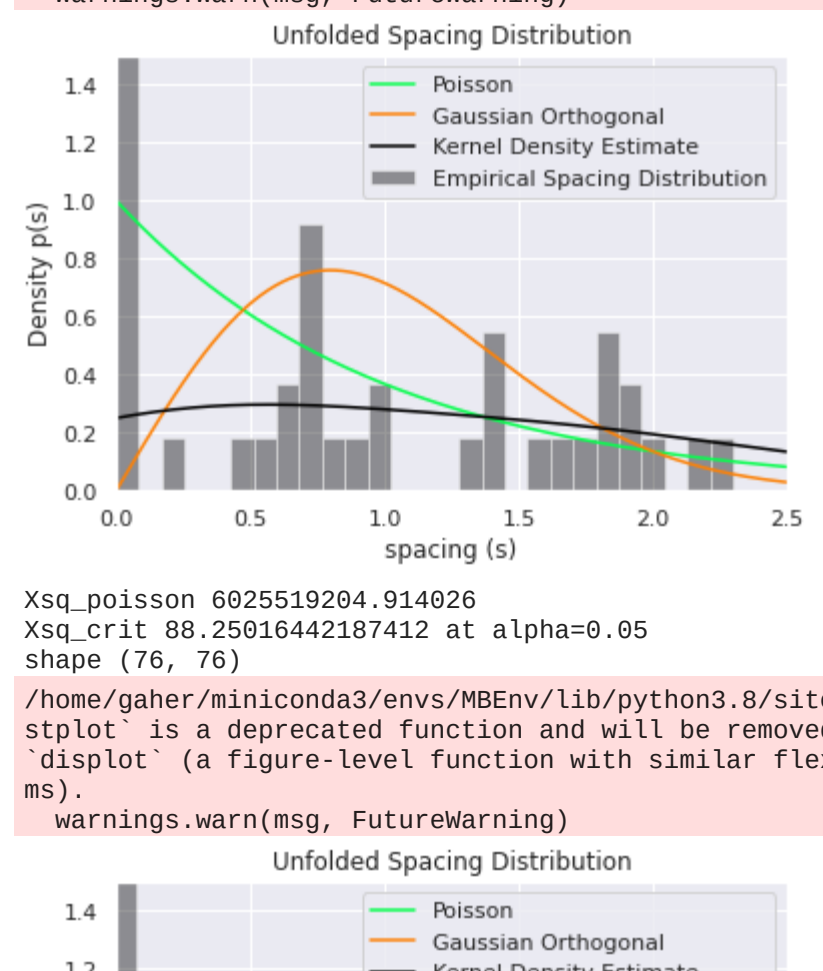
In [13]:
```python
# threshold = 0.03
# nat.visualize_RMT_threshold(df_counts_rel_1C_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_1G_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_1M_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_2C_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_2G_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_2M_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_3G_stan, threshold)
# nat.visualize_RMT_threshold(df_counts_rel_3M_stan, threshold)
```

In [14]:
```python
threshold = 0.76
nat.visualize_RMT_threshold(df_counts_rel_1C_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_1G_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_1M_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_2C_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_2G_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_2M_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_3G_stan, threshold)
nat.visualize_RMT_threshold(df_counts_rel_3M_stan, threshold)
```

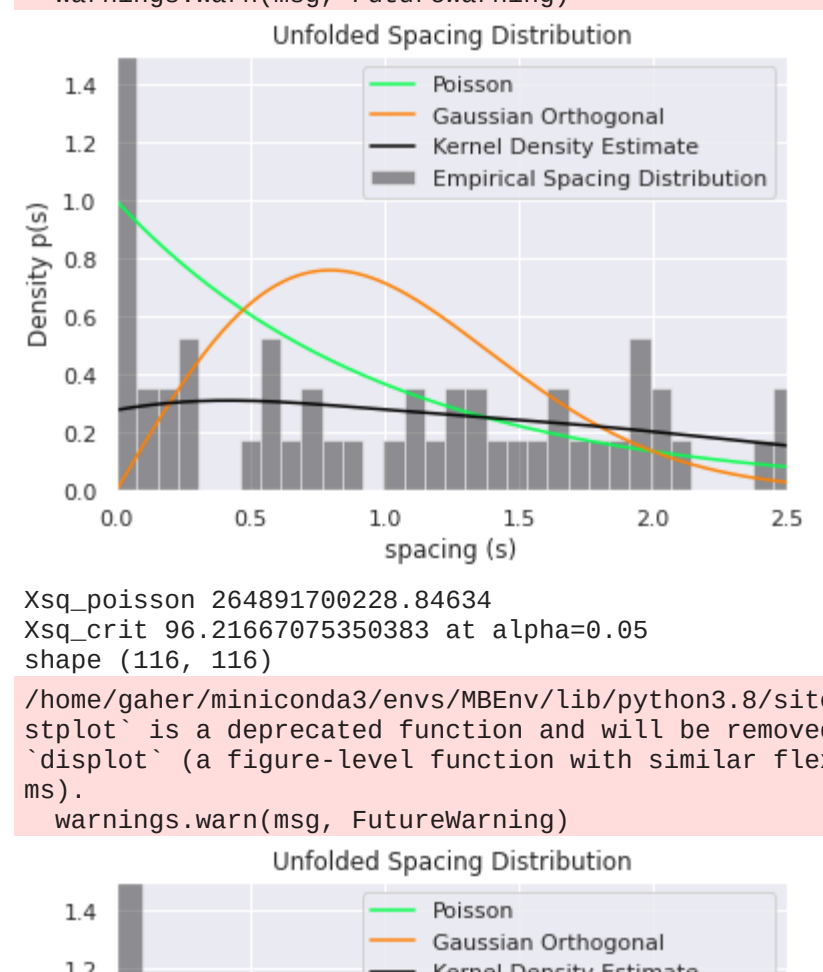
```
shape (111, 111)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



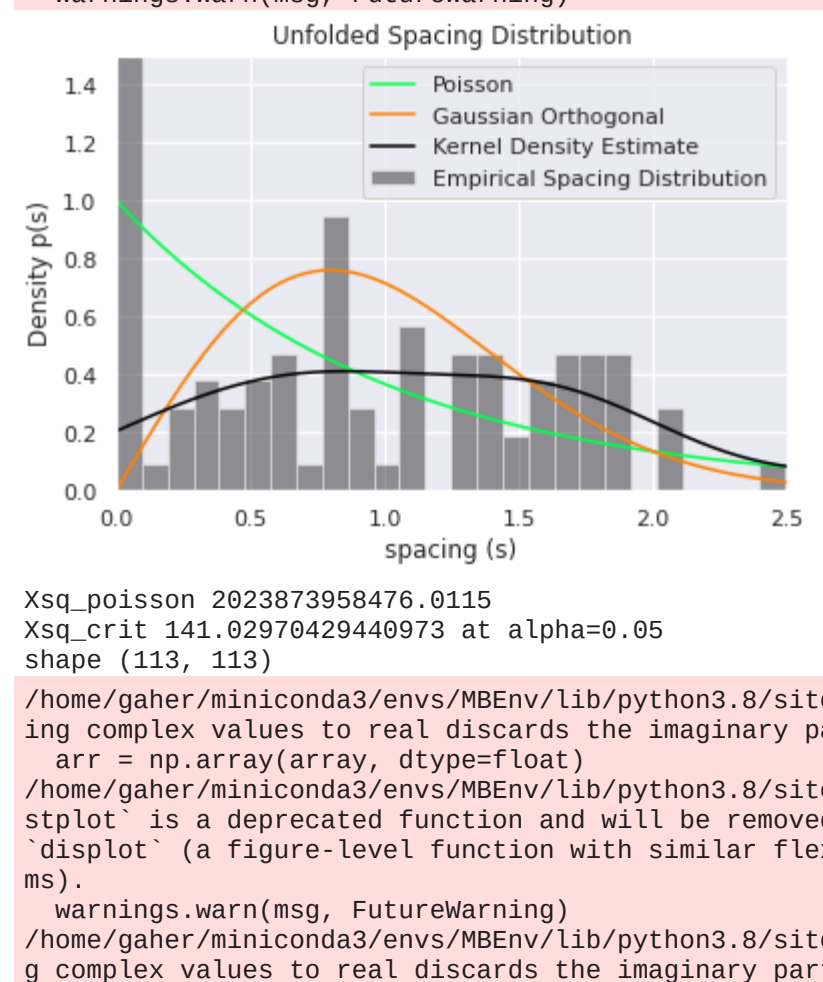

```
Xsq_poisson 18726612675.950092
Xsq_crit 135.48017792835952 at alpha=0.05
shape (69, 69)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



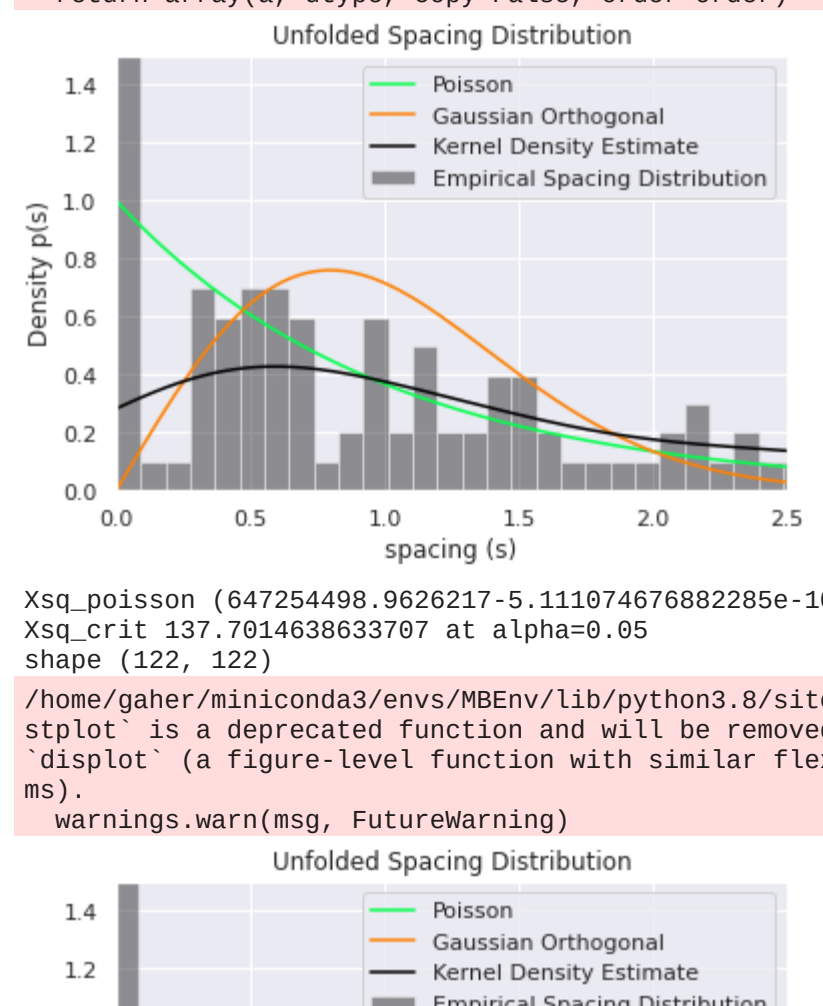

```
Xsq_poisson 6025519204.914026
Xsq_crit 88.25016442187412 at alpha=0.05
shape (76, 76)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



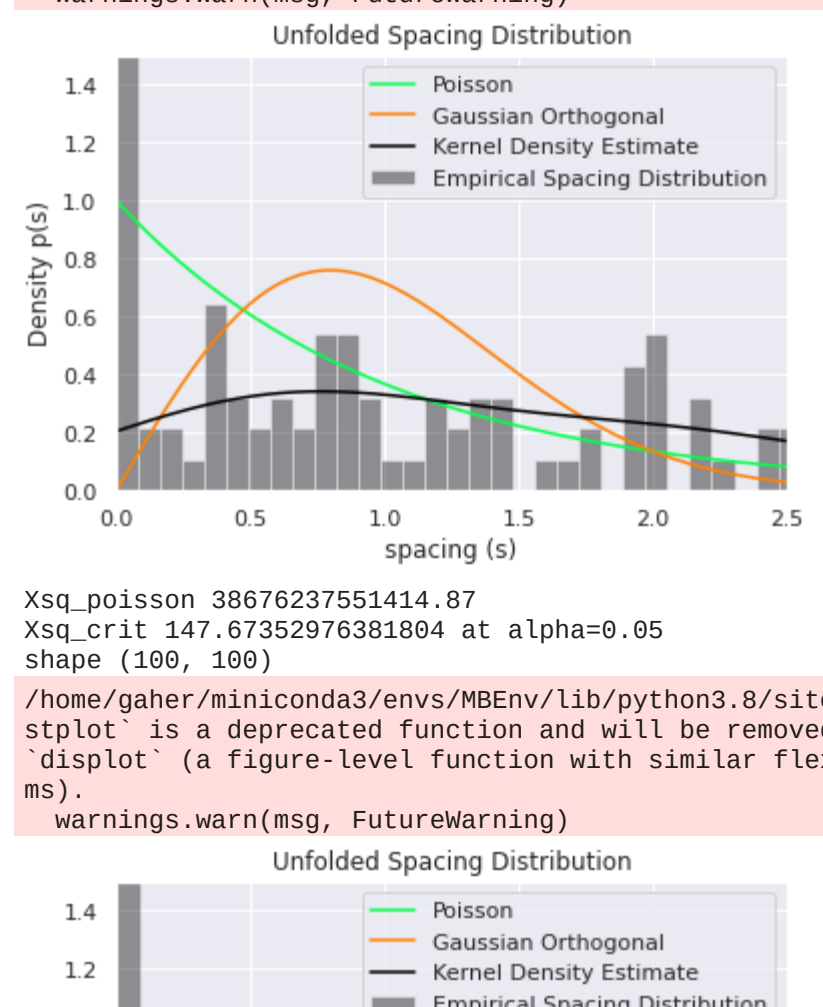

```
Xsq_poisson 264891700228.84634
Xsq_crit 96.21667075350383 at alpha=0.05
shape (116, 116)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



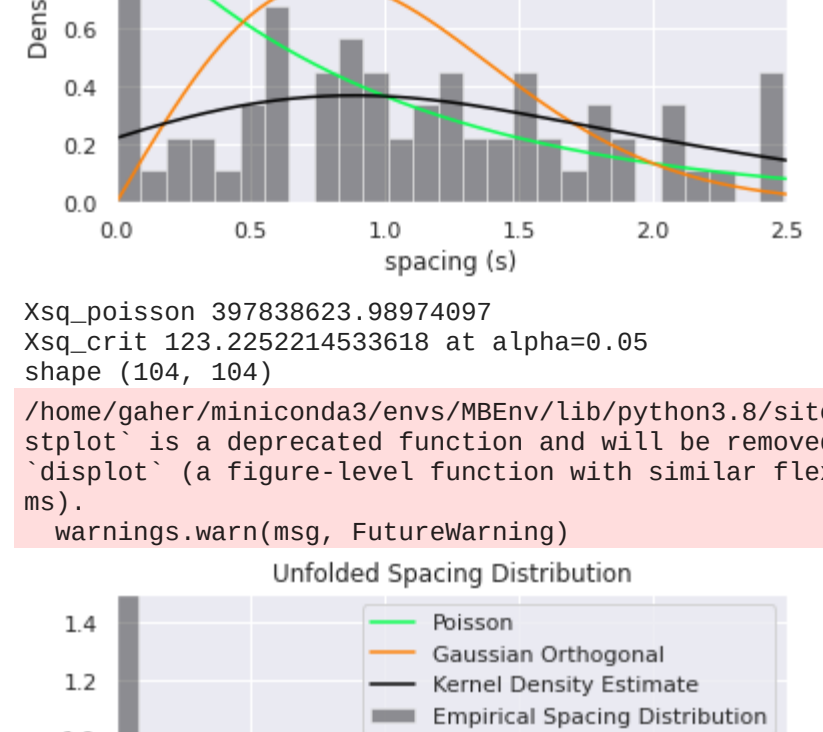

```
Xsq_poisson 20238739584 76.0115
Xsq_crit 141.02970429440973 at alpha=0.05
shape (113, 113)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/empyricalRMT/_validate.py:9: ComplexWarning: Casting complex values to real discards the imaginary part
  arr = np.array(array, dtype=float)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/numpy/core/_asarray.py:85: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```




```
Xsq_poisson (647254498.9626217-5.111074676882285e-10j)
Xsq_crit 137.7014638633707 at alpha=0.05
shape (122, 122)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
Xsq_poisson 38676237551414.87
Xsq_crit 147.67352976381804 at alpha=0.05
shape (100, 100)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```




```
Xsq_poisson 397838623.98974097
Xsq_crit 123.2252214533618 at alpha=0.05
shape (104, 104)
/home/gaher/miniconda3/envs/MBEnv/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```


```
Xsq_poisson 8376307573.361557
Xsq_crit 127.68930826333825 at alpha=0.05
```

In [15]:
```python
df_adjm_1C = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1C), threshold)
df_adjm_1G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1G), threshold)
df_adjm_1M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1M), threshold)
df_adjm_2C = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2C), threshold)
df_adjm_2G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2G), threshold)
df_adjm_2M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2M), threshold)
df_adjm_3G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_3G), threshold)
df_adjm_3M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_3M), threshold)
```

```
shape (111, 111)
shape (69, 69)
shape (76, 76)
shape (116, 116)
shape (113, 113)
shape (122, 122)
shape (100, 100)
shape (104, 104)
```

In [16]:
```python
df_adjm_1C = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1C_stan), threshold)
df_adjm_1G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1G_stan), threshold)
df_adjm_1M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_1M_stan), threshold)
df_adjm_2C = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2C_stan), threshold)
df_adjm_2G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2G_stan), threshold)
df_adjm_2M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_2M_stan), threshold)
df_adjm_3G = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_3G_stan), threshold)
df_adjm_3M = nat.get_thresholded_matrix(nat.get_similarity_matrix(df_counts_rel_3M_stan), threshold)
```

```
shape (111, 111)
shape (69, 69)
shape (76, 76)
shape (116, 116)
shape (113, 113)
shape (122, 122)
shape (100, 100)
shape (104, 104)
```

In [17]:
```python
# save adjacency matrices
df_adjm_1C.to_csv(f"data/adjm/df_adjm_1C_{round(threshold*100)}.csv")
df_adjm_1G.to_csv(f"data/adjm/df_adjm_1G_{round(threshold*100)}.csv")
df_adjm_1M.to_csv(f"data/adjm/df_adjm_1M_{round(threshold*100)}.csv")
df_adjm_2C.to_csv(f"data/adjm/df_adjm_2C_{round(threshold*100)}.csv")
df_adjm_2G.to_csv(f"data/adjm/df_adjm_2G_{round(threshold*100)}.csv")
df_adjm_2M.to_csv(f"data/adjm/df_adjm_2M_{round(threshold*100)}.csv")
df_adjm_3G.to_csv(f"data/adjm/df_adjm_3G_{round(threshold*100)}.csv")
df_adjm_3M.to_csv(f"data/adjm/df_adjm_3M_{round(threshold*100)}.csv")
```

In [18]:
```python
df_adjm_3M
```

Out[18]:

	opitutus spp.	paludibacter propionicigenes	magnetospirillum sp.	rhodopseudomonas palustris	acetobacter spp.	bacteroides spp.	pleomorphomonas oryzae	afipia sp.
opitutus spp.	0	0	0	0	0	0	0	0
paludibacter propionicigenes	0	0	0	0	1	0	0	1
magnetospirillum sp.	0	0	0	0	0	0	0	0
rhodopseudomonas palustris	0	0	0	0	0	0	0	1
acetobacter spp.	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
burkholderia ubonensis	0	0	0	0	0	0	0	0
aeromonas spp.	0	0	0	0	0	0	0	0
methylocystis sp.	0	0	0	0	0	0	0	0
sphingomonas faeni	0	0	0	0	0	0	0	0
lactobacillus helveticus	0	0	0	0	0	0	0	0

104 rows × 104 columns

Analyze Network Properties:

Key Characteristics:

- k-core, average path length, local clustering coefficient
- the average degree, degree distribution
- module, clique, motif

Density: tells you how connected a network is. A 0 density would mean that there are no connections at all, and a 1 would indicate that all possible edges are present (a perfectly connected network).

Transitivity: like density, expresses how interconnected a graph is in terms of a ratio of actual over possible connections. A triangle is formed if two nodes that are connected to a node are also connected to each other.

There are many network metrics derived from shortest path lengths. One such measure is **diameter**, which is the longest of all shortest paths. After calculating all shortest paths between every possible pair of nodes in the network, diameter is the length of the path between the two nodes that are furthest apart. The measure is designed to give you a sense of the network's overall size, the distance from one end of the network to another.

After getting some basic measures of the entire network structure, a good next step is to find which nodes are the most important ones in your network. In network analysis, measures of the importance of nodes are referred to as centrality measures. Because there are many ways of approaching the question "Which nodes are the most important?" there are many different ways of calculating centrality. Here you'll learn about three of the most common centrality measures: degree, betweenness centrality, and eigenvector centrality.

Features of Nodes:

- degree, degree centrality
- node-betweenness, node-betweenness centrality
- closeness, cluster coefficient, coreness
- eigenvector centrality

Degree Distribution: Most networks will have just a few hubs of very high degree, with the rest of similar, much lower degree

Eigenvector centrality: is a kind of extension of degree—it looks at a combination of a node's edges and the edges of that node's neighbors. Eigenvector centrality cares if you are a hub, but it also cares how many hubs you are connected to. It's calculated as a value from 0 to 1: the closer to one, the greater the centrality. Eigenvector centrality is useful for understanding which nodes can get information to many other nodes quickly. If you know a lot of well-connected people, you could spread a message very efficiently. If you've used Google, then you're already somewhat familiar with Eigenvector centrality. Their PageRank algorithm uses an extension of this formula to decide which webpages get to the top of its search results.

Betweenness centrality: is a bit different from the other two measures in that it doesn't care about the number of edges any one node or set of nodes has. Betweenness centrality looks at all the shortest paths that pass through a particular node (see above). To do this, it must first calculate every possible shortest path in your network, so keep in mind that betweenness centrality will take longer to calculate than other centrality measures (but it won't be an issue in a dataset of this size). Betweenness centrality, which is also expressed on a scale of 0 to 1, is fairly good at finding nodes that connect two otherwise disparate parts of a network. If you're the only thing connecting two clusters, every communication between those clusters has to pass through you. In contrast to a hub, this sort of node is often referred to as a broker. Betweenness centrality is not the only way of finding brokerage (and other methods are more systematic), but it's a quick way of giving you a sense of which nodes are important not because they have lots of connections themselves but because they stand between groups, giving the network connectivity and cohesion.

Read More Here: Social Network Analysis: From Graph Theory to Applications with Python

---

Features of Edges:

- shortest path, distance
- edge-betweenness

Sources:

- https://programminghistorian.org/en/lessons/exploring-and-analyzing-network-data-with-python

---

Community detection with modularity

There are many ways of calculating communities, cliques, and clusters in your network, but the most popular method currently is modularity. Modularity is a measure of relative density in your network: a community (called a module or modularity class) has high density relative to other nodes within its module but low density with those outside. Modularity gives you an overall score of how fractious your network is, and that score can be used to partition the network and return the individual communities.

After all modules are separated, each node can be assigned a role based on its topological properties and the role of node i is characterized by its within-module connectivity (zi) and among-module connectivity (Pi)

If all links of node i only belong to its own module, Pi = 0.
If the links of node i are distributed evenly among modules, Pi --> 1.

Peripheral nodes: zi <= 2.5, Pi <= 0.62
Connector nodes: zi <= 2.5, Pi > 0.62
Module hub nodes: zi > 2.5, Pi <= 0.62
Network hub nodes: zi > 2.5, Pi > 0.62

Sources:

- https://www.researchgate.net/post/How_should_I_interpret_the_connectivity_measures_kTotal_kWithin_kOut_kDiff_in_WGCNA
- MENAP paper

In [19]:
```python
import network_analysis_properties as nap
import networkx as nx
```

In [20]:
```python
G_1C = nap.get_network_from_adjm(df_adjm_1C, name="1C")
G_1G = nap.get_network_from_adjm(df_adjm_1G, name="1G")
G_1M = nap.get_network_from_adjm(df_adjm_1M, name="1M")
G_2C = nap.get_network_from_adjm(df_adjm_2C, name="2C")
G_2G = nap.get_network_from_adjm(df_adjm_2G, name="2G")
G_2M = nap.get_network_from_adjm(df_adjm_2M, name="2M")
G_3G = nap.get_network_from_adjm(df_adjm_3G, name="3G")
G_3M = nap.get_network_from_adjm(df_adjm_3M, name="3M")

df_eigen_1C, df_eigengenes_1C = nap.get_leading_eigengenes(G_1C, df_counts_rel_1C_stan, set_node_module_members
df_eigen_1G, df_eigengenes_1G = nap.get_leading_eigengenes(G_1G, df_counts_rel_1G_stan, set_node_module_members
df_eigen_1M, df_eigengenes_1M = nap.get_leading_eigengenes(G_1M, df_counts_rel_1M_stan, set_node_module_members
df_eigen_2C, df_eigengenes_2C = nap.get_leading_eigengenes(G_2C, df_counts_rel_2C_stan, set_node_module_members
df_eigen_2G, df_eigengenes_2G = nap.get_leading_eigengenes(G_2G, df_counts_rel_2G_stan, set_node_module_members
df_eigen_2M, df_eigengenes_2M = nap.get_leading_eigengenes(G_2M, df_counts_rel_2M_stan, set_node_module_members
df_eigen_3G, df_eigengenes_3G = nap.get_leading_eigengenes(G_3G, df_counts_rel_3G_stan, set_node_module_members
df_eigen_3M, df_eigengenes_3M = nap.get_leading_eigengenes(G_3M, df_counts_rel_3M_stan, set_node_module_members

Gs = [G_1C, G_1G, G_1M, G_2C, G_2G, G_2M, G_3G, G_3M]
```

```
setting node_module_membership for 1C graph
setting node_module_membership for 1G graph
setting node_module_membership for 1M graph
setting node_module_membership for 2C graph
setting node_module_membership for 2G graph
setting node_module_membership for 2M graph
setting node_module_membership for 3G graph
setting node_module_membership for 3M graph
```

In [21]:
```python
# save network properties to csv
nap.networks_attributes_to_csv(Gs, f"data/network/df_network_properties_{round(threshold*100)}.csv")
```

Out[21]:

	number_of_nodes	number_of_edges	density	transitivity	network_is_connected	network_diameter_of_largest_component	modularity_s
1C	111.0	408.0	0.066830	0.615015	False	11.0	0.59
1G	69.0	211.0	0.089940	0.683778	False	6.0	0.47
1M	76.0	193.0	0.067719	0.757720	False	12.0	0.68
2C	116.0	809.0	0.121289	0.656939	True	7.0	0.49
2G	113.0	391.0	0.061789	0.661108	True	13.0	0.71
2M	122.0	841.0	0.113941	0.803094	False	11.0	0.53
3G	100.0	238.0	0.048081	0.512983	False	10.0	0.63
3M	104.0	313.0	0.058439	0.633424	False	11.0	0.65

In [22]:
```python
# # save node properties to csv
# nap.nodes_attributes_to_csv(G_1C, f"data/network/df_network_1C_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_1G, f"data/network/df_network_1G_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_1M, f"data/network/df_network_1M_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_2C, f"data/network/df_network_2C_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_2G, f"data/network/df_network_2G_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_2M, f"data/network/df_network_2M_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_3G, f"data/network/df_network_3G_{round(threshold*100)}.csv")
# nap.nodes_attributes_to_csv(G_3M, f"data/network/df_network_3M_{round(threshold*100)}.csv")
```

In [23]:
```python
# # save module eigenvalues and eigengenes to csv
# df_eigen_1C.to_csv(f"data/network_module_eig/df_module_eigenvalues_1C_{round(threshold*100)}.csv")
# df_eigen_1G.to_csv(f"data/network_module_eig/df_module_eigenvalues_1G_{round(threshold*100)}.csv")
# df_eigen_1M.to_csv(f"data/network_module_eig/df_module_eigenvalues_1M_{round(threshold*100)}.csv")
# df_eigen_2C.to_csv(f"data/network_module_eig/df_module_eigenvalues_2C_{round(threshold*100)}.csv")
# df_eigen_2G.to_csv(f"data/network_module_eig/df_module_eigenvalues_2G_{round(threshold*100)}.csv")
# df_eigen_2M.to_csv(f"data/network_module_eig/df_module_eigenvalues_2M_{round(threshold*100)}.csv")
# df_eigen_3G.to_csv(f"data/network_module_eig/df_module_eigenvalues_3G_{round(threshold*100)}.csv")
# df_eigen_3M.to_csv(f"data/network_module_eig/df_module_eigenvalues_3M_{round(threshold*100)}.csv")

# df_eigengenes_1C.to_csv(f"data/network_module_eig/df_module_eigengenes_1C_{round(threshold*100)}.csv")
# df_eigengenes_1G.to_csv(f"data/network_module_eig/df_module_eigengenes_1G_{round(threshold*100)}.csv")
# df_eigengenes_1M.to_csv(f"data/network_module_eig/df_module_eigengenes_1M_{round(threshold*100)}.csv")
# df_eigengenes_2C.to_csv(f"data/network_module_eig/df_module_eigengenes_2C_{round(threshold*100)}.csv")
# df_eigengenes_2G.to_csv(f"data/network_module_eig/df_module_eigengenes_2G_{round(threshold*100)}.csv")
# df_eigengenes_2M.to_csv(f"data/network_module_eig/df_module_eigengenes_2M_{round(threshold*100)}.csv")
# df_eigengenes_3G.to_csv(f"data/network_module_eig/df_module_eigengenes_3G_{round(threshold*100)}.csv")
# df_eigengenes_3M.to_csv(f"data/network_module_eig/df_module_eigengenes_3M_{round(threshold*100)}.csv")
```

In [24]:
```python
print("list of node attributes:", set([k for n in G_1C.nodes for k in G_1C.nodes[n].keys()]))

# attribute = "degree"
# nap.display_sorted_property(G_1C, attribute=attribute)
# nap.display_sorted_property(G_1G, attribute=attribute)
# nap.display_sorted_property(G_1M, attribute=attribute)
# nap.display_sorted_property(G_2C, attribute=attribute)
# nap.display_sorted_property(G_2G, attribute=attribute)
# nap.display_sorted_property(G_2M, attribute=attribute)
# nap.display_sorted_property(G_3G, attribute=attribute)
# nap.display_sorted_property(G_3M, attribute=attribute)
```

```
list of node attributes: {'among_module_connectivity', 'within_module_degree', 'eigenvector_centrality', 'node_
module_membership', 'degree', 'within_module_connectivity', 'node_role', 'modularity', 'betweenness'}
```

Visualize Network

Default Layout

In [25]:
```python
# for G in Gs:
#     nap.visualize_network(G)
```

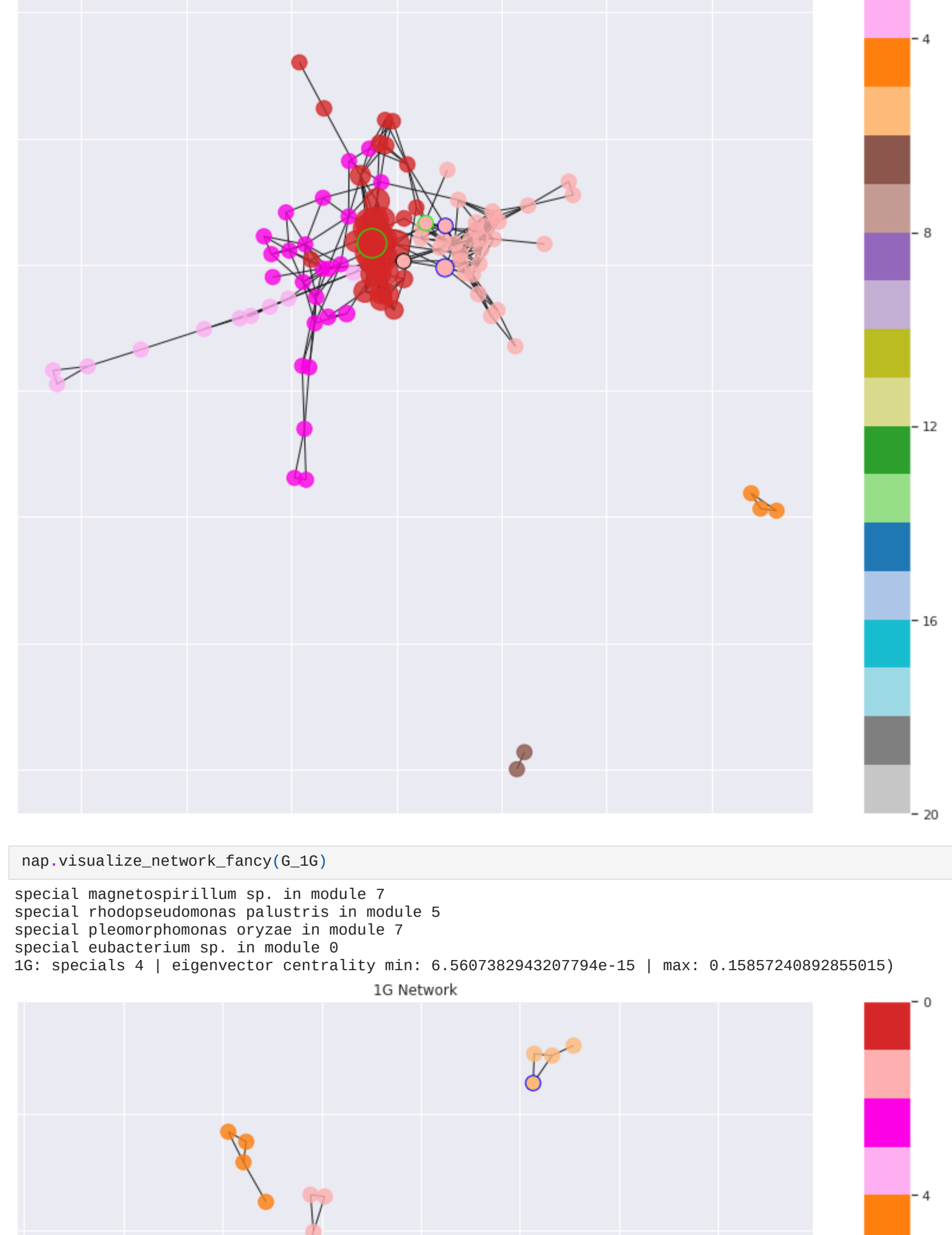
Try Circular Layout

In [26]:
```python
# for G in Gs:
#     nap.visualize_network(G, layout="circular")
```

Fancy Styling

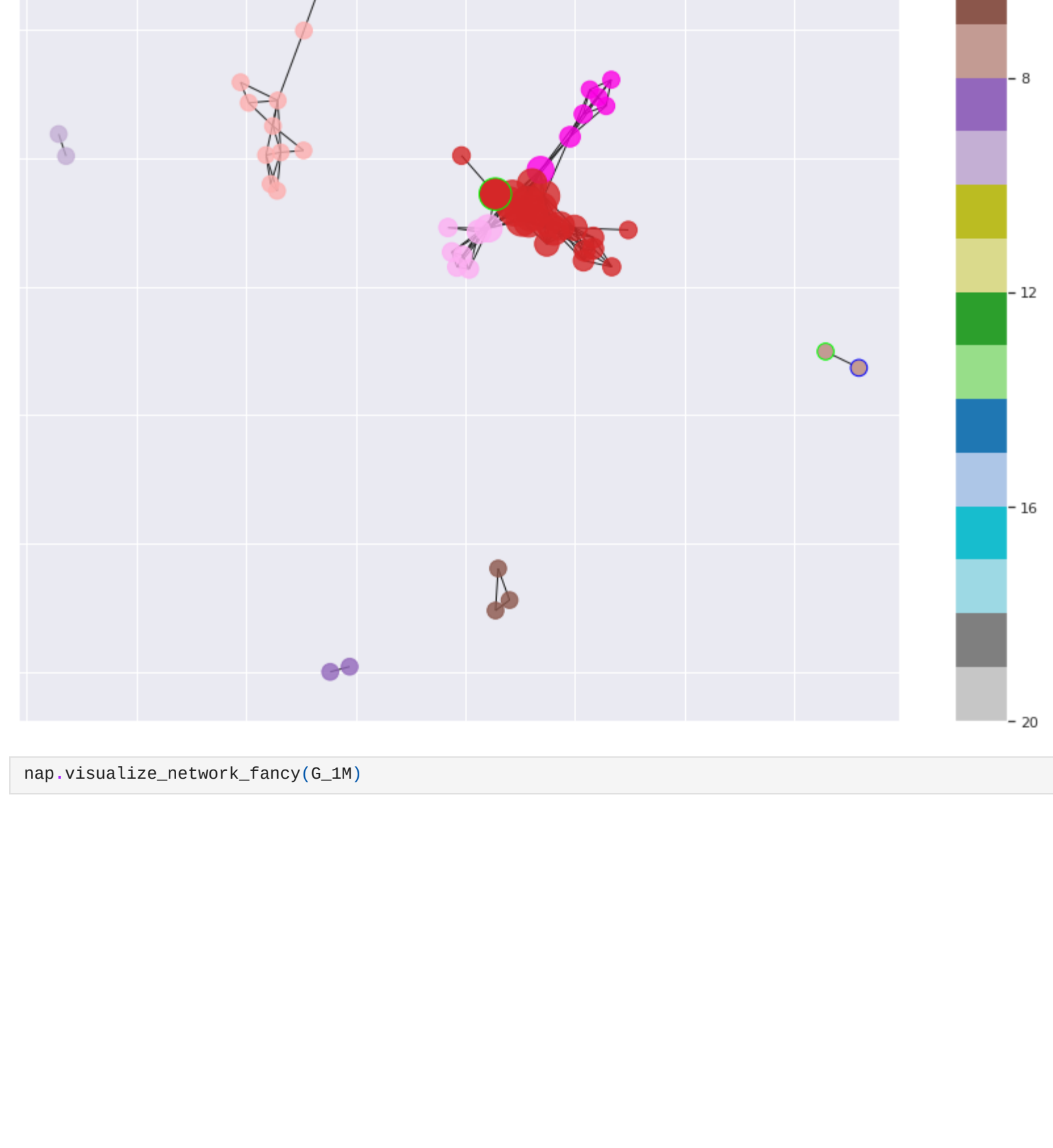
In [45]:
```python
nap.visualize_network_fancy(G_1C)
```

```
special opitutus spp. in module 1
special magnetospirillum sp. in module 1
special rhodopseudomonas palustris in module 1
special pleomorphomonas oryzae in module 1
special eubacterium sp. in module 0
1C: specials 5 | eigenvector centrality min: 8.90631887421286e-30 | max: 0.26240765238577135)
```



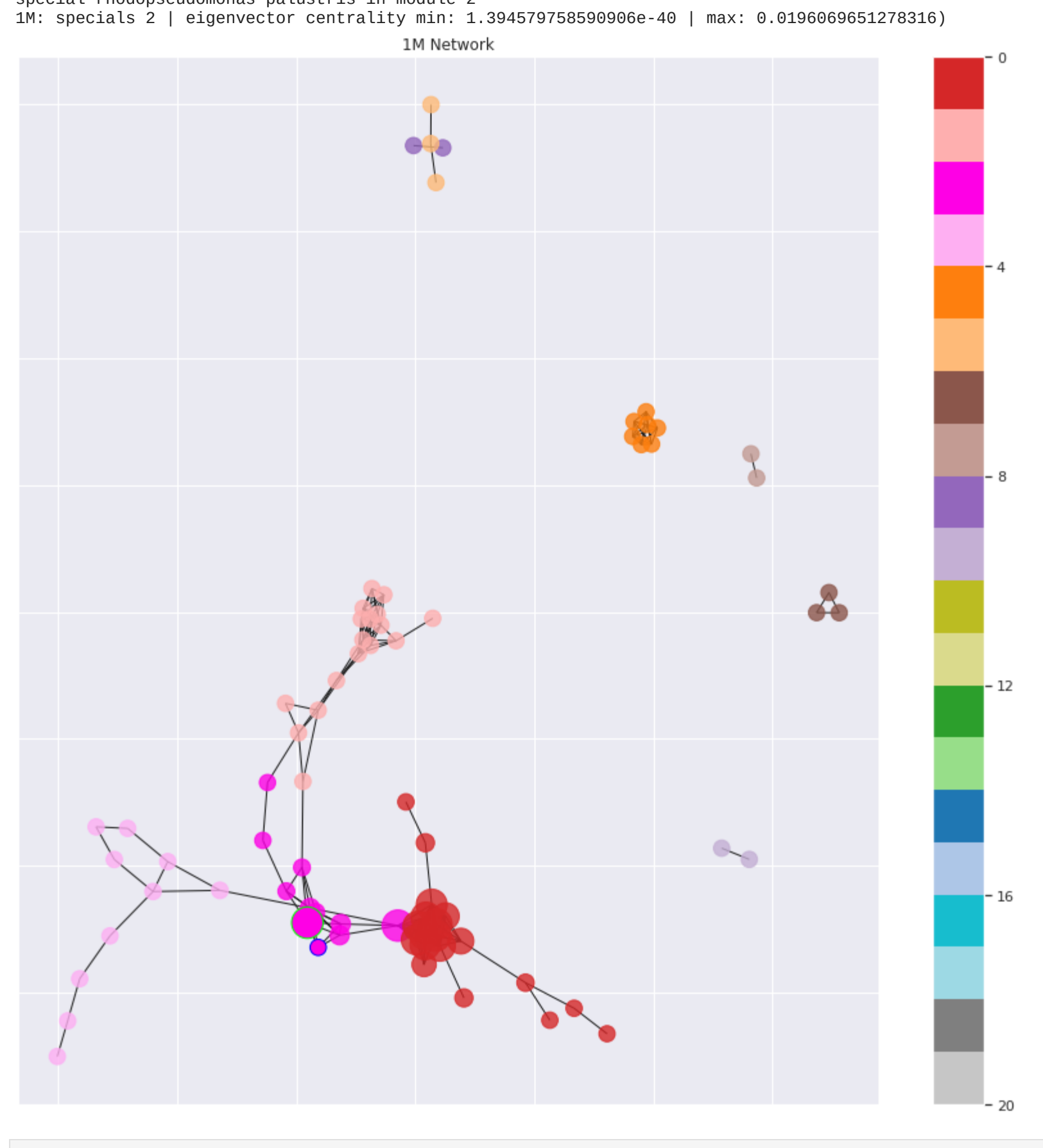
In [46]:
```python
nap.visualize_network_fancy(G_1G)
```

```
special magnetospirillum sp. in module 7
special rhodopseudomonas palustris in module 5
special pleomorphomonas oryzae in module 7
special eubacterium sp. in module 0
1G: specials 4 | eigenvector centrality min: 6.5607382943207794e-15 | max: 0.15857240892855015)
```



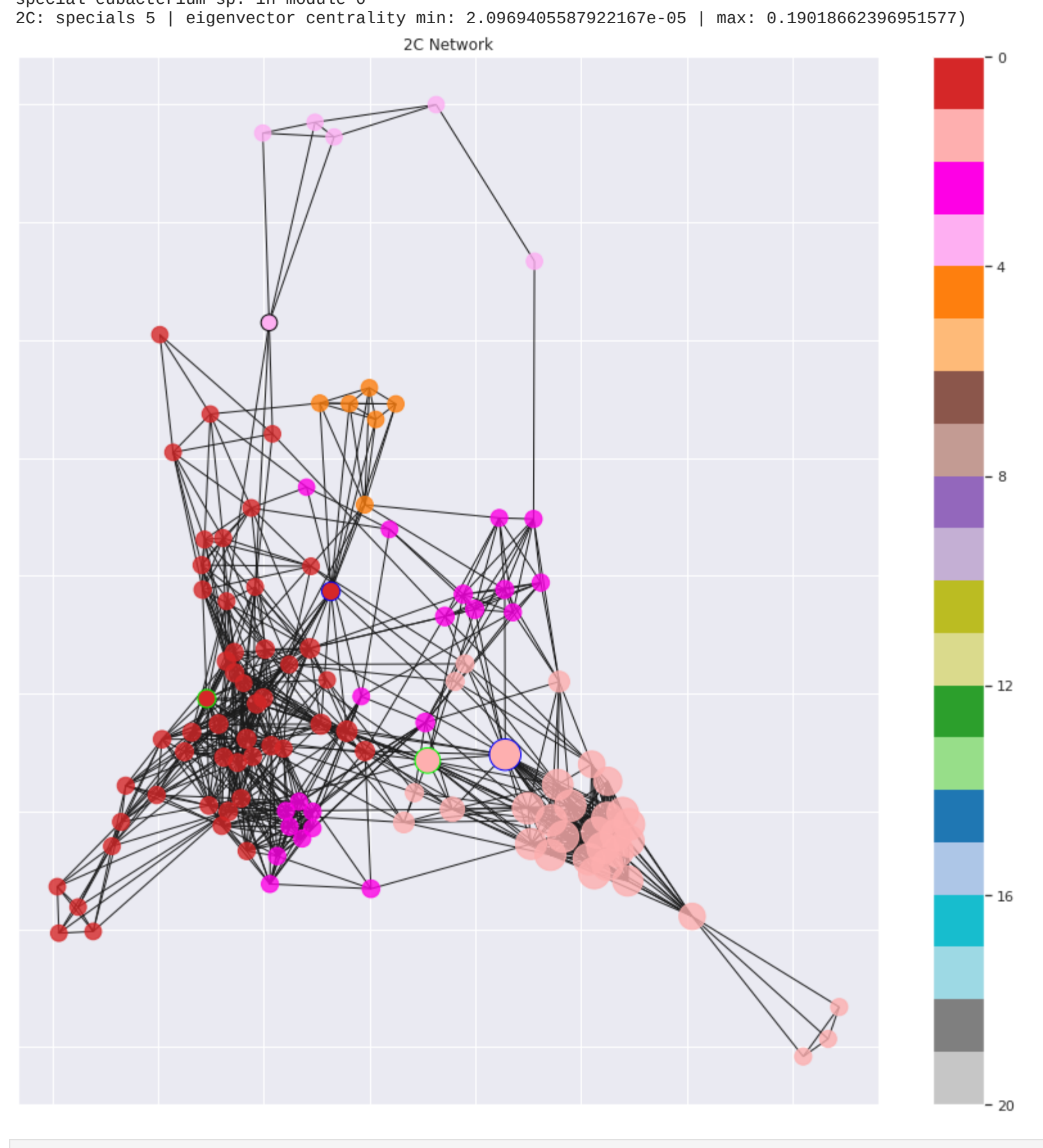
In [63]:
```python
nap.visualize_network_fancy(G_1M)
```

```
special magnetospirillum sp. in module 2
special rhodopseudomonas palustris in module 2
1M: specials 2 | eigenvector centrality min: 1.394579758590906e-40 | max: 0.0196069651278316)
```



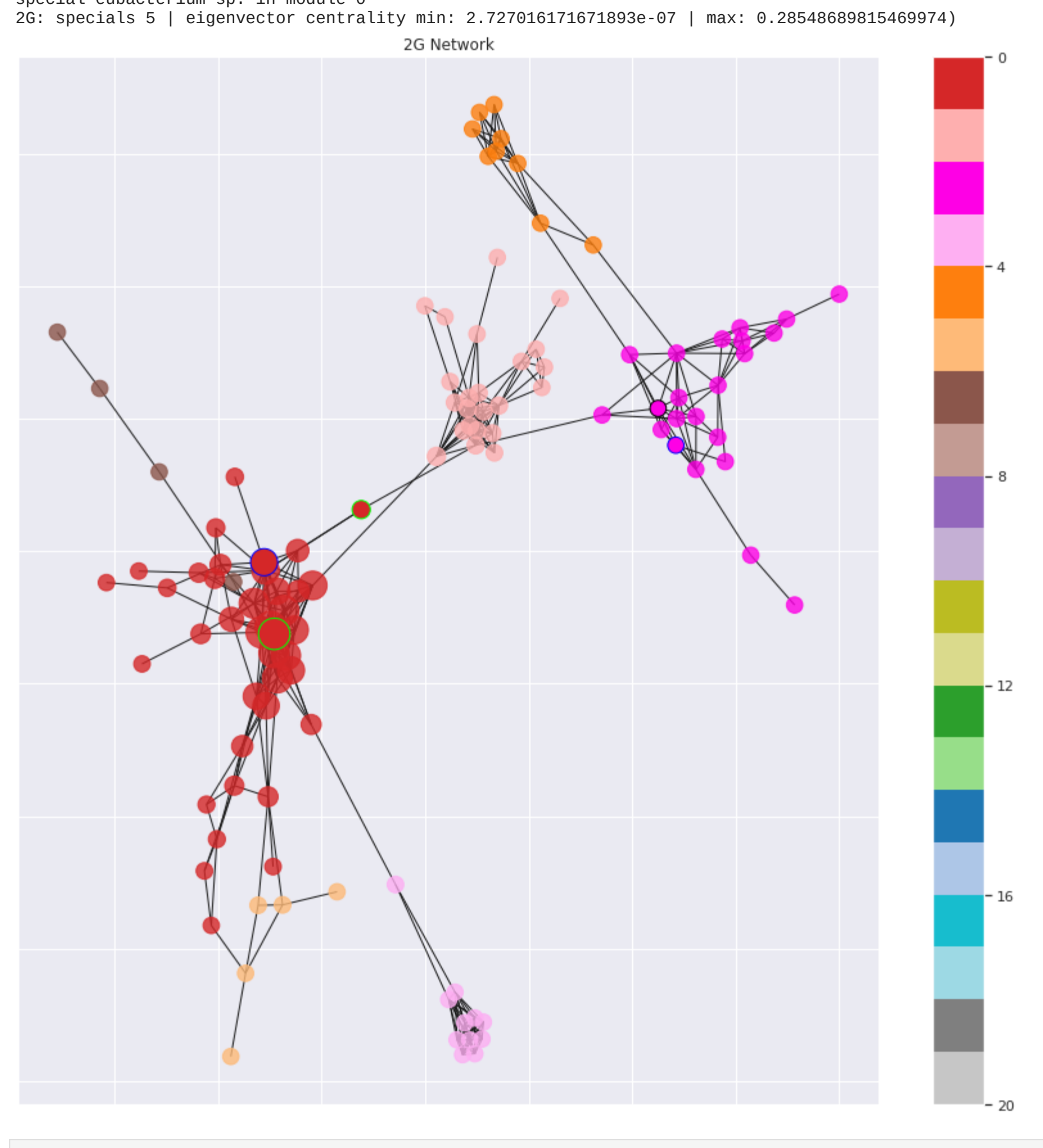
In [64]:
```
nap.visualize_network_fancy(G_2C)
```

```
special opitutus spp. in module 3
special magnetospirillum sp. in module 1
special rhodopseudomonas palustris in module 0
special pleomorphomonas oryzae in module 1
special eubacterium sp. in module 0
2C: specials 5 | eigenvector centrality min: 2.0969405587922167e-05 | max: 0.19018662396951577)
```



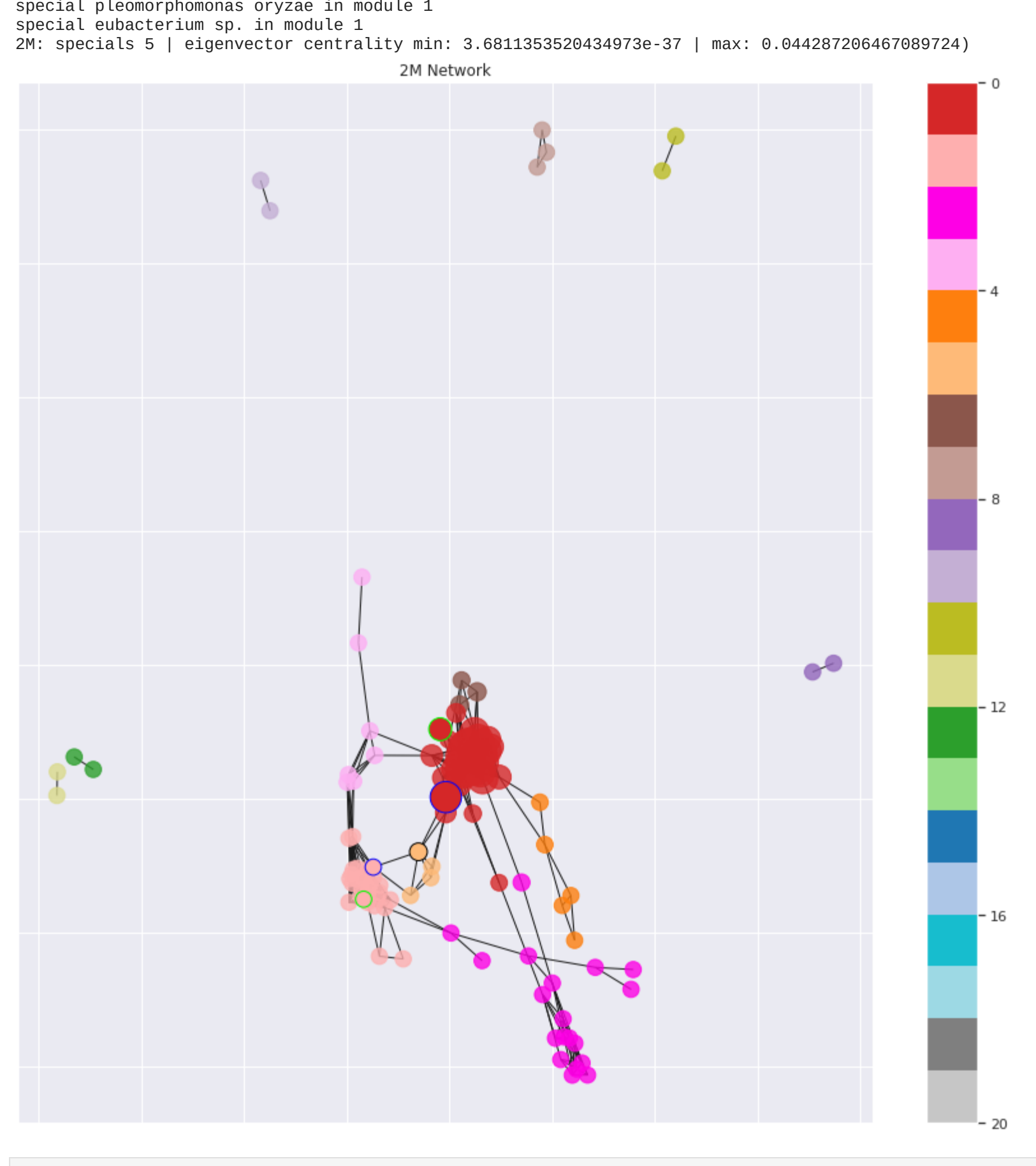
In [67]:
```
nap.visualize_network_fancy(G_2G)
```

```
special opitutus spp. in module 2
special magnetospirillum sp. in module 0
special rhodopseudomonas palustris in module 0
special pleomorphomonas oryzae in module 2
special eubacterium sp. in module 0
2G: specials 5 | eigenvector centrality min: 2.727016171671893e-07 | max: 0.28548689815469974)
```



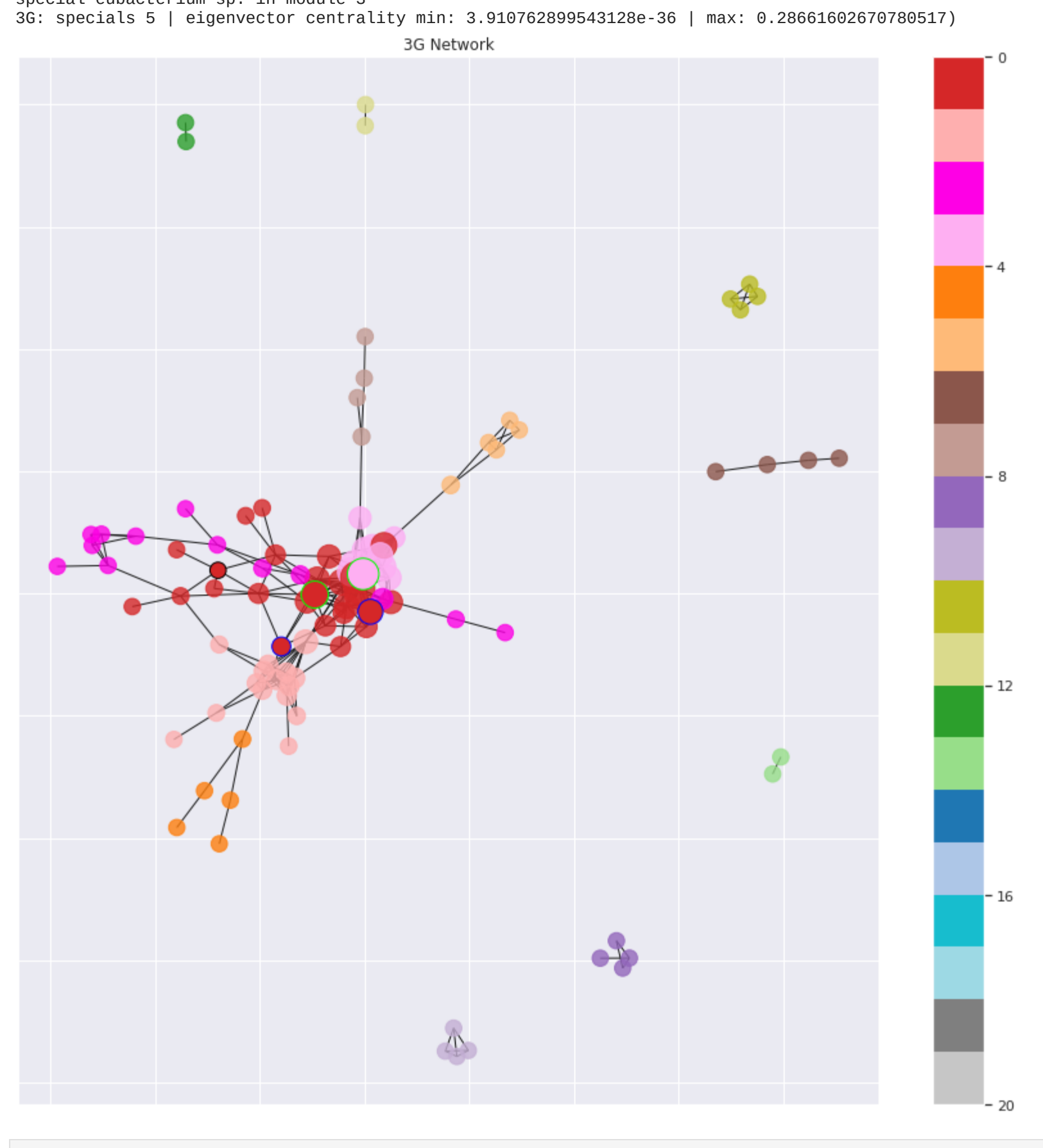
In [68]:
```
nap.visualize_network_fancy(G_2M)
```

```
special opitutus spp. in module 5
special magnetospirillum sp. in module 0
special rhodopseudomonas palustris in module 0
special pleomorphomonas oryzae in module 1
special eubacterium sp. in module 1
2M: specials 5 | eigenvector centrality min: 3.6811353520434973e-37 | max: 0.044287206467089724)
```



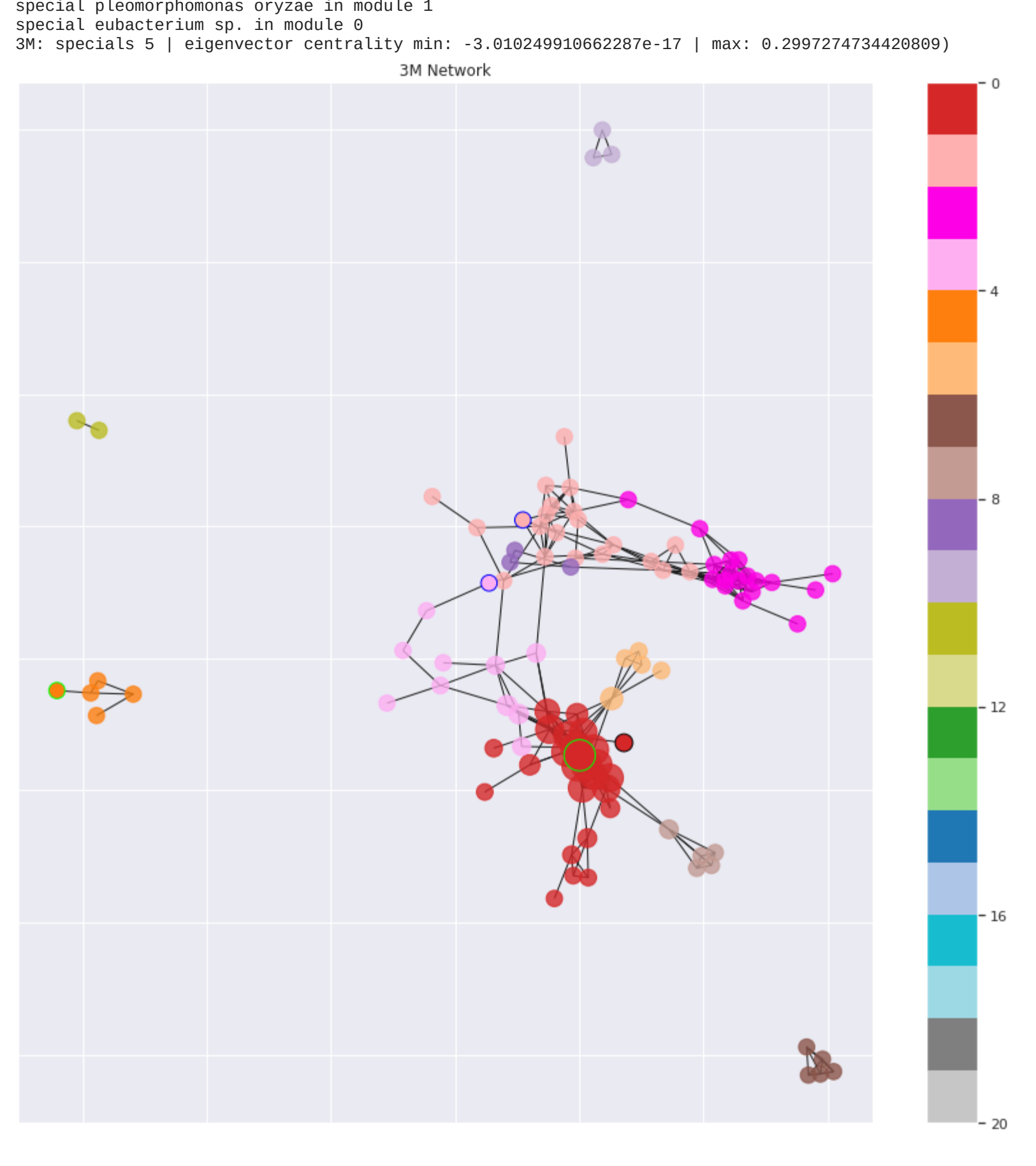
In [72]:
```
nap.visualize_network_fancy(G_3G)
```

```
special opitutus spp. in module 0
special magnetospirillum sp. in module 0
special rhodopseudomonas palustris in module 0
special pleomorphomonas oryzae in module 0
special eubacterium sp. in module 3
3G: specials 5 | eigenvector centrality min: 3.910762899543128e-36 | max: 0.28661602670780517)
```



In [82]:
```
nap.visualize_network_fancy(G_3M)
```

```
special opitutus spp. in module 0
special magnetospirillum sp. in module 4
special rhodopseudomonas palustris in module 3
special pleomorphomonas oryzae in module 1
special eubacterium sp. in module 0
3M: specials 5 | eigenvector centrality min: -3.010249910662287e-17 | max: 0.2997274734420809)
```



Eigengene Analysis

In [83]:
```
df_eigengenes_1C
```

Out[83]:

	1C3A	1C3B	1C5A	1C5B	1C7A	1C7B	1C10A	1C10B	1C20B	1C20A
0	0.232834	0.103732	0.350327	1.131969e-01	4.303742e-01	-1.420407e-01	6.246697e-01	-3.015334e-01	-1.138125e-01	3.162278e-01
1	0.142155	0.125037	0.599700	-3.256597e-01	-4.212469e-01	2.355902e-01	3.909844e-01	1.054213e-01	3.910917e-02	-3.162278e-01
2	-0.551949	0.654229	-0.335327	1.776015e-01	4.737870e-02	1.135660e-01	-7.932608e-02	4.250789e-02	1.042227e-02	-3.162278e-01
3	0.164922	0.151427	-0.044159	8.336887e-01	-2.937490e-01	-7.091393e-02	2.419097e-01	-5.330581e-02	1.430587e-02	-3.162278e-01
4	-0.105409	0.993736	-0.037118	-7.259869e-17	2.009679e-16	-7.259869e-17	-7.643385e-17	-8.026901e-17	-7.451627e-17	-6.830880e-17
5	-0.434972	0.082391	-0.226020	6.169796e-01	1.272345e-01	-9.730865e-02	-2.808844e-01	-9.762021e-02	1.830809e-01	4.739839e-01
6	-0.053199	0.169005	-0.045431	9.223619e-01	-4.972542e-02	-2.364038e-02	-2.436366e-01	-2.060801e-01	-7.094700e-02	-7.680426e-02

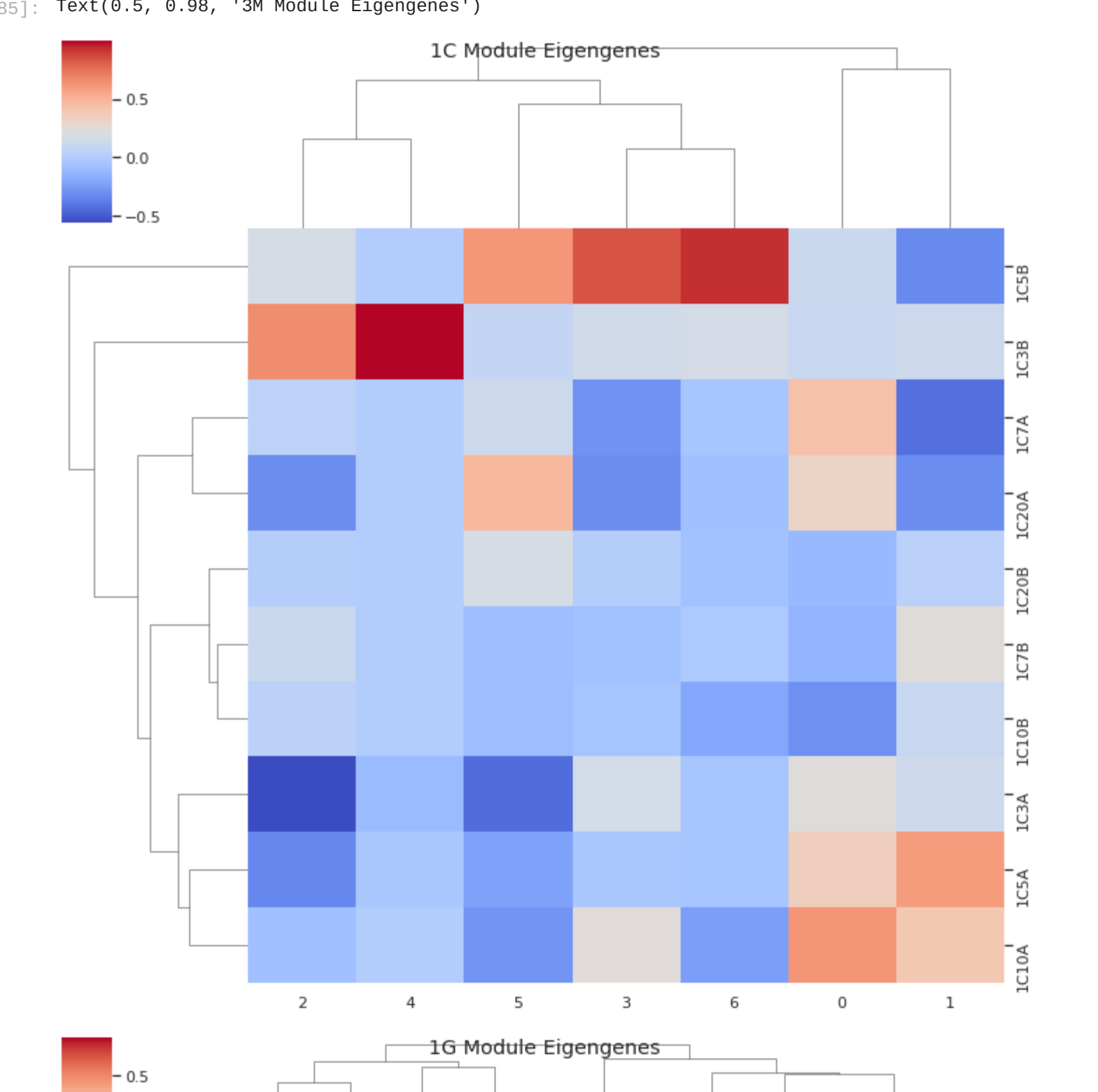
In [84]:
```
import seaborn as sns; sns.set_theme(color_codes=True)
```

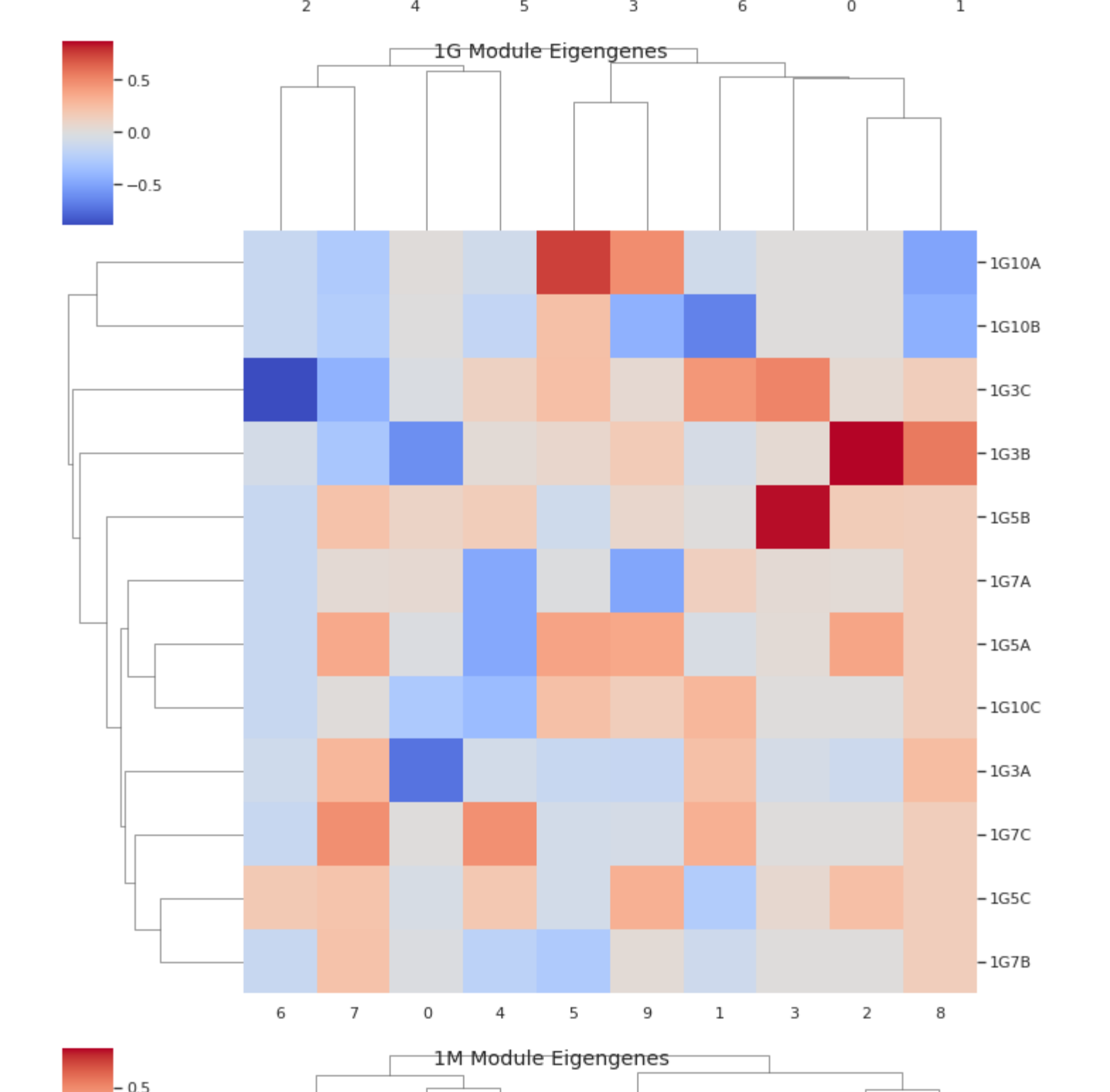
Visualize Eigengenes

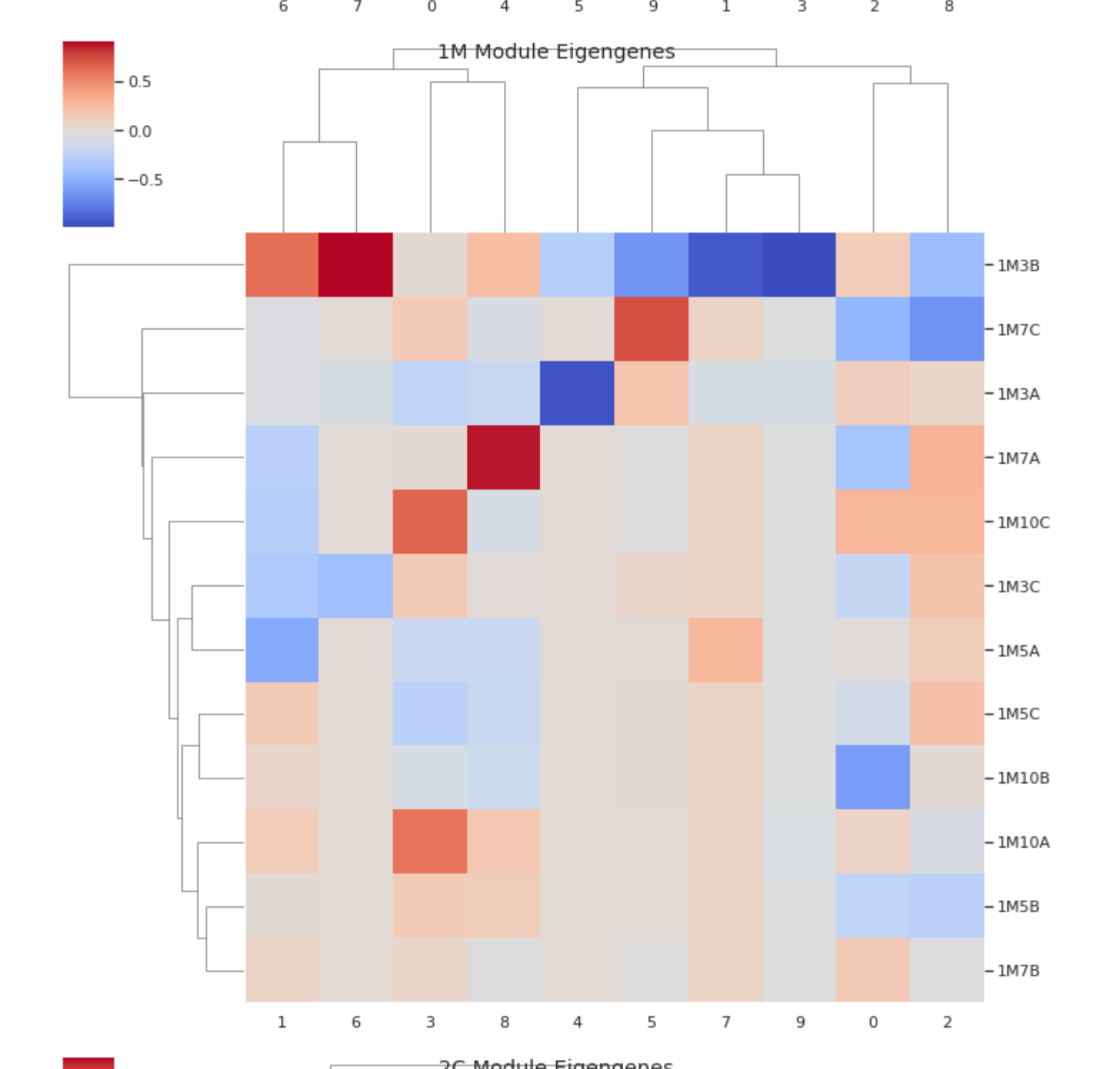
These are the main patterns of variation across samples

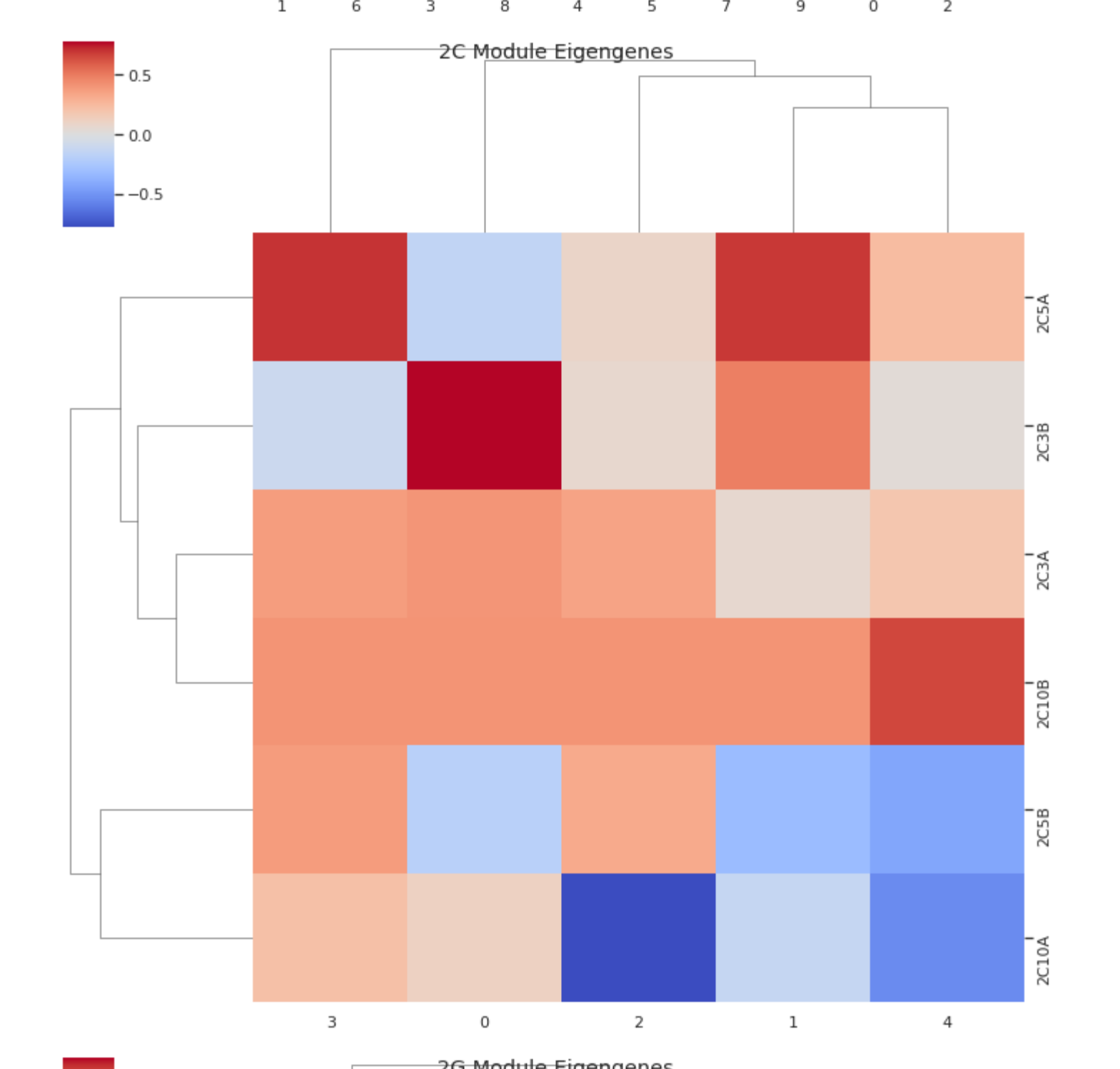
In [85]:
```
sns.clustermap(df_eigengenes_1C.T, cmap="coolwarm").fig.suptitle('1C Module Eigengenes')
sns.clustermap(df_eigengenes_1G.T, cmap="coolwarm").fig.suptitle('1G Module Eigengenes')
sns.clustermap(df_eigengenes_1M.T, cmap="coolwarm").fig.suptitle('1M Module Eigengenes')
sns.clustermap(df_eigengenes_2C.T, cmap="coolwarm").fig.suptitle('2C Module Eigengenes')
sns.clustermap(df_eigengenes_2G.T, cmap="coolwarm").fig.suptitle('2G Module Eigengenes')
sns.clustermap(df_eigengenes_2M.T, cmap="coolwarm").fig.suptitle('2M Module Eigengenes')
sns.clustermap(df_eigengenes_3G.T, cmap="coolwarm").fig.suptitle('3G Module Eigengenes')
sns.clustermap(df_eigengenes_3M.T, cmap="coolwarm").fig.suptitle('3M Module Eigengenes')
```

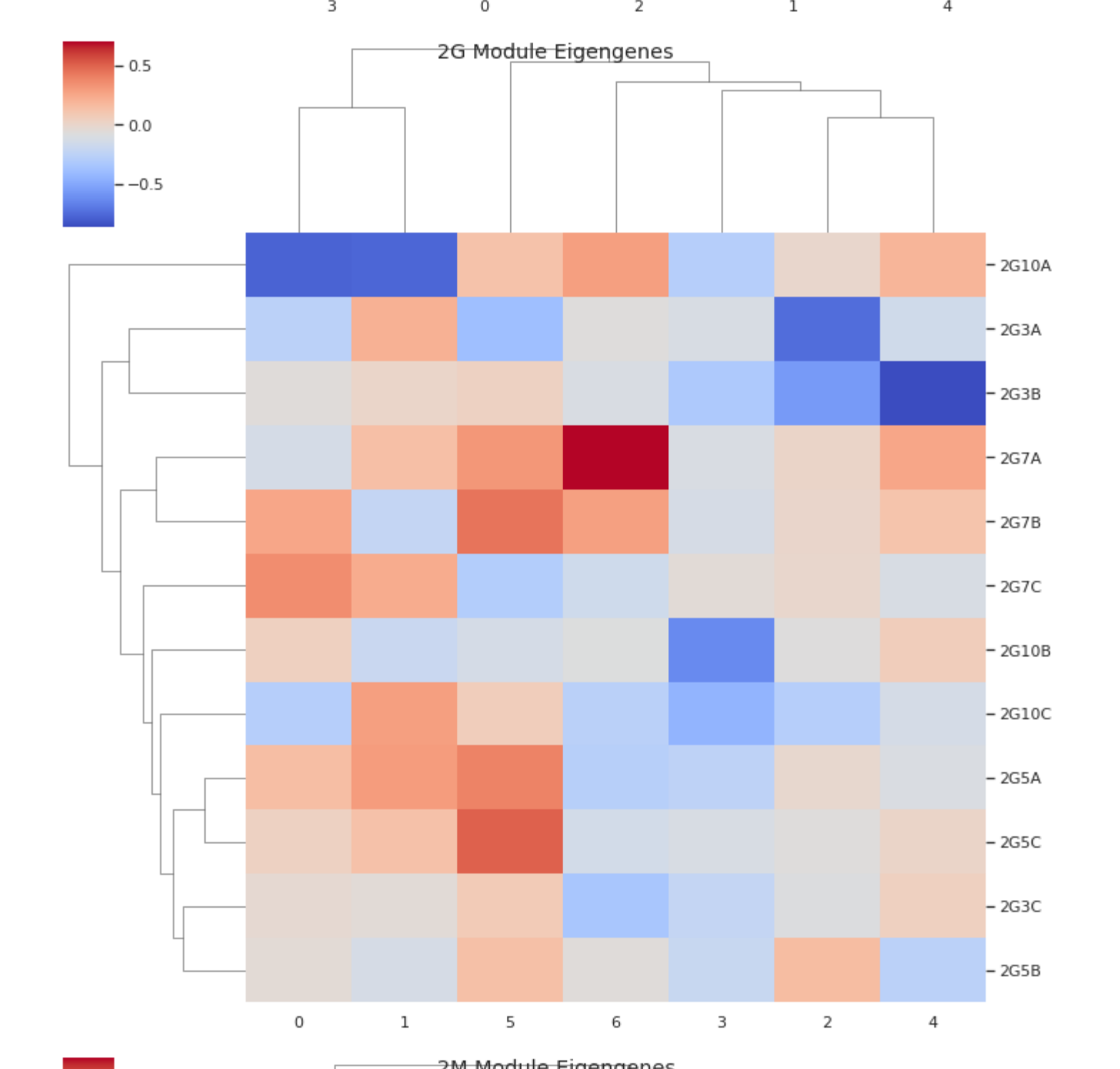
Out[85]: `Text(0.5, 0.98, '3M Module Eigengenes')`

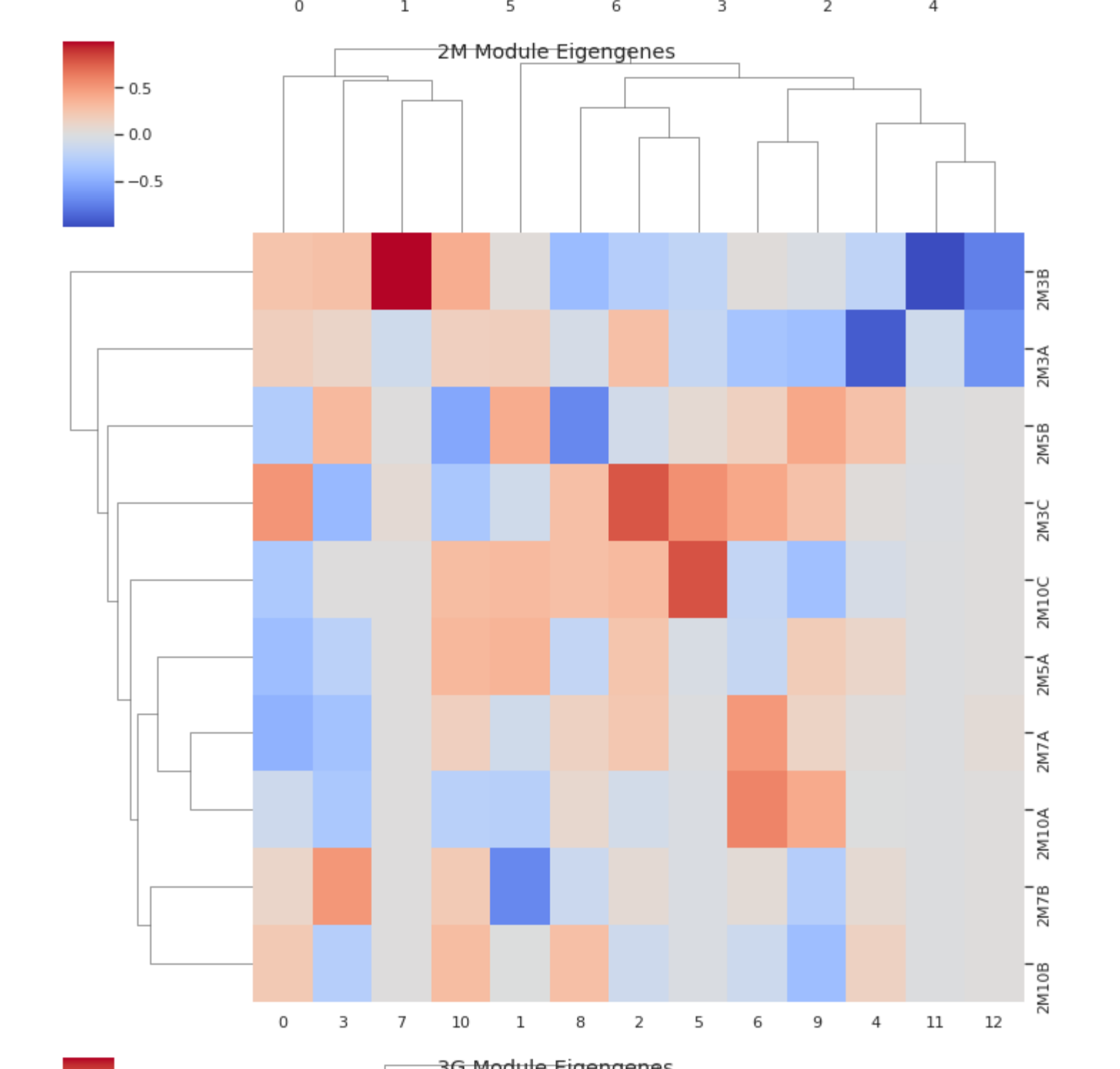


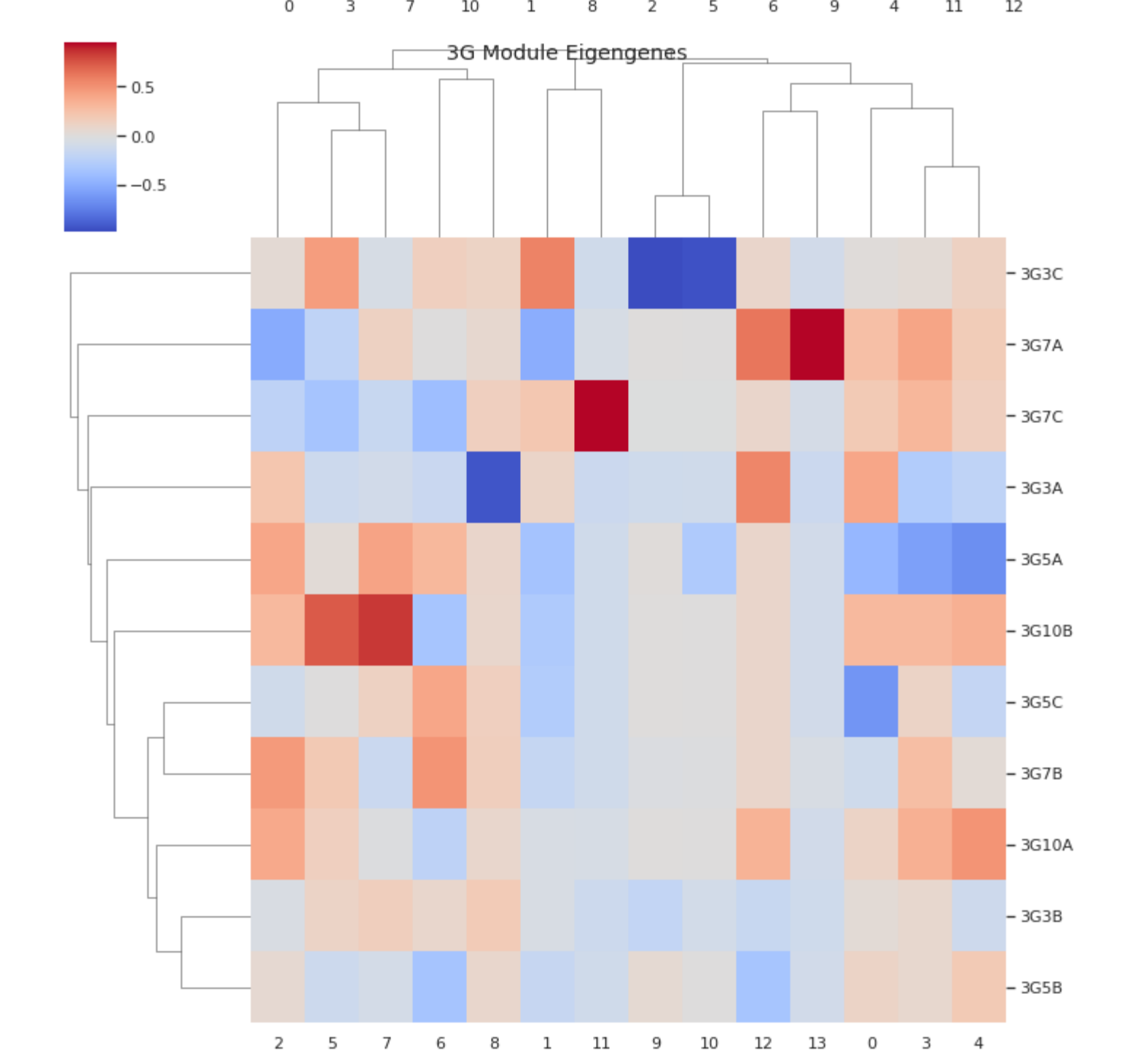


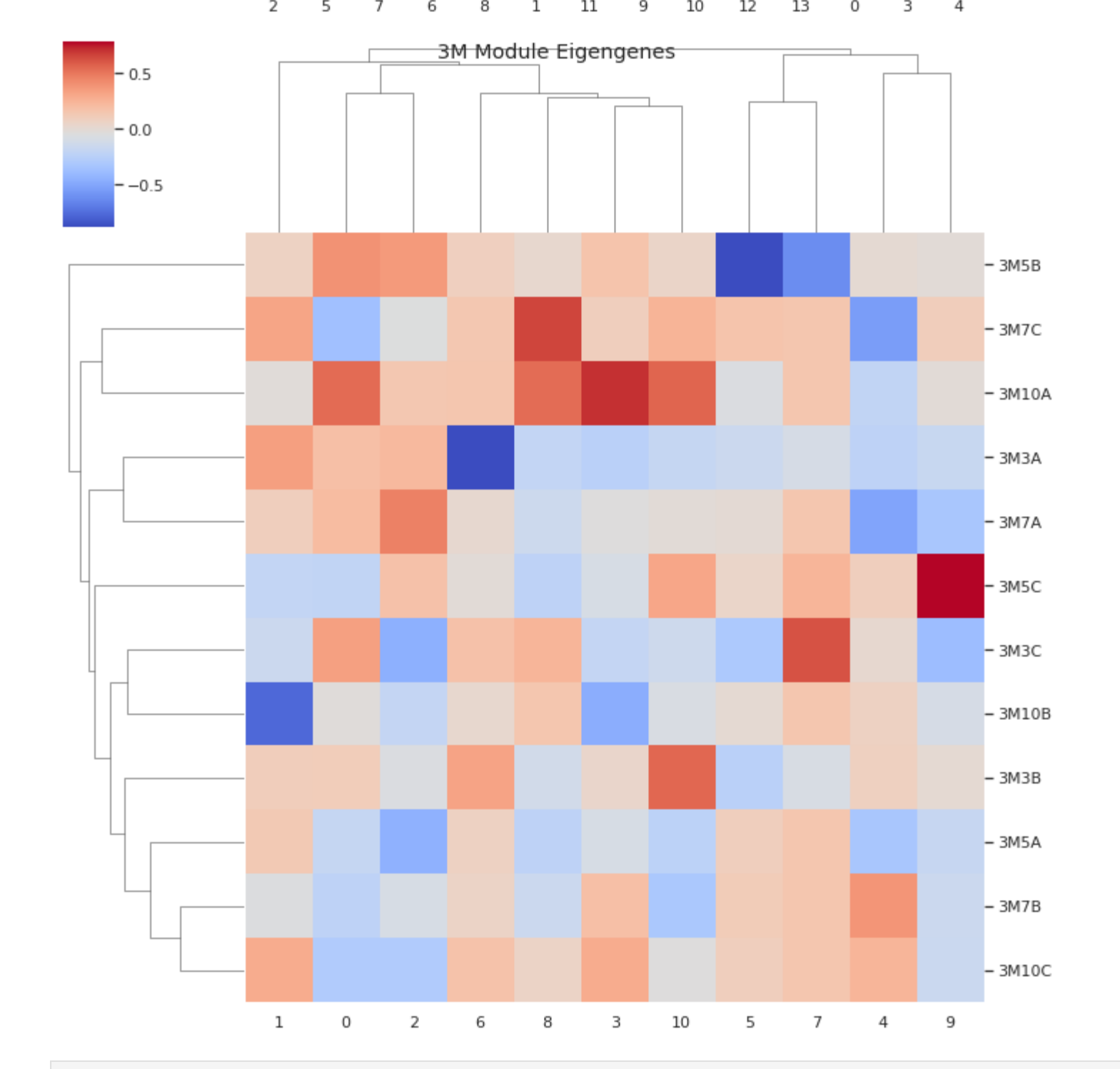












In [ ]: