David Lieberman AMATH797

P-Set #3

(1) Disconnected $\Rightarrow$ Reducible: For a disconnected graph $G=(V,E)$ with adjacency matrix W, WLOG G has 2 connected components $A,B \subsetneq V(G)$. with $|A|=m$ & $|B|=n$.

Thus, $\forall a \in A$ & $\forall b \in B$, $\{a,b\} \notin E(G) \Rightarrow W_{ij} = \begin{cases} 0, & 1 \le i \le m,\ 1 \le j \le m \\ 0, & m+1 \le i \le m+n,\ m+1 \le j \le m+n \\ 0 \text{ or } 1, & \text{otherwise} \end{cases}$

$\Rightarrow W = \begin{bmatrix} 0 & B \\ A & 0 \end{bmatrix}$ So W is block upper triangular, Therefore W is reducible. ∎

Disconnected $\Leftarrow$ Reducible: For a graph with reducible adjacency matrix W, by definition of reducible, $\exists P$ s.t. $PWP^T = T$, where T is block upper triangular.

$T = \begin{bmatrix} 0 & B' \\ A' & 0 \end{bmatrix}$ So no $a \in A'$ is adjacent to any $b \in B'$ $\Rightarrow$ no edge with an endpoint in both A' & B'.

Therefore the relabeled graph corresponding to adjacency matrix T is disconnected, with connected components A' & B'. As P is a permutation matrix, the relabeled graph defined by T is isomorphic to the original graph defined by W. ∎

② A $W = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$, $\sum_i \lambda_i = Tr(W) = 0$

$\Rightarrow$ either $\lambda_i = 0, \forall i$ **OR** $\exists \lambda_i > 0 \wedge \lambda_j < 0$, for at least one i & j

Since all elements of W are non-negative, by Perron-Frobenius Theorem $\exists \lambda_i > 0$ for some i $\therefore \exists \lambda_j < 0$ for some j. ■

B $W = \begin{bmatrix} 0 & B \\ A & 0 \end{bmatrix}$ for a bipartite graph

Take $\lambda_i > 0$ & $v_i = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ to be its corresponding eigenvector s.t.

$$W v_i = \begin{bmatrix} B v_2 \\ A v_1 \end{bmatrix} = \begin{bmatrix} \lambda_i v_1 \\ \lambda_i v_2 \end{bmatrix} = \lambda_i v_i$$

Now consider, $v_j = \begin{bmatrix} -v_1 \\ v_2 \end{bmatrix}$

$$W v_j = \begin{bmatrix} B v_2 \\ -A v_1 \end{bmatrix} = \begin{bmatrix} \lambda_i v_1 \\ -\lambda_i v_2 \end{bmatrix} = -\lambda_i \begin{bmatrix} -v_1 \\ v_2 \end{bmatrix} = -\lambda_i v_j$$

So v_j is an eigenvector corresponding to eigenvalue $-\lambda_i$ ■

(3) Let $L=D-W$ be the graph Laplacian of a graph with a single connected component, and take x to be an eigenvector corresponding to eigenvalue 0. Thus we have:

$$Lx=(D-W)x=0x=0$$

$$\Rightarrow \sum_{j=1}^{n} \deg(i)x_i - w_{ij}x_j = \sum_{j=1}^{n} w_{ij}x_i - w_{ij}x_j = 0$$

$$= \sum_{j=1}^{n} w_{ij}(x_i - x_j) = 0$$

$$x^TLx = x^T(D-W)x = x^T 0 x$$

$$\Rightarrow \sum_{i=1}^{n} x_i\left(\sum_{j=1}^{n} w_{ij}(x_i-x_j)\right) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(2x_i^2 - 2x_ix_j) = 0$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\underbrace{(x_i^2 - 2x_ix_j + x_j^2)}_{\text{Follows by Symmetry of Multiplication}} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \underbrace{w_{ij}}_{\geq 0}\underbrace{(x_i - x_j)^2}_{\geq 0} = 0$$

Fix an i. Then by connectivity we know,

$$\exists j \text{ s.t. } w_{ij} > 0 \Rightarrow x_i = x_j \ \forall i \ \therefore x \propto \vec{1}$$

So we have shown the geometric multiplicity of $\lambda_1 = 0$ to be 1. Furthermore, as L is symmetric PSD, it has a spectral decomposition. Therefore, its eigenbasis is full rank, with n linearly independent eigenvectors, which correspond to n unique eigenvalues.
So we have shown the algebraic multiplicity of $\lambda_1 = 0$ is also 1, for a graph with a single connected component.

We proceed by <u>induction on components</u>.
By Induction Hypothesis, graph G_1 has $n-1$ connected components, and the multiplicity of $\lambda=0$ of its graph Laplacian is $n-1$. Now, take $G' = G_1 \dot\cup G_2$ to be the disjoint union between G_1, and a connected graph G_2, so G' has n connected components. WLOG assume the vertices of G' are ordered according to the connected components they belong to; otherwise we may permute the vertex labels to obtain a graph isomorphic to G' whose adjacency matrix is block diagonal. This implies the graph Laplacian of G' is also block diagonal, and may be expressed as:

$$\mathbb{L}[G'] = \begin{bmatrix} \mathbb{L}[G_1] & 0 \\ 0 & \mathbb{L}[G_2] \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

Now consider a vector (above) with 1s along the dimension of of $\mathbb{L}[G_2]$ and 0s elsewhere. Embed the $n-1$ eigenvectors with $\lambda=0$ of $\mathbb{L}[G_1]$ (by IH) using 0s analogously and observe they are orthogonal to the first vector, which is the unique eigenvector with $\lambda=0$ for a single connected component by the base case. Thus we have found n linearly independent eigenvectors corresponding to $\lambda=0$ for $\mathbb{L}[G']$, so the geometric multiplicity is at least n.
Furthermore, the characteristic polynomial of $\mathbb{L}[G']$ is given by the product of the characteristic polynomials of $\mathbb{L}[G_1]$ & $\mathbb{L}[G_2]$

$$\chi(\mathbb{L}[G']) = \prod_{i=1}^{2} \chi(\mathbb{L}[G_i])$$

But we know the algebriac multiplicity of $\lambda=0$ for $\mathbb{L}[G_1]$ is $n-1$ by IH, and the algebraic multiplicity of $\lambda=0$ for $\mathbb{L}[G_2]$ is 1 by the base case, so the <u>algebraic multiplicity</u> of $\lambda=0$ for $\mathbb{L}[G']$ is n. Additionally, the algebraic multiplicity upper bounds the geometric multiplicity, thus the <u>geometric multiplicity</u> of $\lambda=0$ for $\mathbb{L}[G']$ is also n. ∎

5 $\mathbb{P}\{X_{t+1} \in S^c \mid X_t \in S\} + \mathbb{P}\{X_{t+1} \in S \mid X_t \in S^c\}$

$$= \frac{\sum_{i \in S} \sum_{j \in S^c} \pi_i \mathbb{P}_{ij}}{\frac{Vol(S)}{Vol(G)}} + \frac{\sum_{i \in S^c} \sum_{j \in S} \pi_i \mathbb{P}_{ij}}{\frac{Vol(S^c)}{Vol(G)}}$$

$$= \frac{\sum_{i \in S} \sum_{j \in S^c} \left(\frac{\cancel{deg(i)}}{\cancel{Vol(G)}}\right)\left(\frac{W_{ij}}{\cancel{deg(i)}}\right)}{\frac{Vol(S)}{\cancel{Vol(G)}}} + \frac{\sum_{i \in S^c} \sum_{j \in S^c} \left(\frac{\cancel{deg(i)}}{\cancel{Vol(G)}}\right)\left(\frac{W_{ij}}{\cancel{deg(i)}}\right)}{\frac{Vol(S^c)}{\cancel{Vol(G)}}}$$

$$= \frac{\sum_{i \in S} \sum_{j \in S^c} W_{ij}}{Vol(S)} + \frac{\sum_{i \in S^c} \sum_{j \in S} W_{ij}}{Vol(S^c)}$$

$$= \frac{Cut(S)}{Vol(S)} + \frac{Cut(S^c)}{Vol(S^c)} = Ncut(S)$$

⑥ In ③ we showed $x^T L x = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(x_i - x_j)^2$

$$y^T L y = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\underbrace{y_i - y_j}_{})^2$$

If $\begin{cases} i,j \in S \\ i,j \in S^c \end{cases} \Rightarrow y_i = y_j \Rightarrow 0$, so WLOG $i \in S$ & $j \in S^c$

$$\therefore = \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\left(\left(\frac{Vol(S^c)}{Vol(S)\,Vol(G)}\right)^{1/2} + \left(\frac{Vol(S)}{Vol(S^c)\,Vol(G)}\right)^{1/2}\right)^2\right)$$

$$= \frac{1}{2}\left(\frac{Vol(S^c)}{Vol(S)\,Vol(G)} + \frac{Vol(S)}{Vol(S^c)\,Vol(G)} + 2\left(\frac{\cancel{Vol(S^c)}}{\cancel{Vol(S)}\,Vol(G)} \cdot \frac{\cancel{Vol(S)}}{\cancel{Vol(S^c)}\,Vol(G)}\right)^{1/2}\right)\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\right)$$

$$= \frac{1}{2}\left(\frac{1}{Vol(S)} + \frac{1}{Vol(S^c)}\right)\left(\sum_{i\in S}\sum_{j\in S^c} w_{ij} + \sum_{i\in S^c}\sum_{j\in S} w_{ij}\right)$$

$$= \frac{1}{2}\left(\frac{1}{Vol(S)} + \frac{1}{Vol(S^c)}\right)\left(Cut(S) + Cut(S^c)\right)$$

$$= \frac{1}{2}\left(\frac{2\,Cut(S)}{Vol(S)} + \frac{2\,Cut(S^c)}{Vol(S^c)}\right)$$

$$= \frac{Cut(S)}{Vol(S)} + \frac{Cut(S^c)}{Vol(S^c)} = Ncut(S)$$

AMATH797 PSet3 David Lieberman

In [1]:

```
import os
import numpy as np
import sklearn
from sklearn.cluster import KMeans
from sklearn.neighbors import kneighbors_graph
from sklearn.cluster import SpectralClustering
np.random.seed(0)

import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import networkx as nx
from networkx.generators import stochastic_block_model
options = {
    'node_color': '#000000',
    'node_size': 125,
    #'edge_color': '#696969',
    'edge_cmap': plt.cm.Spectral,
    'edge_size': 2.5,
    'alpha': 0.75,
    }

import scipy
from scipy.io import loadmat
from scipy.sparse import csgraph
from sklearn.feature_extraction.image import extract_patches_2d
from PIL import Image

import torch
from torch import tensor
from itertools import combinations

# from google.colab import drive
# drive.mount('/content/drive')
# cd 'drive/My Drive/2020S_AMATH797/PSet3'
os.chdir(os.path.expanduser(os.sep.join(["/mnt","c","Users","darkg","Desktop", "Homework Scans", "2020S_AMATH797"
, "PSet3"])))
# os.chdir(os.path.expanduser(os.sep.join(["~","Desktop", "Homework Scans", "2020S_AMATH797", "PSet3"])))
```

In [2]:

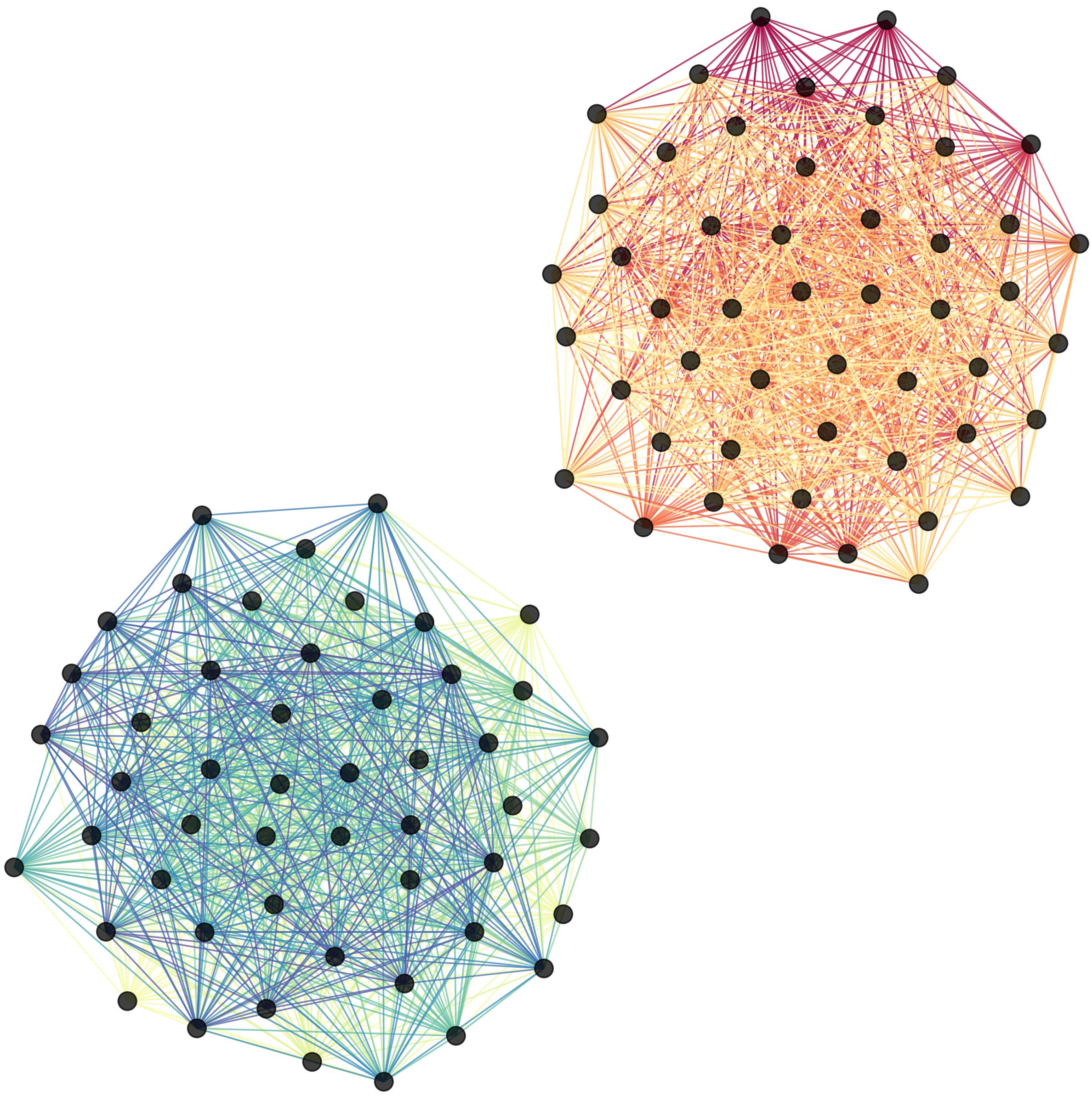
```
def draw_stochastic_block(n, p, q, draw_algo):
  sizes = [n // 2, n - (n // 2)]
  probs = [[p, q], [q, p]]
  G = stochastic_block_model(sizes, probs, seed = 0)
  pos = nx.nx_agraph.graphviz_layout(G, prog = draw_algo)
  #pos = nx.nx_pydot.pydot_layout(G, prog = draw_algo)
  plt.figure(figsize=(10, 10))
  nx.draw(G, pos, **options, edge_color = np.arange(G.number_of_edges()))
```

In [3]:

```
def eigen_laplacian(n, p, q):
  sizes = [n // 2, n // 2]
  probs = [[p, q], [q, p]]
  G = stochastic_block_model(sizes, probs, seed=0)

  # Graph Laplacian
  L = nx.laplacian_matrix(G)
  vals, vecs = np.linalg.eig(L.A)
  vals_sorted = vals[np.argsort(vals)]
  vecs_sorted = vecs[:, np.argsort(vals)]

  # Expected Laplacian
  EL = (p+q)*(n//2)*np.identity(n) - (p+q)/2*np.ones((n,n)) - (p-q)/2*np.outer(np.repeat([1,-1], n//2), np.repeat
([1,-1], n//2))
  Evals, Evecs = np.linalg.eig(EL)
  Evals_sorted = Evals[np.argsort(Evals)]
  Evecs_sorted = Evecs[:, np.argsort(Evals)]

  fig, ax = plt.subplots(1, 3, figsize = (12, 4), sharey=True)
  j = 1
  for i in np.arange(3):
      ax[i].scatter(x = np.arange(len(vecs_sorted[:, i])), y = vecs_sorted[:, i], marker = "o", color = plt.cm.Pa
ired(j), alpha = 0.75, label = str(round(vals_sorted[i].real, 3)))
      ax[i].scatter(x = np.arange(len(Evecs_sorted[:, i])), y = Evecs_sorted[:, i], marker = "X", color = plt.cm.
Paired(j - 1), label = str(round(Evals_sorted[i].real, 3)))
      ax[i].legend(title = "Eigenvalue", loc = "lower right")
      ax[i].set(title = "Eigenvector #" + str(i + 1), xlabel = "", ylabel = "")
      j += 2
  plt.subplots_adjust(wspace = 0.1)
  fig.add_subplot(111, frameon = False)
  plt.tick_params(labelcolor='none', top = False, bottom = False, left = False, right = False)
  plt.xlabel("idx")
  legend_elements = [Line2D([], [], color = "black", marker = "o", linestyle = "", label = "Actual"), Line2D([],
[], color = "grey", marker = "X", linestyle = "", label = "Expected")]
  fig.legend(handles = legend_elements, title = "Laplacian", loc = "center right", borderaxespad = 0.15)
```

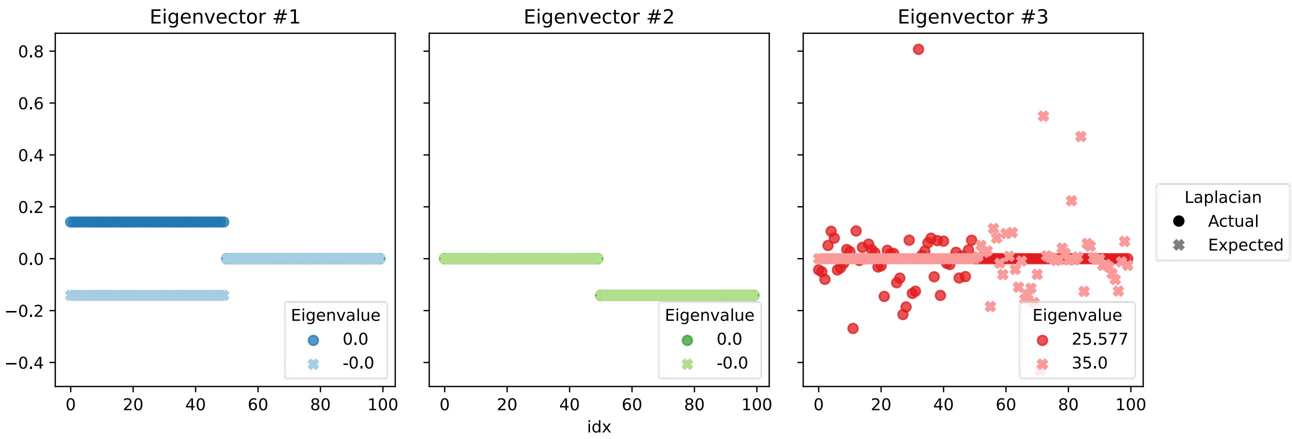
In [4]:

```
draw_stochastic_block(n = 100, p = 0.7, q = 0.0, draw_algo = "fdp")
```



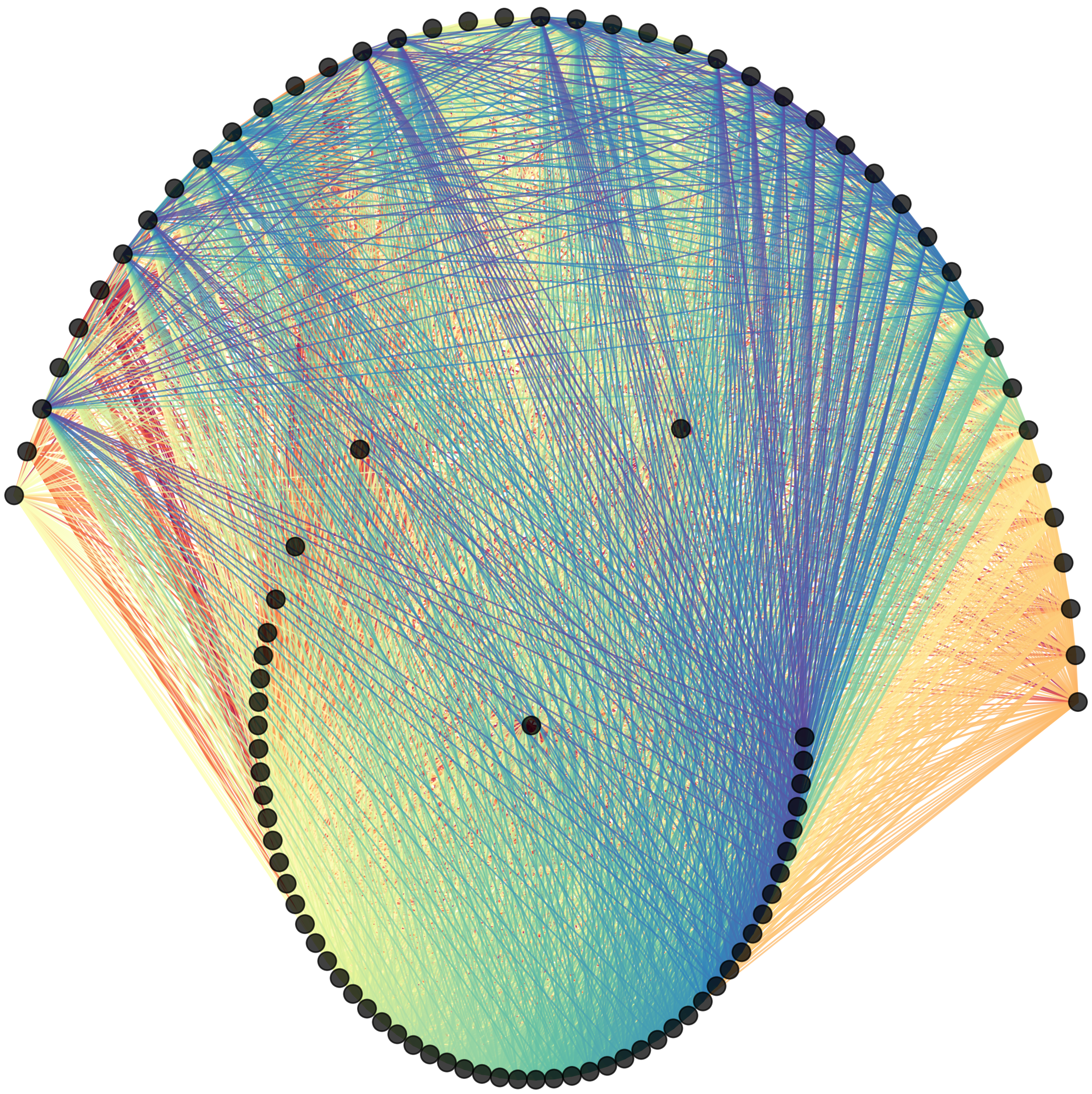
In [5]:

```
eigen_laplacian(n = 100, p = 0.7, q = 0.0)
```



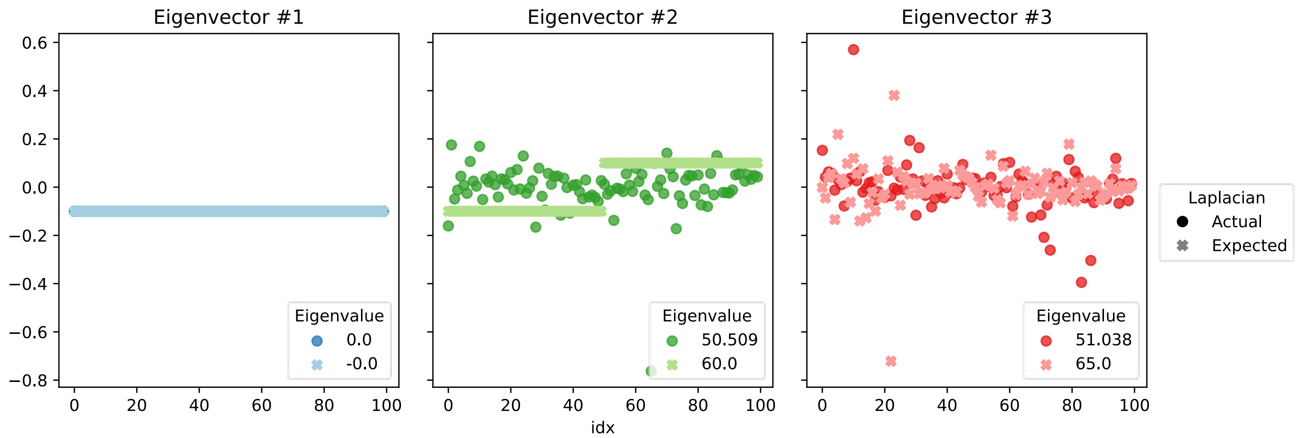
In [6]:

```
draw_stochastic_block(n = 100, p = 0.7, q = 0.6, draw_algo = "twopi")
```



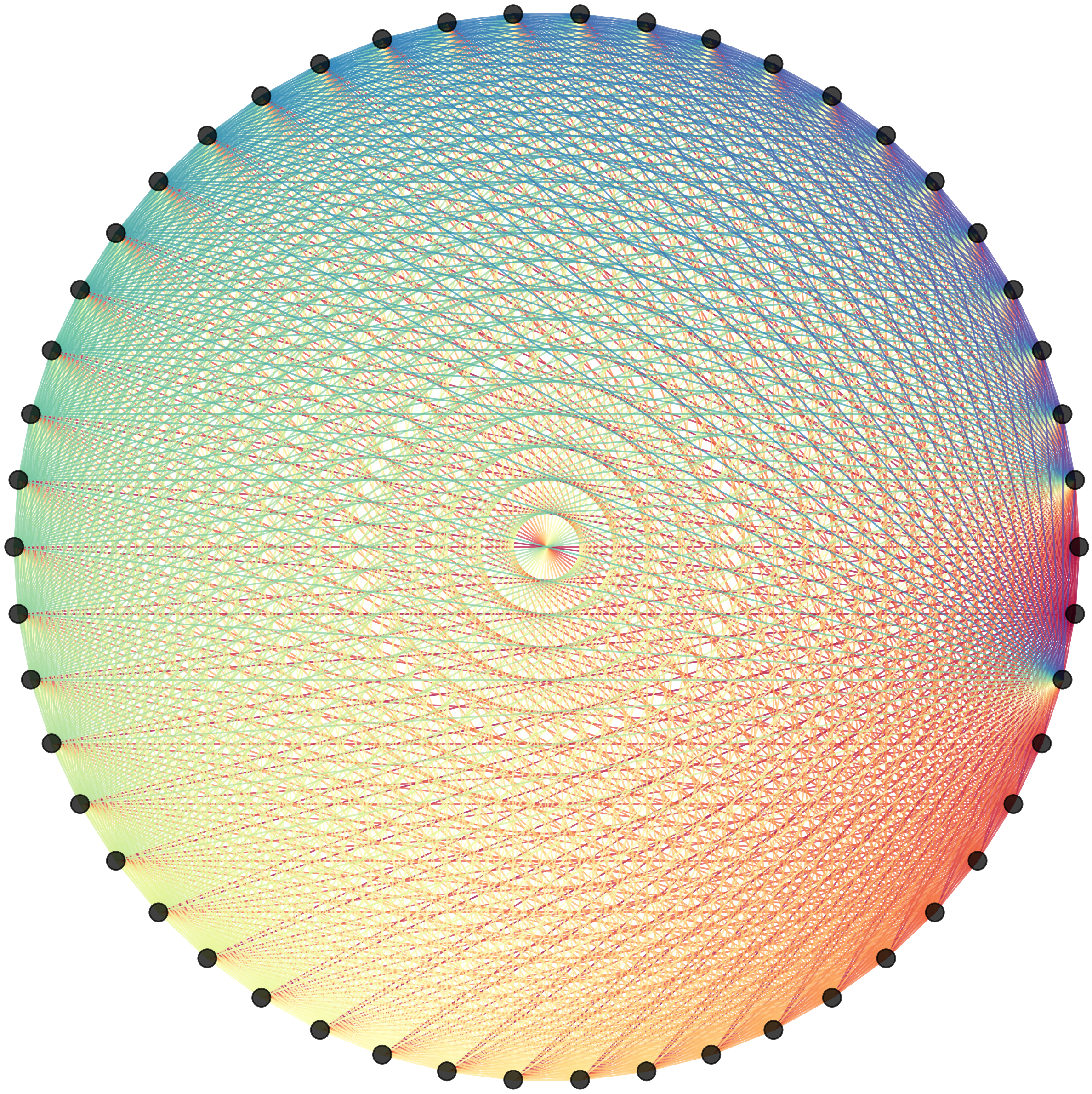
In [7]:

```
eigen_laplacian(n = 100, p = 0.7, q = 0.6)
```



In [8]:

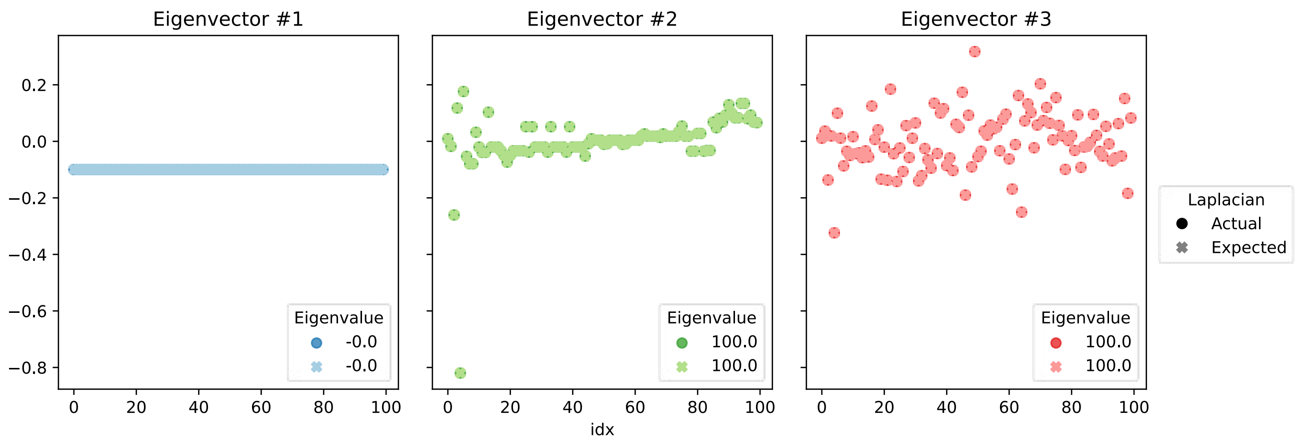
```
draw_stochastic_block(n = 50, p = 1.0, q = 1.0, draw_algo = "circo")
```



Complete Graph (Note only 50 nodes are visualized for computational purposes)

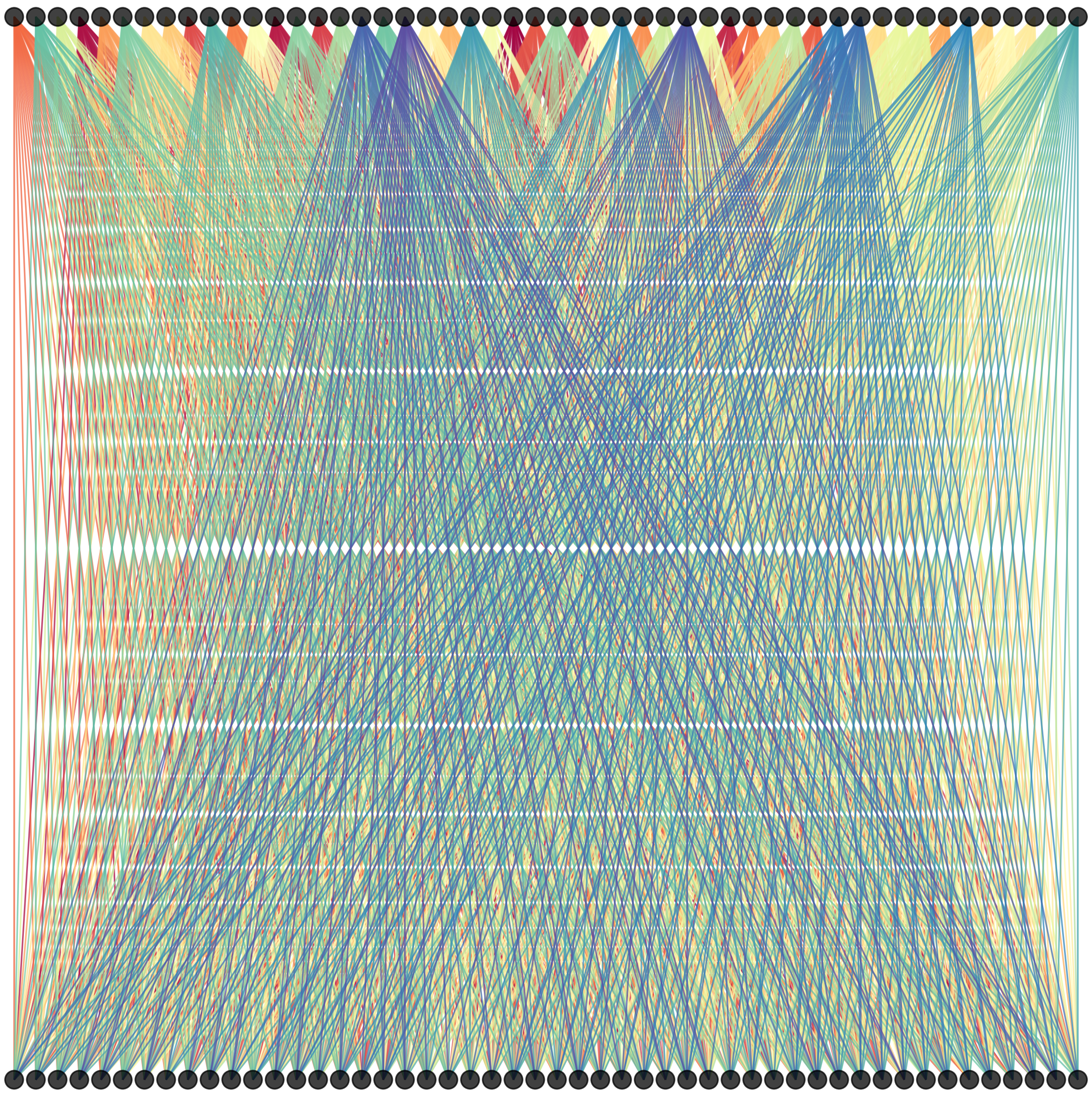
In [9]:

```
eigen_laplacian(n = 100, p = 1.0, q = 1.0)
```



In [10]:

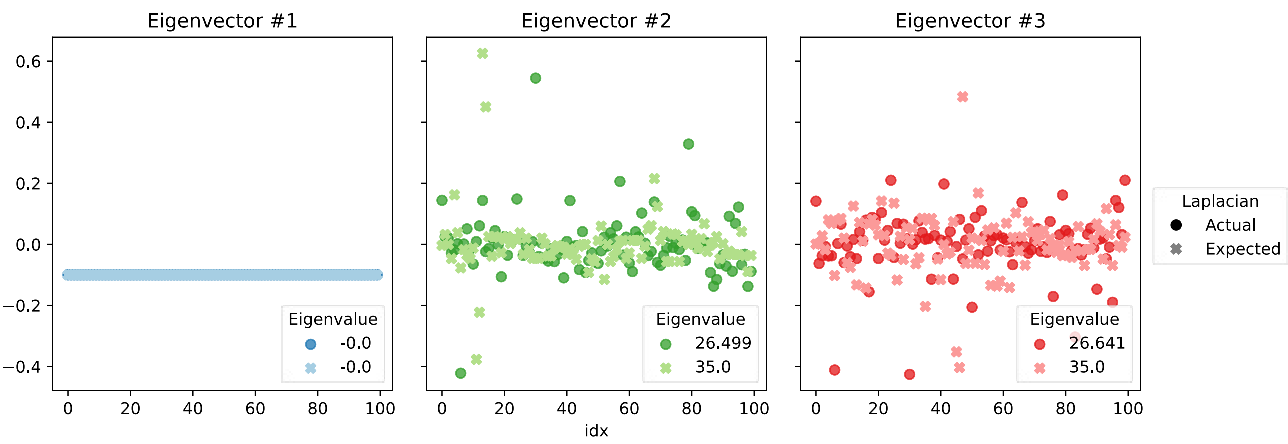
```
draw_stochastic_block(n = 100, p = 0.0, q = 0.7, draw_algo = "dot")
```



(Nearly) Complete Bipartite Graph

In [11]:

```
eigen_laplacian(n = 100, p = 0.0, q = 0.7)
```



In [12]:

```
data6 = scipy.io.loadmat('Data6.mat')['XX'].flatten()
smile = data6[2]
ring = data6[5]
```

In [13]:

```
def spectral_clustering(data, num_clusters, affinity, num_neighbors=5, gamma=1):

  if affinity == "knn":
    Affinity = kneighbors_graph(data, n_neighbors = num_neighbors).toarray()
  if affinity == "rbf":
    Affinity = np.exp(-gamma * np.linalg.norm(data[..., np.newaxis] - data.transpose(), axis = 1)**2)

  L_sym = nx.normalized_laplacian_matrix(nx.from_numpy_array(Affinity))

  vals, vecs = np.linalg.eig(L_sym.A)
  sorted_idx = np.argsort(vals)
  vals = vals[sorted_idx].real
  vecs = vecs[:,sorted_idx].real

  kmeans = KMeans(n_clusters = num_clusters).fit(vecs[:,1:num_clusters])
  return kmeans.labels_
```

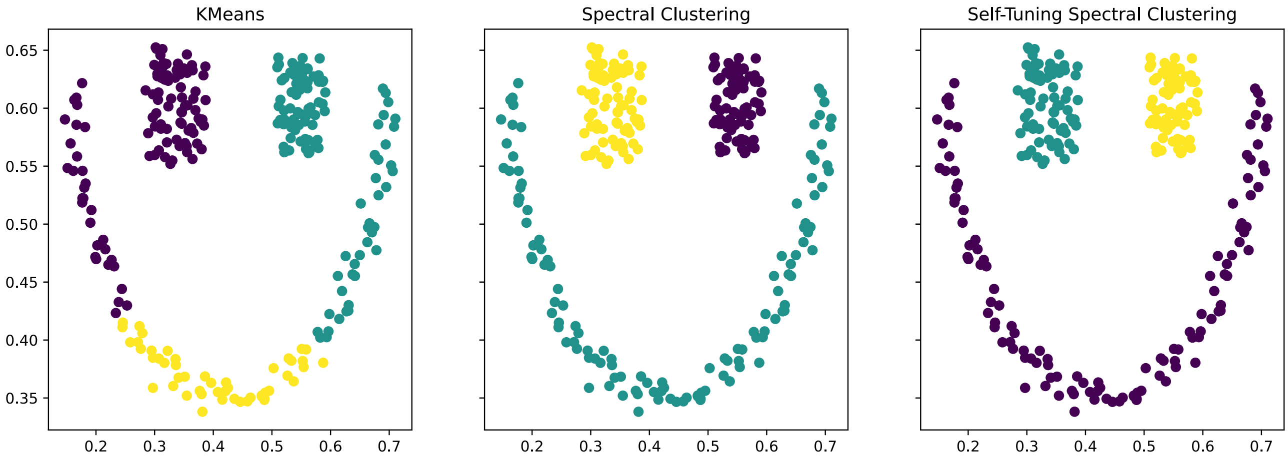
In [14]:

```
def self_tuning_spectral_clustering(data, num_clusters, neighbors_sigma):

  sigma = np.array([])
  for i in np.arange(len(data)):
    temp = [np.linalg.norm(data[i,:] - point[0]) for point in zip(data)]
    temp.sort()
    sigma = np.append(sigma, temp[neighbors_sigma])

  pairwise_distance = np.linalg.norm(data[..., np.newaxis] - data.transpose(), axis =1)
  pairwise_variance = sigma[..., np.newaxis] * sigma.transpose()
  Affinity_hat = np.exp(-pairwise_distance**2 / pairwise_variance)
  np.fill_diagonal(Affinity_hat, 0)
  L = csgraph.laplacian(Affinity_hat, normed=True)

  vals, vecs = np.linalg.eig(L)
  sorted_idx = np.argsort(vals)
  vals = vals[sorted_idx].real
  vecs = vecs[:,sorted_idx].real

  kmeans = KMeans(n_clusters = num_clusters).fit(vecs[:,1:num_clusters])
  return kmeans.labels_
```

In [15]:

```
KMeans_clusters = KMeans(n_clusters = 3, random_state = 0).fit_predict(smile)
spectral_clusters = spectral_clustering(smile, num_clusters = 3, affinity = "knn", num_neighbors = 5)
self_tuning_spectral_clusters = self_tuning_spectral_clustering(smile, num_clusters = 3, neighbors_sigma = 7)

fig, ax = plt.subplots(1, 3, figsize = (15, 5), sharey=True)
ax[0].set_title("KMeans")
ax[0].scatter(smile[:, 0], smile[:, 1], c = KMeans_clusters)

ax[1].set_title("Spectral Clustering")
ax[1].scatter(smile[:, 0], smile[:, 1], c = spectral_clusters)

ax[2].set_title("Self-Tuning Spectral Clustering")
ax[2].scatter(smile[:, 0], smile[:, 1], c = self_tuning_spectral_clusters)
```

Out[15]:



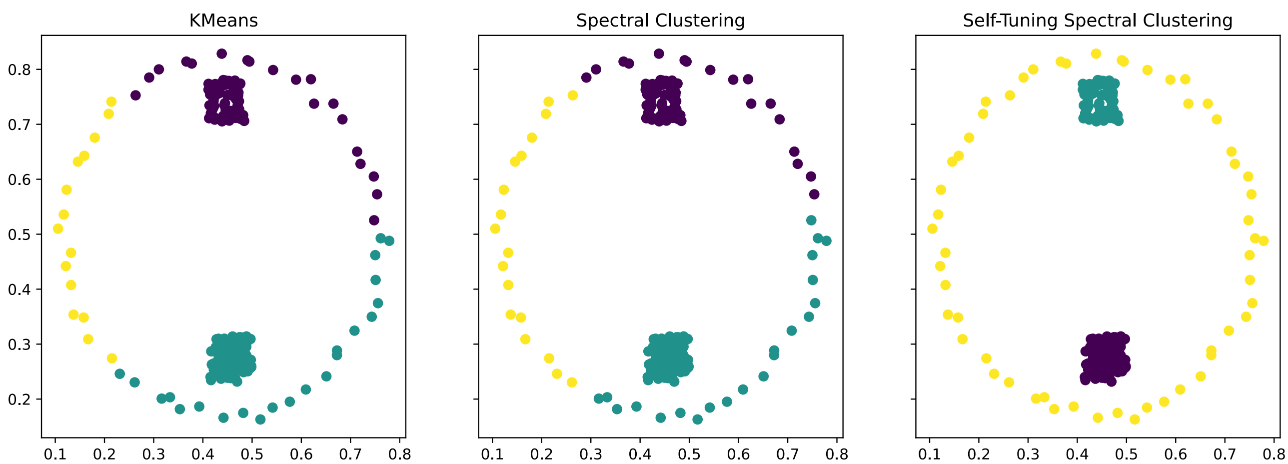
Spectral Clustering: Using k-nearest neighbors with k=5

Self-Tuning Spectral Clustering: σ_neighbors = 7

In [16]:

```
KMeans_clusters = KMeans(n_clusters = 3, random_state = 0).fit_predict(ring)
spectral_clusters = spectral_clustering(ring, num_clusters = 3, affinity = "rbf", gamma = 0.3)
self_tuning_spectral_clusters = self_tuning_spectral_clustering(ring, num_clusters = 3, neighbors_sigma = 7)

fig, ax = plt.subplots(1, 3, figsize = (15, 5), sharey=True)
ax[0].set_title("KMeans")
ax[0].scatter(ring[:, 0], ring[:, 1], c = KMeans_clusters)

ax[1].set_title("Spectral Clustering")
ax[1].scatter(ring[:, 0], ring[:, 1], c = spectral_clusters)

ax[2].set_title("Self-Tuning Spectral Clustering")
ax[2].scatter(ring[:, 0], ring[:, 1], c = self_tuning_spectral_clusters)
```

Out[16]:



Spectral Clustering: Using RBF kernel with γ=0.3

Self-Tuning Spectral Clustering: σ_neighbors = 7

KMeans performs quite poorly across both datasets. This is because one of the core assumptions of KMeans is that the clusters are convex and disjoint, while the clusters in the datasets clearly violate this assumption.

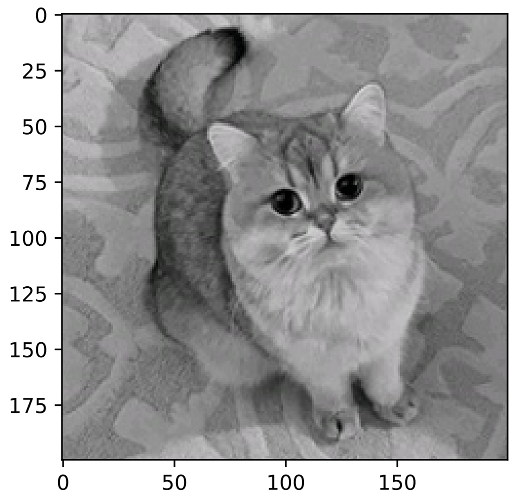
Spectral clustering works well for the smile dataset where the distance between each cluster is relatively equal. However in the ring dataset, we see that the while the inner two squares are far away from each other, they are both very close to the third cluster, circumscibed ellipse. Thus the RBF kernel's global scaling does a poor job differentiating between the inner square and the proximal arc of the ellipse. This is solved by the scaling parameter in the Self-Tuning Spectral Clustering algorithm which adaptively scales the bandwidth of the kernel to account for the local statistics of the neighborhoods surrounding points.

In [17]:

```
cat = np.asarray(Image.open("cat_smol.png").convert("L"))
plt.imshow(cat, cmap='gray', vmin=0, vmax=255)
```

Out[17]:

```
<matplotlib.image.AxesImage at 0x7f8292050668>
```

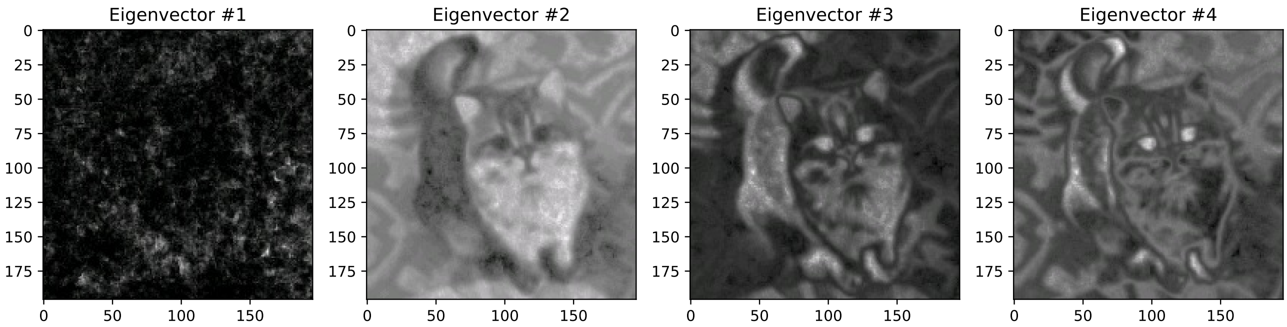


In [18]:

```
patches = extract_patches_2d(cat, (5, 5))
patches = patches.reshape((38416, 25))

patches_graph = nx.from_scipy_sparse_matrix(kneighbors_graph(patches, n_neighbors = 5))
L_sym = nx.normalized_laplacian_matrix(patches_graph)

vals, vecs = scipy.sparse.linalg.eigs(L_sym, k=100, which='SR')
vecs = vecs.real
```

In [19]:

```
fig, ax = plt.subplots(1, 4, figsize = (15, 25))
for i in np.arange(4):
  ax[i].set_title("Eigenvector #" + str(i + 1))
  ax[i].imshow(np.reshape(vecs[:, i], ((196, 196))), cmap = 'gray')
```



In [20]:

```
kmeans = KMeans(n_clusters = 7, random_state = 132).fit(vecs.real) #132, 133, 172
clusters = kmeans.labels_

plt.imshow(np.reshape(clusters, ((196, 196))), cmap='gray', alpha = 0.75)
```

Out[20]:

```
<matplotlib.image.AxesImage at 0x7f81f64842e8>
```