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"])))
```

Problem 4

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()))
```

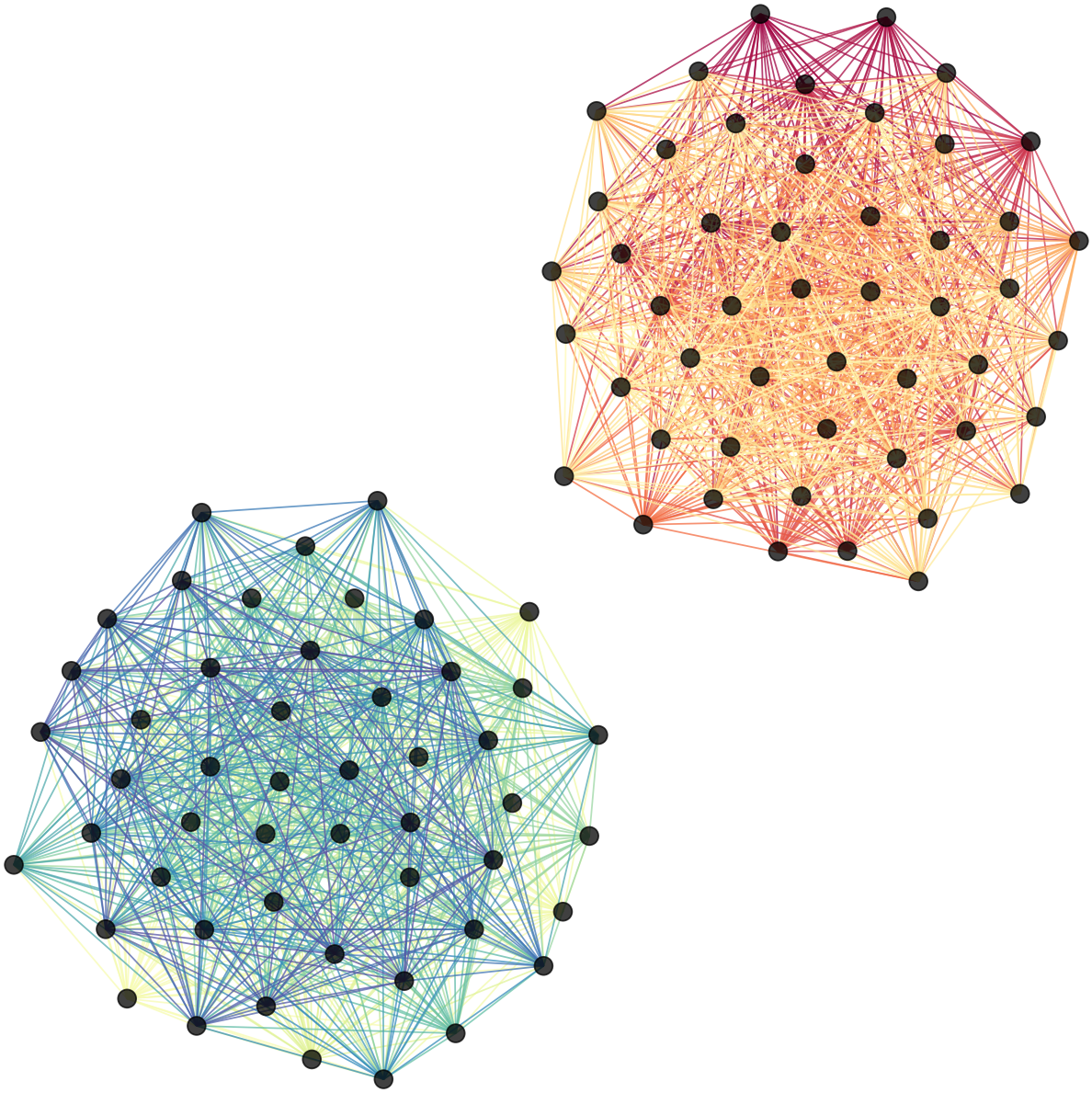
In [3]:

```
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)
```

Part A

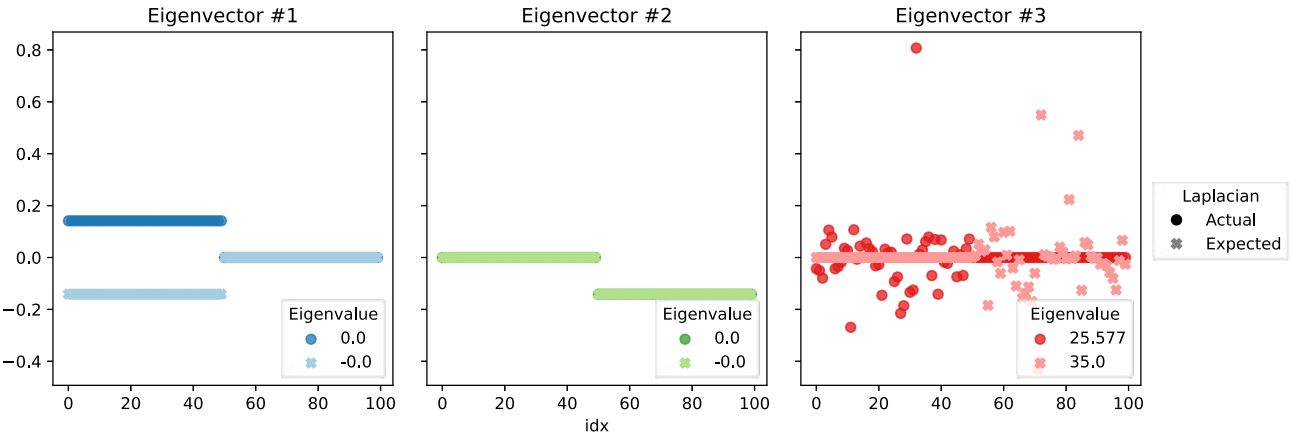
In [4]:

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



In [5]:

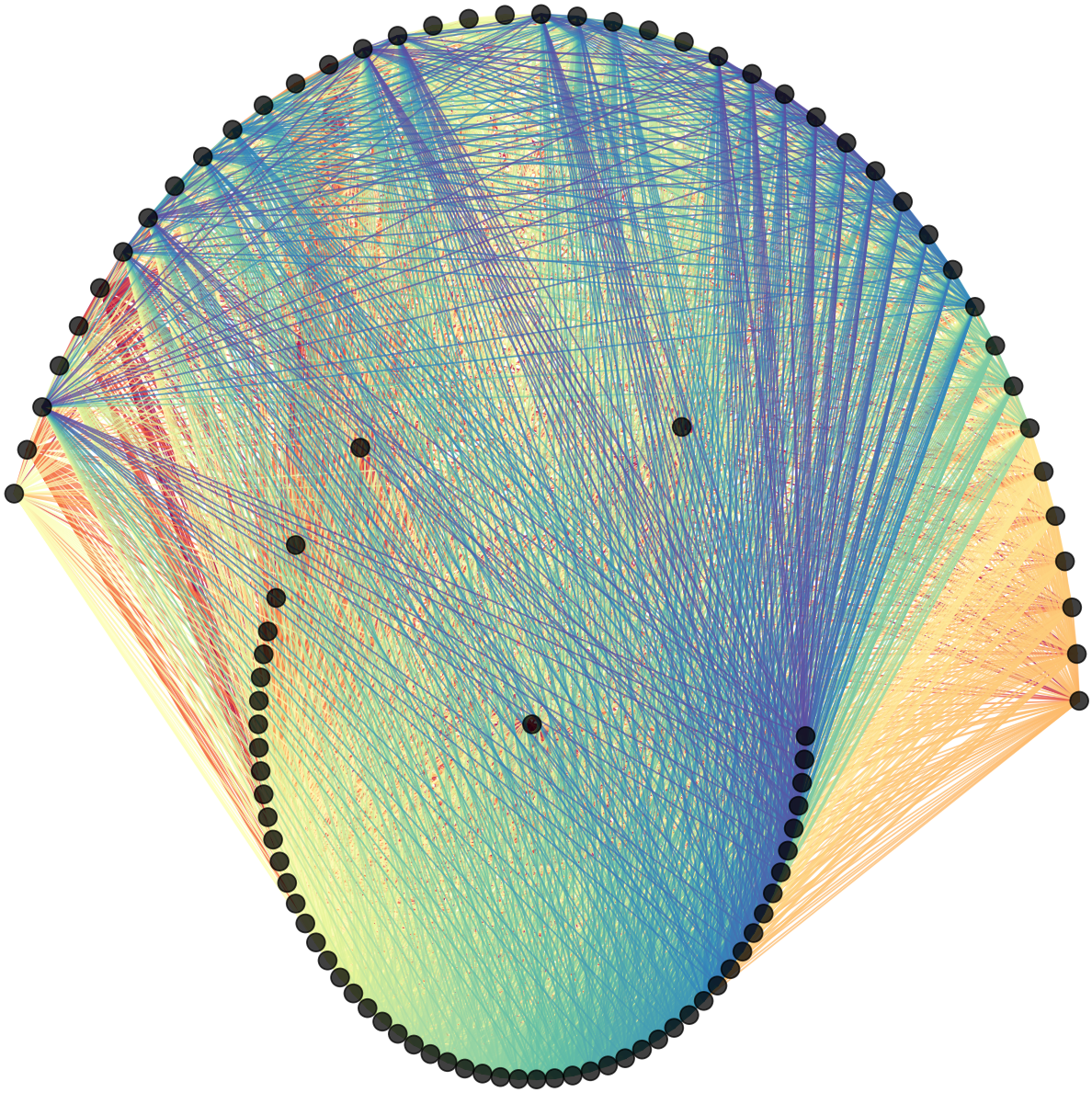
```
eigen_laplacian(n = 100, p = 0.7, q = 0.0)
```



Part B

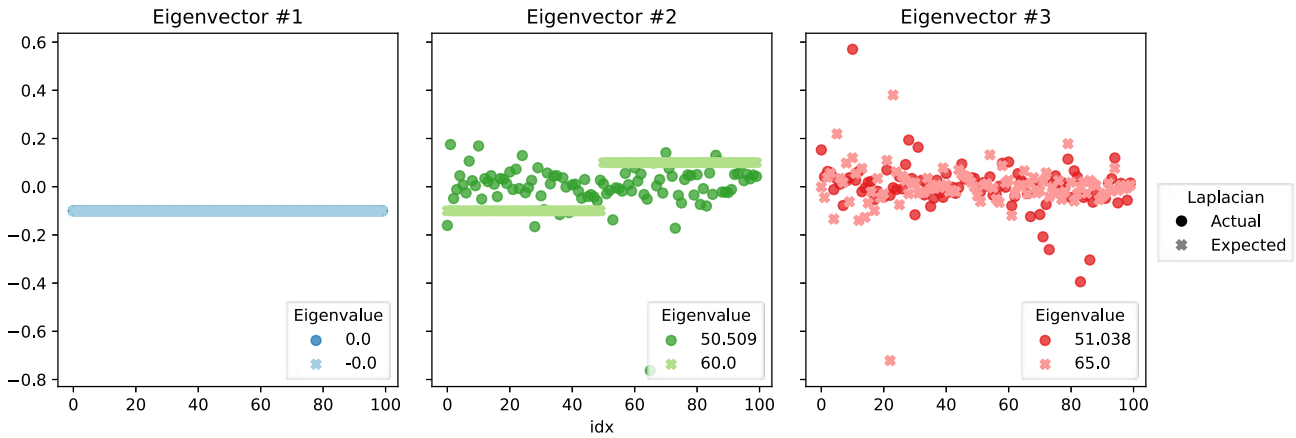
In [6]:

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



In [7]:

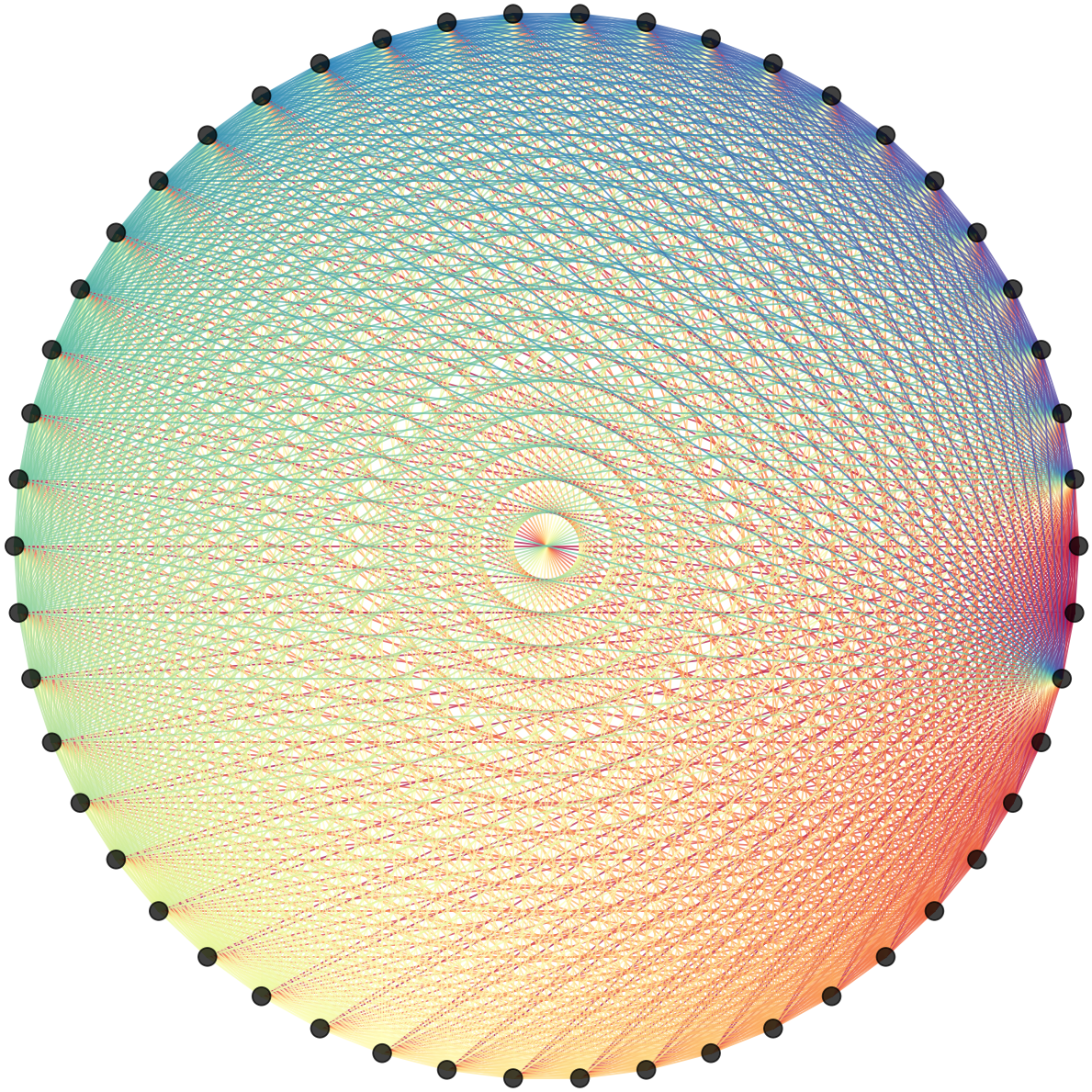
```
eigen_laplacian(n = 100, p = 0.7, q = 0.6)
```



Part C

In [8]:

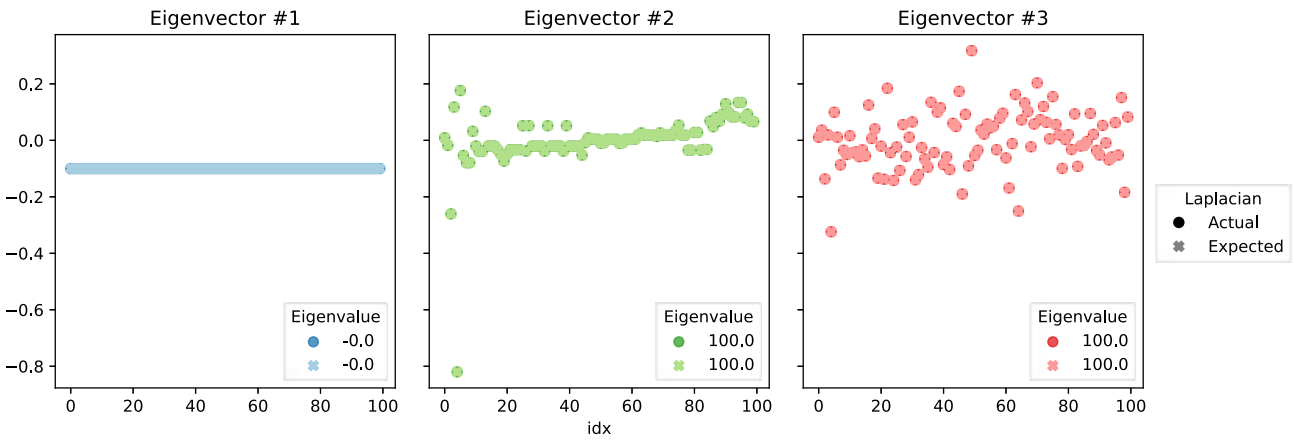
```
draw_stochastic_block(n = 50, p = 1.0, q = 1.0, draw_algo = "circo")
```



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

In [9]:

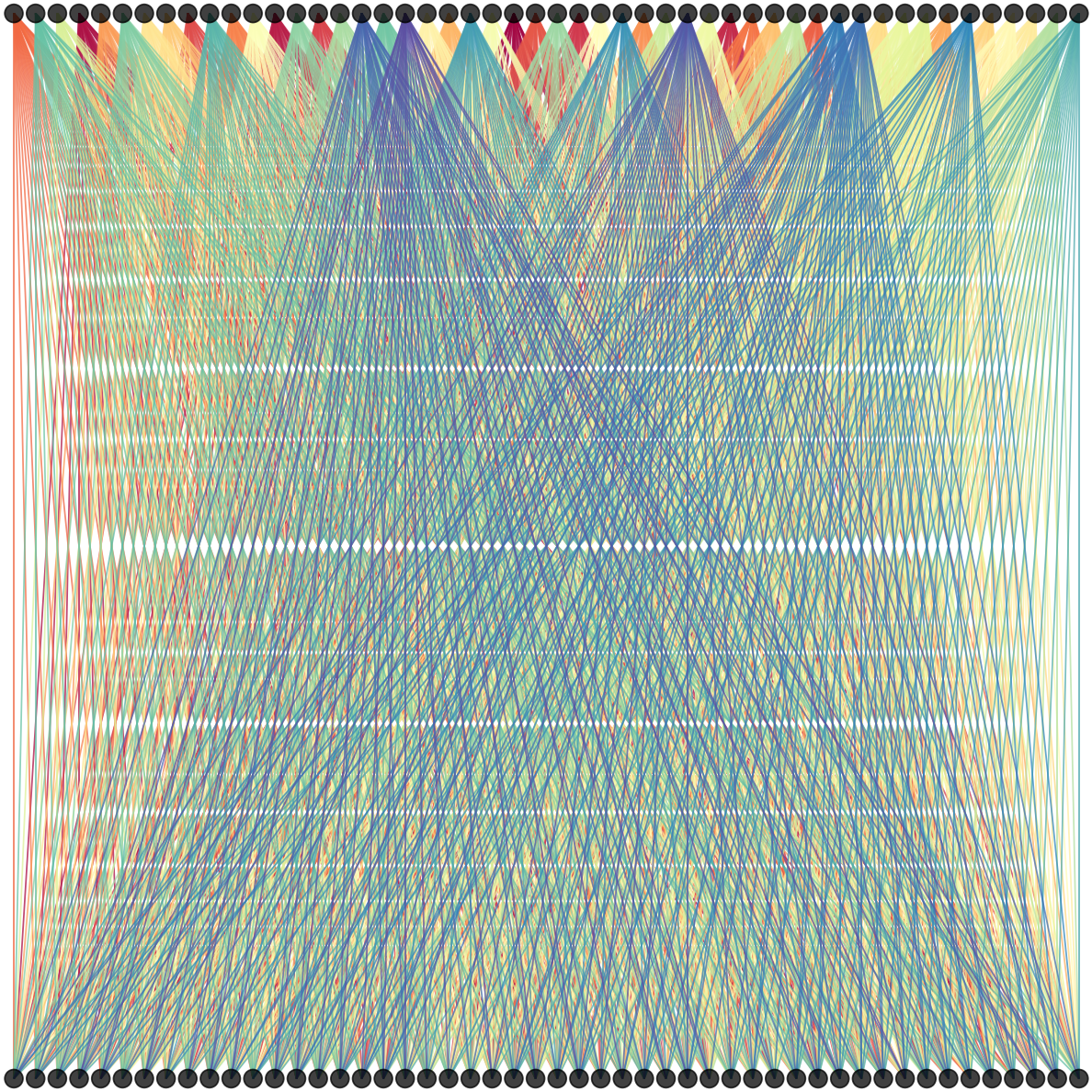
```
eigen_laplacian(n = 100, p = 1.0, q = 1.0)
```



Part D

In [10]:

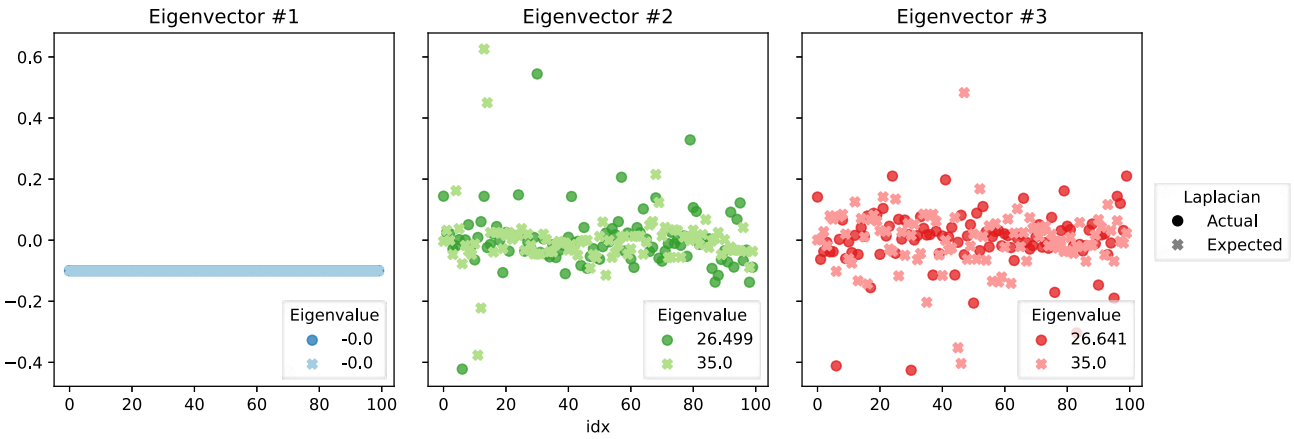
```
draw_stochastic_block(n = 100, p = 0.0, q = 0.7, draw_algo = "dot")
```



(Nearly) Complete Bipartite Graph

In [11]:

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



Problem 7

In [12]:

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

Part A

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.eigh(L_sym.A)

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

In [14]:

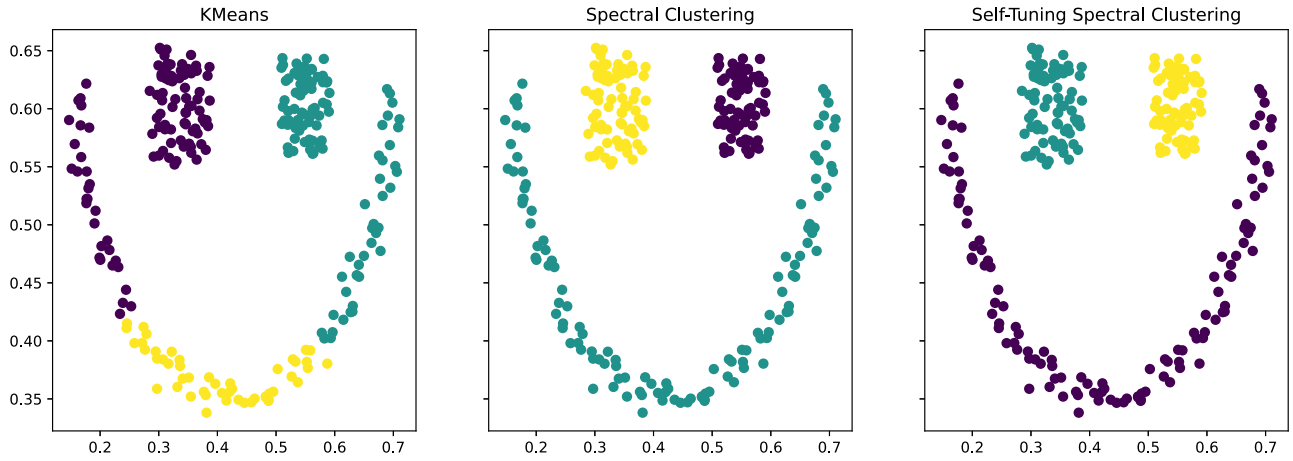
```
def self_tuning_spectral_clustering(data, num_clusters, neighbors_sigma):


  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_sym = csgraph.laplacian(Affinity_hat, normed=True)
  vals, vecs = np.linalg.eigh(L_sym)

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

Part B & C

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]:



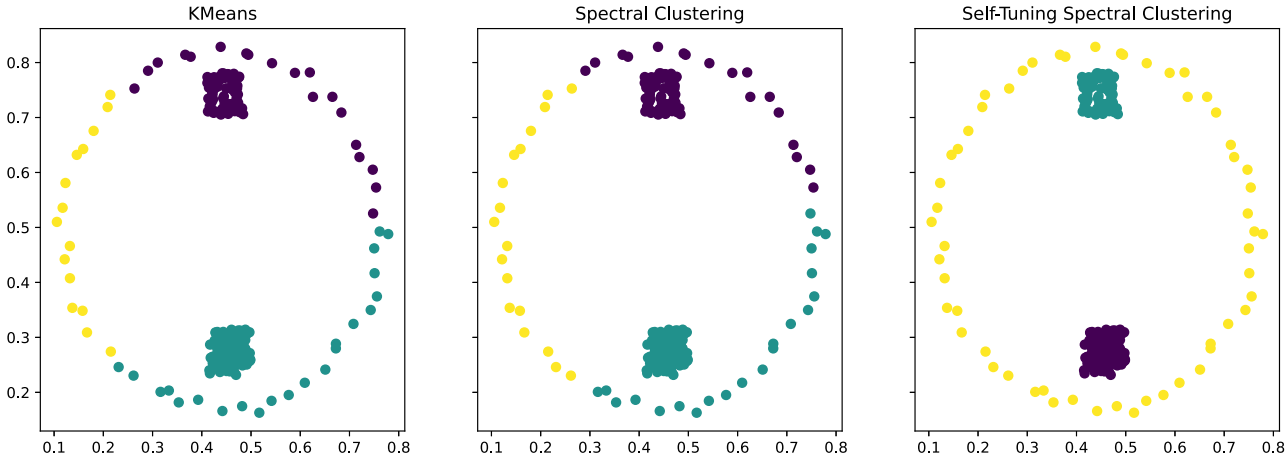
Spectral Clustering: Using k-nearest neighbors with k=5

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

Part D

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, circumscribed 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.

Problem 8

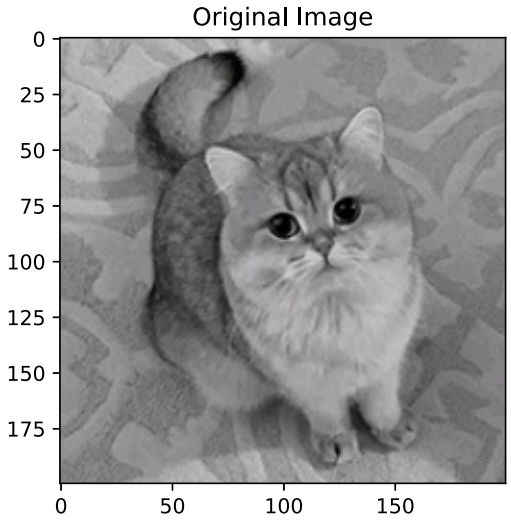
Part A

In [17]:

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

Out[17]:

```
Text(0.5, 1.0, 'Original Image')
```



Part B

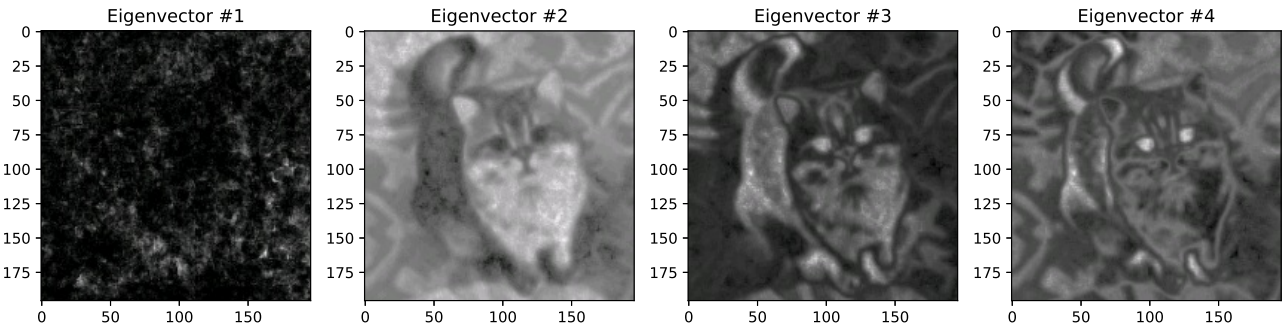
In [18]:

```
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
```

Part C

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')
```



Part D

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)
plt.title('Segmented Image')
```

Out[20]:

```
Text(0.5, 1.0, 'Segmented Image')
```