CS168 Project 6

Henry Lin, Kaylee Bement

1. A.

Line Graph:

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix} - \begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{matrix} = \begin{matrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{matrix}$$

Line Graph with Added Point:

$$\begin{matrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{matrix}$$

Circle Graph:

$$\begin{matrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{matrix}$$

Circle Graph with Added Point:

$$\begin{matrix} 3 & -1 & 0 & 0 & -1 & -1 \\ -1 & 3 & -1 & 0 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & -1 & 3 & -1 \\ -1 & -1 & -1 & -1 & -1 & 5 \end{matrix}$$

B.

B

```
line_a = np.zeros((n, n))
for i in range(n - 1):
        line_a[i][i + 1] = 1
        line_a[i + 1][i] = 1
line_d_vec = [1 if i == 0 or i == n - 1 else 2 for i in range(n)]
line_d = np.diag(line_d_vec)
line_l = line_d - line_a
line_l_eig_vals, line_l_eig_vecs = np.linalg.eig(line_l)
```

```
line_a_eig_vals, line_a_eig_vecs = np.linalg.eig(line_a)
plot_eigenvecs(line_l_eig_vals, line_l_eig_vecs, "Graph A, Laplacian", "1b_a_i")
plot_eigenvecs(line_a_eig_vals, line_a_eig_vecs, "Graph A, Adjacency", "1b_a_ii")

line_add_a = np.zeros((n, n))
for i in range(n - 1):
	line_add_a[i][i + 1] = 1
	line_add_a[i + 1][i] = 1
	line_add_a[n - 1][i] = 1
	line_add_a[i][n - 1] = 1
line_add_d_vec = [2 if i == 0 or i == n - 2 else 3 for i in range(n)]
line_add_d_vec[n - 1] = n - 1
line_add_d = np.diag(line_add_d_vec)
line_add_l = line_add_d - line_add_a
line_add_l_eig_vals, line_add_l_eig_vecs = np.linalg.eig(line_add_l)
line_add_a_eig_vals, line_add_a_eig_vecs = np.linalg.eig(line_add_a)
plot_eigenvecs(line_add_l_eig_vals, line_add_l_eig_vecs, "Graph B, Laplacian",
"1b_b_i")
plot_eigenvecs(line_add_a_eig_vals, line_add_a_eig_vecs, "Graph B, Adjacency",
"1b_b_ii")

circle_a = np.zeros((n, n))
for i in range(1, n - 1):
	circle_a[i][i + 1] = 1
	circle_a[i][i - 1] = 1
	circle_a[i + 1][i] = 1
	circle_a[i - 1][i] = 1
circle_a[n - 1][0] = 1
circle_a[0][n - 1] = 1
circle_d = np.diag([2 for i in range(n)])
circle_l = circle_d - circle_a
circle_l_eig_vals, circle_l_eig_vecs = np.linalg.eig(circle_l)
circle_a_eig_vals, circle_a_eig_vecs = np.linalg.eig(circle_a)
plot_eigenvecs(circle_l_eig_vals, circle_l_eig_vecs, "Graph C, Laplacian", "1b_c_i")
plot_eigenvecs(circle_a_eig_vals, circle_a_eig_vecs, "Graph C, Adjacency", "1b_c_ii")

circle_add_a = np.zeros((n, n))
for i in range(1, n - 2):
	circle_add_a[i][i + 1] = 1
	circle_add_a[i][i - 1] = 1
	circle_add_a[i + 1][i] = 1
	circle_add_a[i - 1][i] = 1
	circle_add_a[i][n - 1] = 1
	circle_add_a[n - 1][i] = 1
```

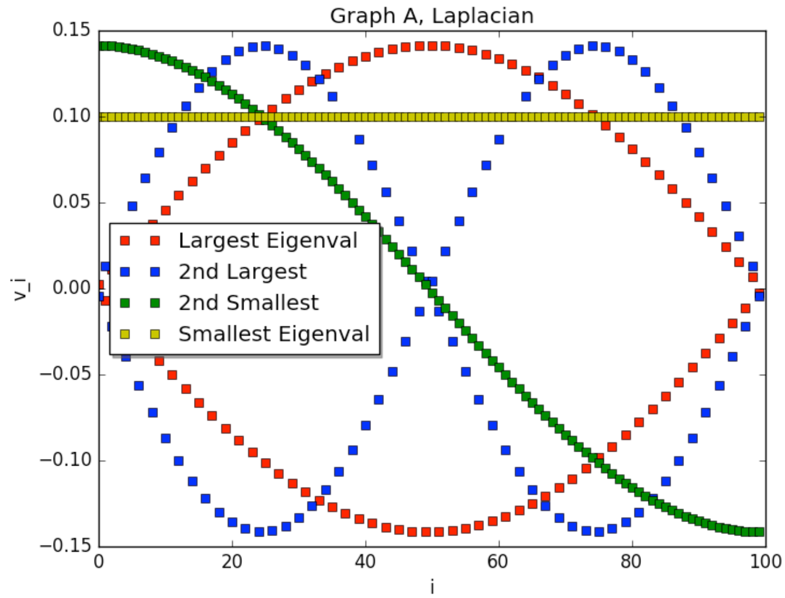
```
circle_add_a[n - 2][0] = 1
circle_add_a[0][n - 2] = 1
circle_add_a[0][n - 1] = 1
circle_add_a[n - 1][0] = 1
circle_add_a[n - 2][n - 1] = 1
circle_add_a[n - 1][n - 2] = 1
circle_add_d = np.diag([3 if i != n - 1 else n - 1 for i in range(n)])
circle_add_l = circle_add_d - circle_add_a
circle_add_l_eig_vals, circle_add_l_eig_vecs = np.linalg.eig(circle_add_l)
circle_add_a_eig_vals, circle_add_a_eig_vecs = np.linalg.eig(circle_add_a)
plot_eigenvecs(circle_add_l_eig_vals, circle_add_l_eig_vecs, "Graph D, Laplacian",
"1b_d_i")
plot_eigenvecs(circle_add_a_eig_vals, circle_add_a_eig_vecs, "Graph D, Adjacency",
"1b_d_ii")
```

In general, v^tLv is the sum of squares of differences between values of neighboring nodes, and eigenvectors corresponding to the lowest eigenvalues minimize the squared distance between neighbors, while those with high eigenvalues maximize discrepancy between neighbor values.

For the Laplacian, the eigenvectors make sense because the smallest eigenvalue vector has the same value since the graphs always have a single connected component, the second smallest eigenvalue vector has a very small distance between each of the points, which will make v^tLv very small, and the two large eigenvalue vectors have very large distances between each of the points, which will make v^tLv very large. It should be noted that the eigenvectors for a given graph and the same graph but with an added point are similar, but each of the vectors that are not of the smallest value have an outlier point separate from the general pattern, and the largest valued eigenvector exploits this by keeping this outlier point as far away from the rest of the points as possible, which then leads to an extremely high eigenvalue for the largest eigenvector. For example, the largest eigenvalue for the line graph is 3.999, while the largest value for the line graph with an added point is 99.999 due to this outlier point.

Line Graph:

- Laplacian
  - Smallest: $7.4397 * 10^{-16}$
  - 2^{nd} Smallest: 0.000987
  - Largest: 3.999
  - 2^{nd} Largest: 3.996



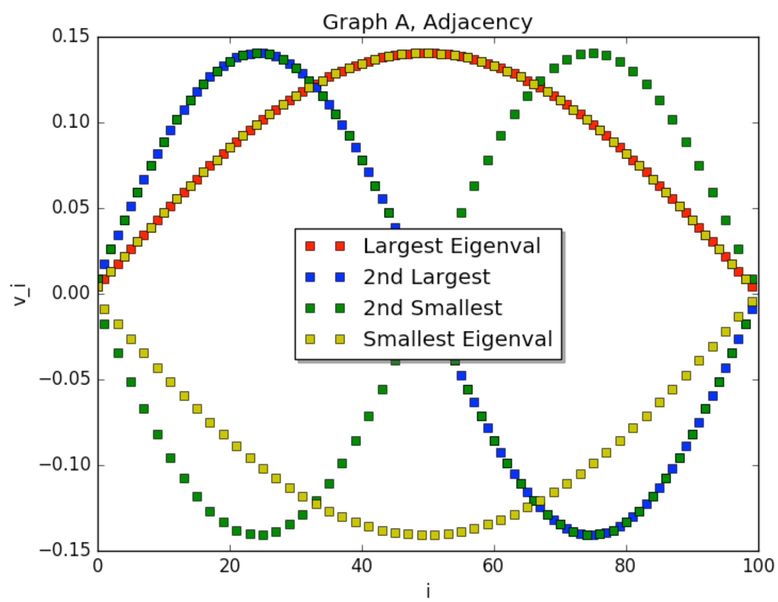
Adjacency

Smallest: -1.999

2nd Smallest: -1.996

Largest: 1.999

2nd Largest: 1.996



Line Graph with Added Point:

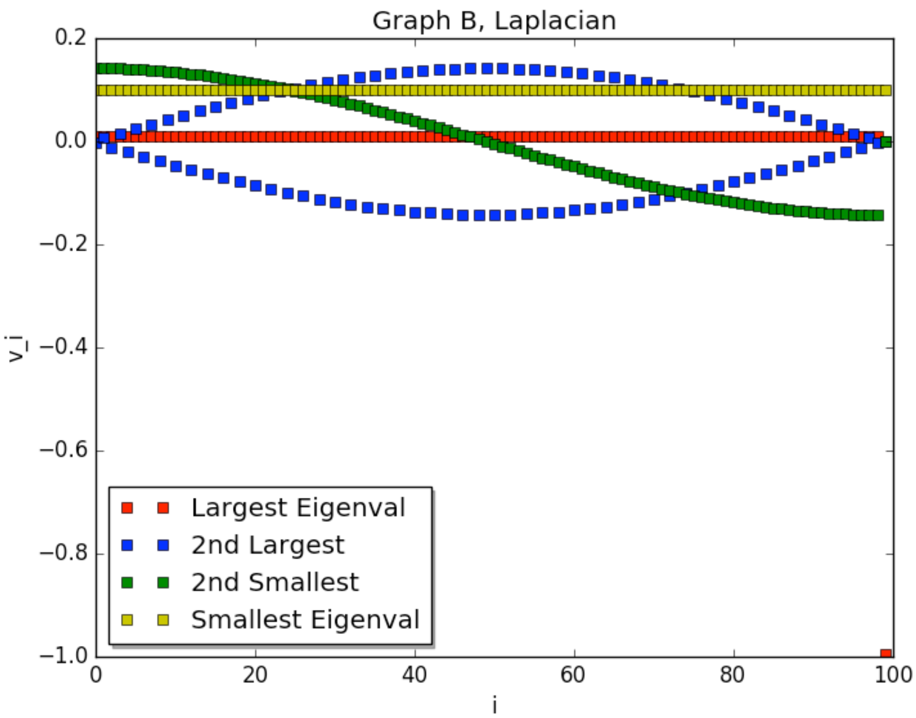
Laplacian

Smallest: -8.919 * 10^{-16}

2nd Smallest: 1.001

Largest: 99.999

2nd Largest: 4.999



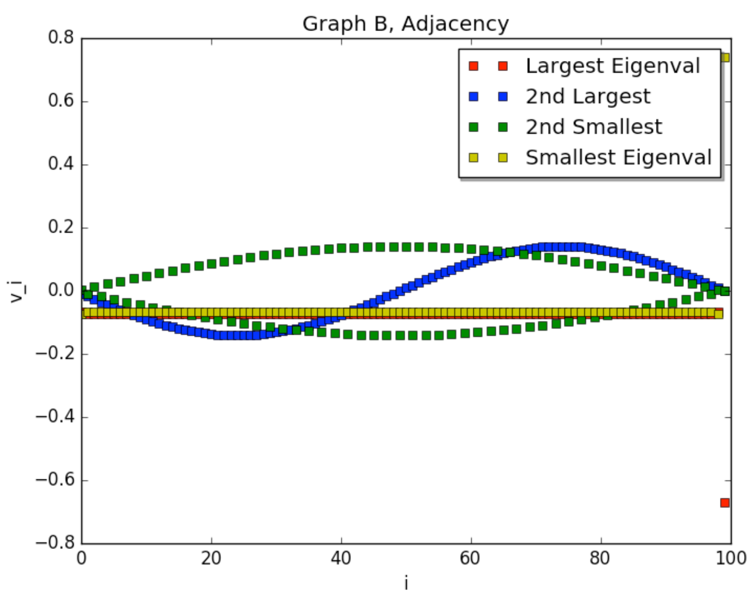
Adjacency

Smallest: -9.01

2^{nd} Smallest: -1.999

Largest: 10.989

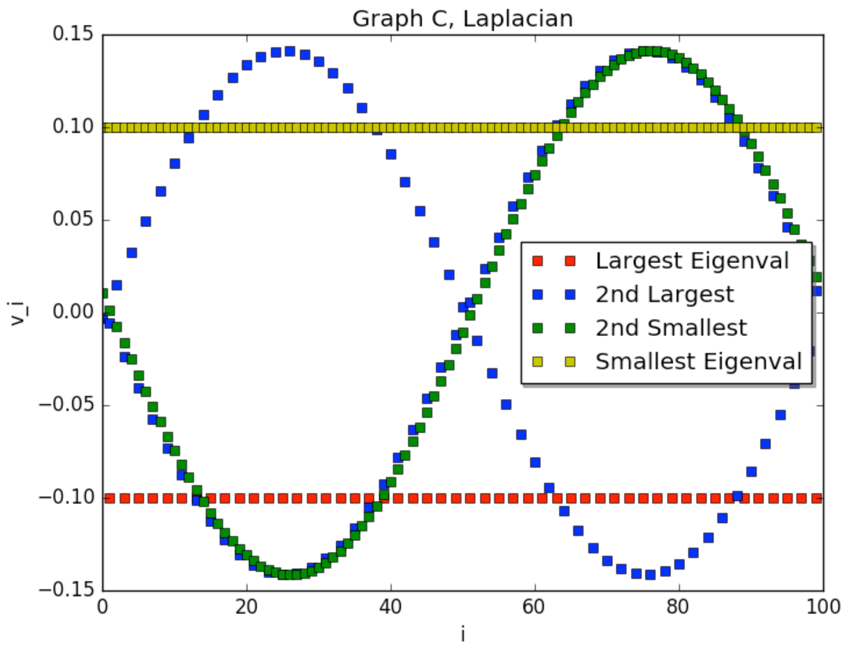
2^{nd} Largest: 1.996



Circle:

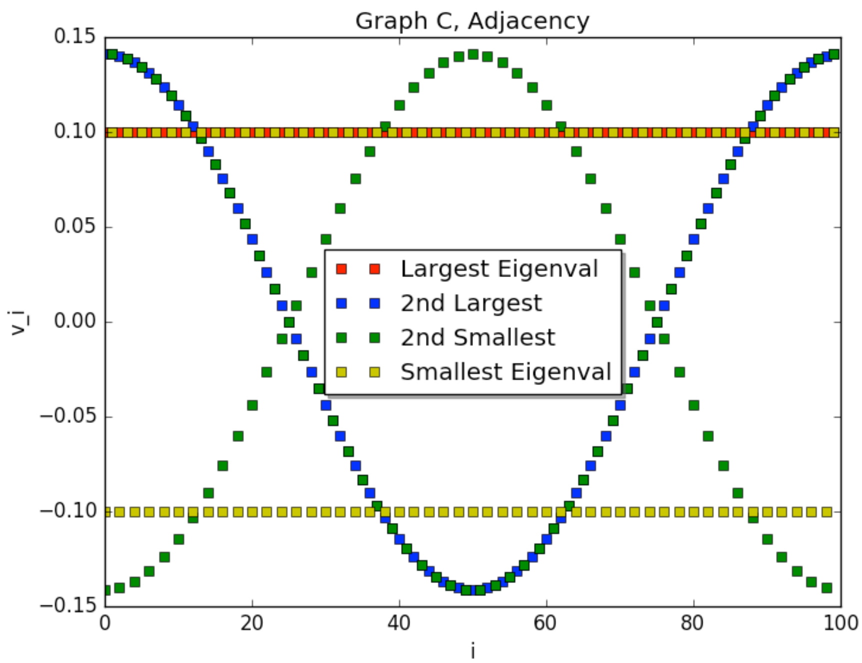
Laplacian

Smallest: $1.332 * 10^{-15}$
2^{nd} Smallest: 0.0039
Largest: 3.9999
2^{nd} Largest: 3.996



Adjacency

Smallest: -1.999
2^{nd} Smallest: -1.996
Largest: 2.000
2^{nd} Largest: 1.996



Circle with Added Point:

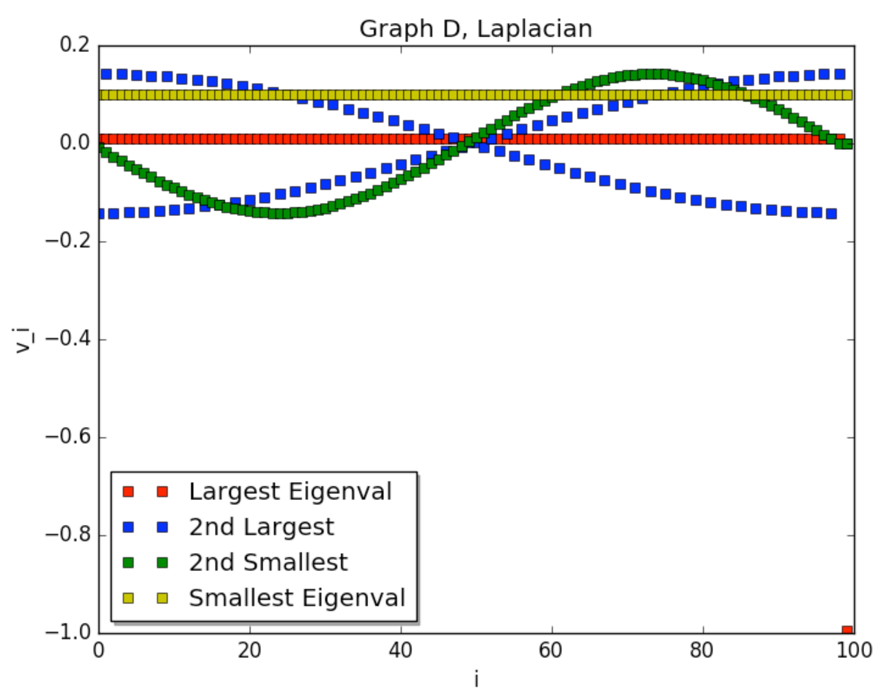
Laplacian

Smallest: -2.345 * 10^{-15}

2^{nd} Smallest: 1.004

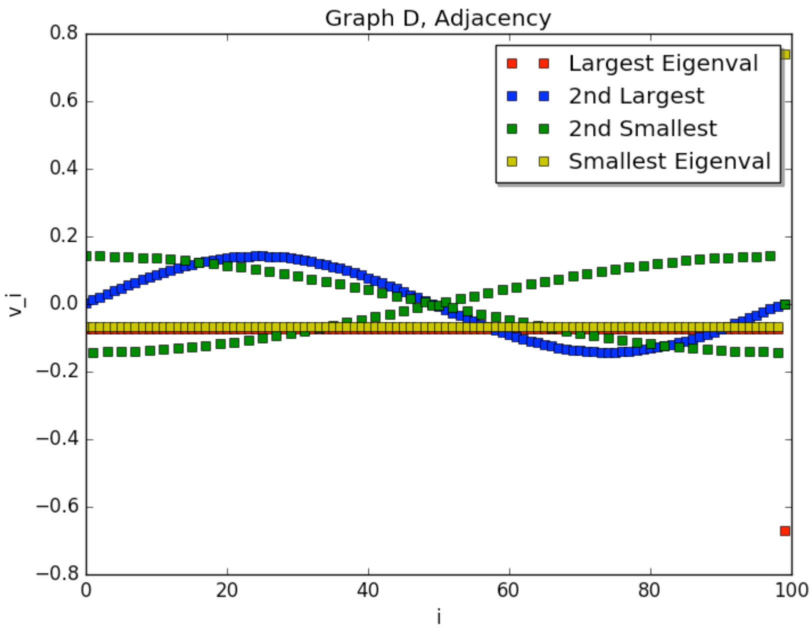
Largest: 99.999

2^{nd} Largest: 4.999

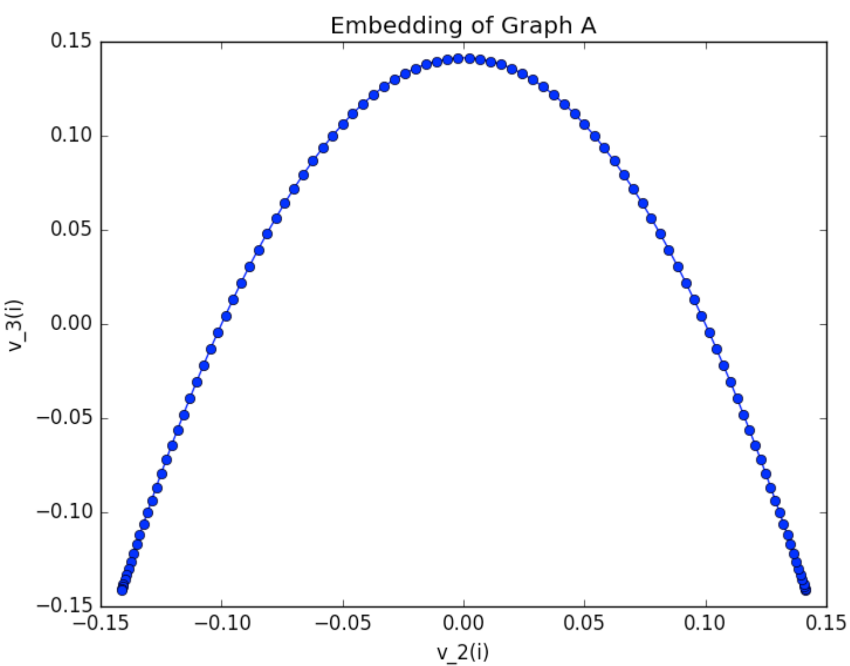


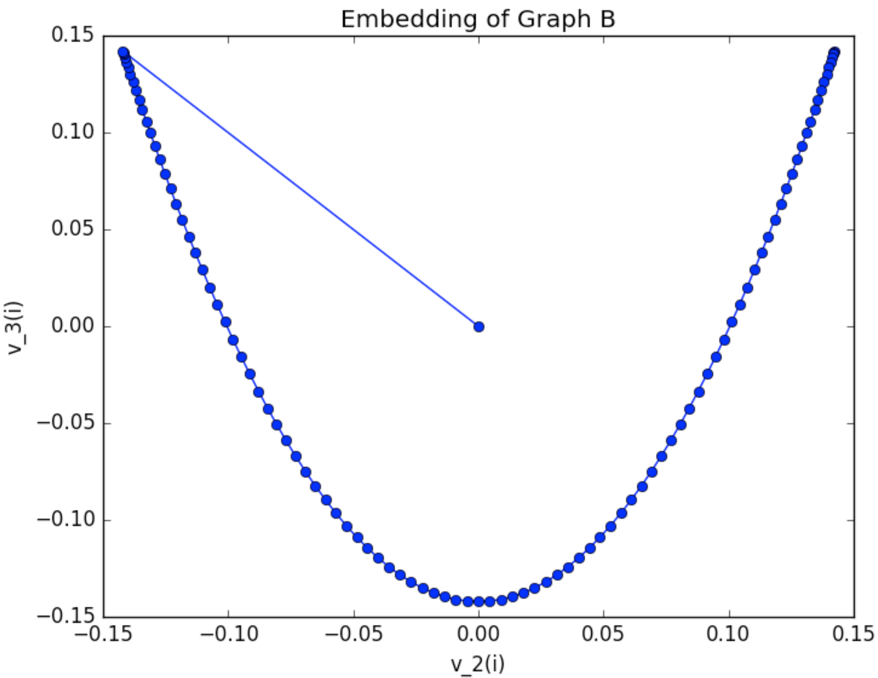
Adjacency

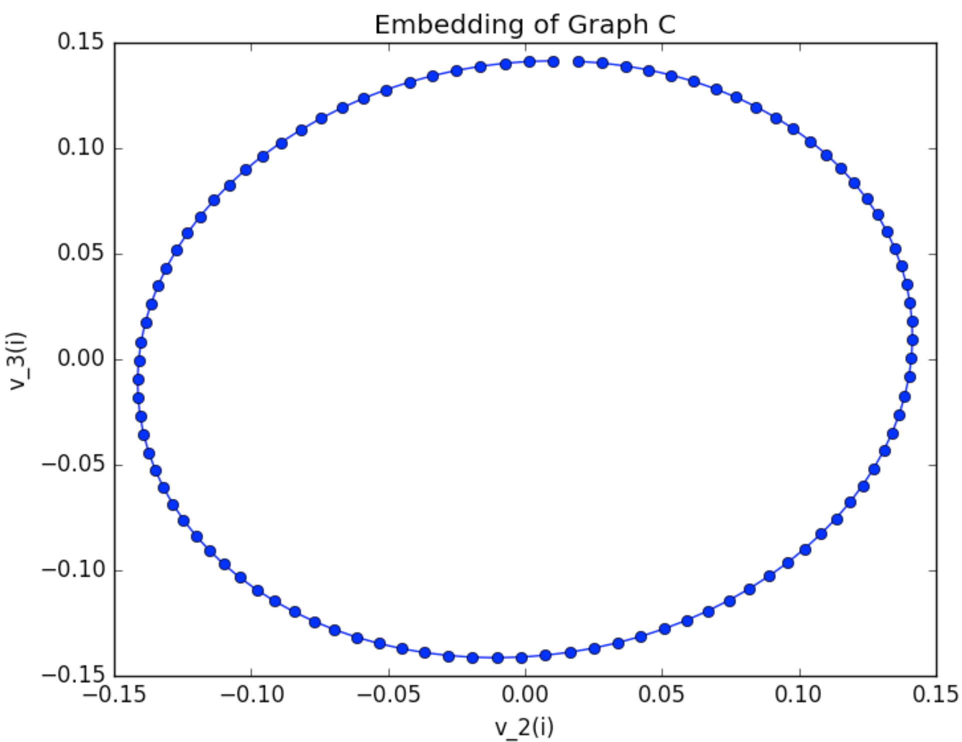
Smallest: -9.00
2nd Smallest: -1.999
Largest: 10.999
2nd Largest: 1.996

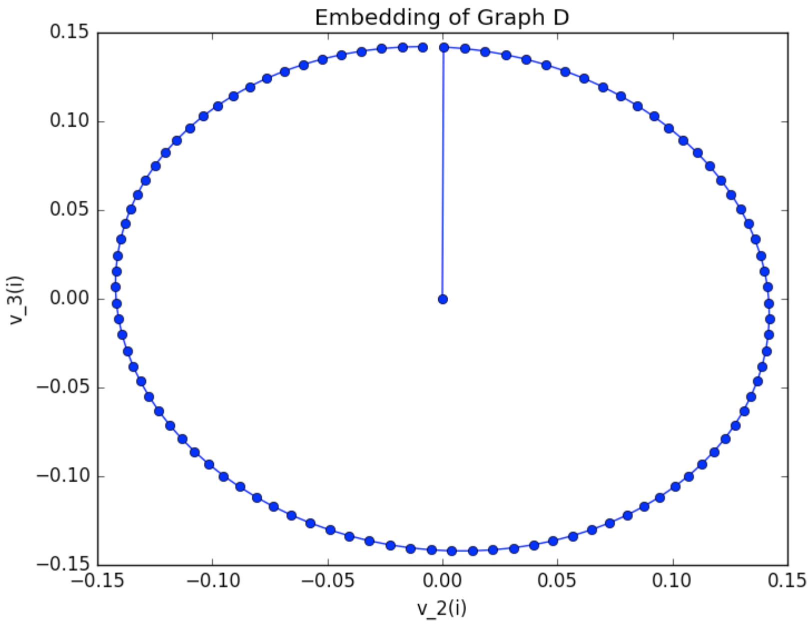


C.

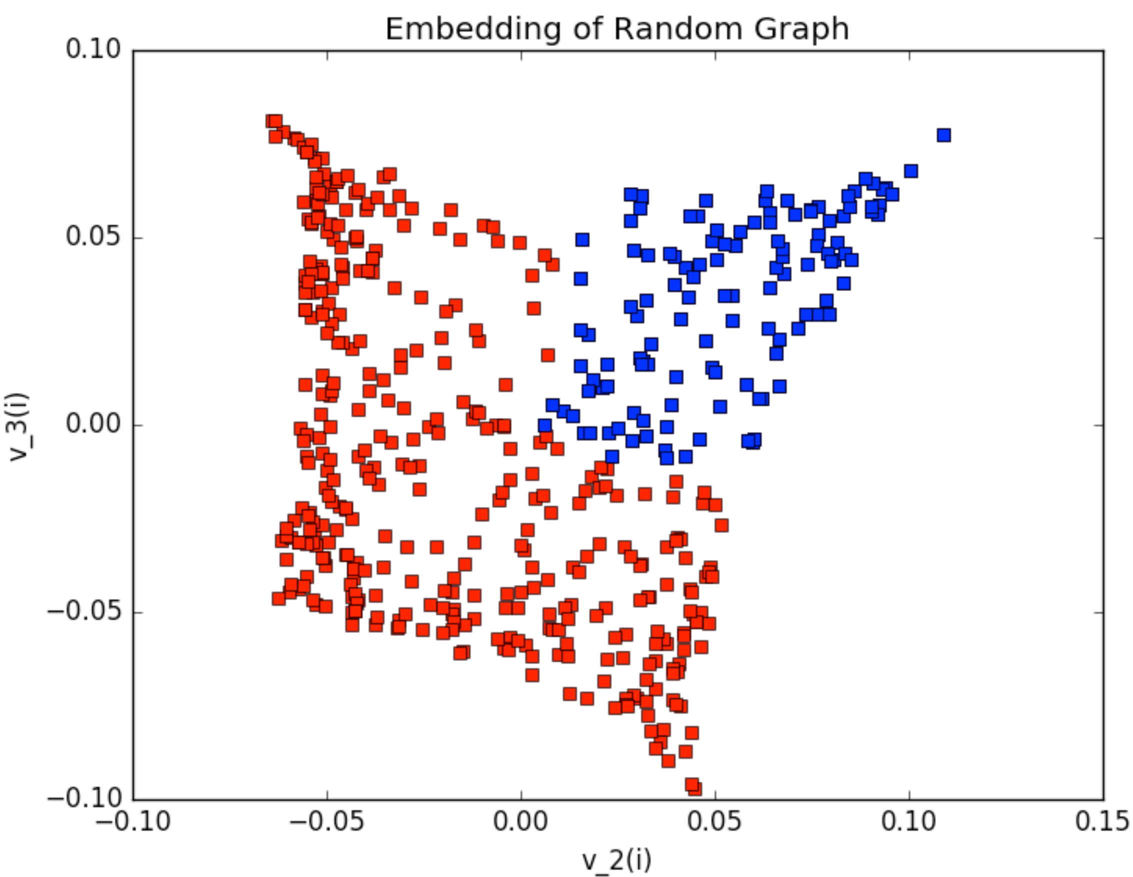



Embedding of Graph B
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
v_3(i)
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
v_2(i)


Embedding of Graph C
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
v_3(i)
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
v_2(i)



D.



The blue squares represent the images of points where both coordinates are less than 1/2. These points are clustered together in the embedding. Since low eigenvectors are trying to find ways of assigning different values to vertices such that neighbors have similar values, looking at the embedding of low eigenvectors is a good way to find clusterings such as those where both coordinates of a point are less than ½ in this case since all such points will be neighbors (as they all have a distance of ¼ or less from each other).

2. A.

   B.

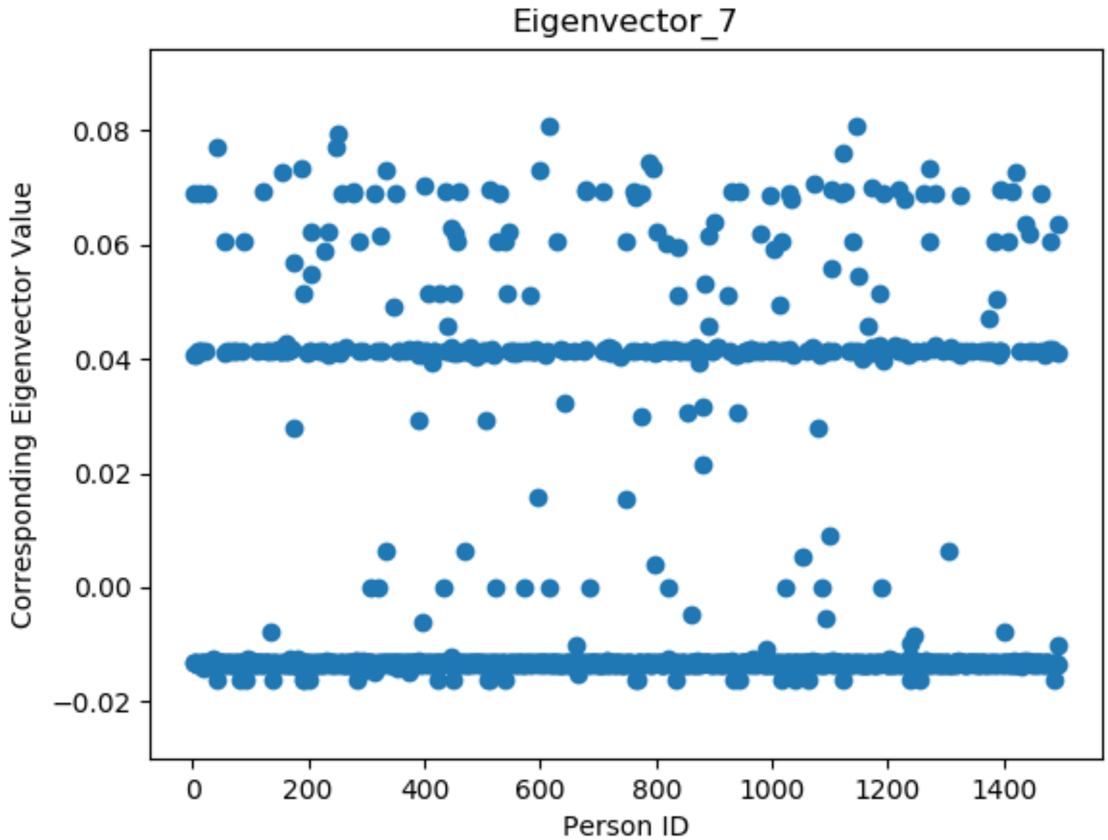
```
unique_people = 1495
def process_array_into_D_and_A(array):
	D = np.zeros((unique_people, unique_people)) #degree matrixx
	A = np.zeros_like(D) #adjacency matrix
	for row in range(array.shape[0]): #iterating through each row
		person = array[row][0]
		#print("person: ", person)
		friend = array[row][1]
		D[person - 1, person - 1] += 1
		A[person - 1, friend - 1] = 1


	return D, A

def plot_two_eigenvectors(vector1, vector2, vector1_name, vector2_name, title, filename):
	#vector 1 should be bigger eigenvector
	plt.scatter(vector1, vector2)
	plt.xlabel(vector1_name)
	plt.ylabel(vector2_name)
	plt.title(title)
	plt.savefig(filename + ".png", format = 'png')
	plt.close()

def plot_eigenvector_vs_person(eigenvector, title, filename):
	number_people = eigenvector.shape[0]
	people_list = [x for x in range(1, number_people + 1)]
	plt.scatter(people_list, eigenvector)
	plt.xlabel("Person ID")
	plt.ylabel("Corresponding Eigenvector Value")
	plt.title(title)
	plt.savefig(filename + ".png", format = 'png')
```

```
    plt.close()

#print("Shape of friendship array: ", friendship_array.shape)
D, A = process_array_into_D_and_A(friendship_array)

Laplacian = D - A
#print("Laplacian: ", Laplacian)

eigenvalues, eigenvectors = np.linalg.eig(Laplacian)
#eigenvalues_list = sorted(eigenvalues.tolist())
idx = eigenvalues.argsort() #[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
#eigenvectors = np.log(eigenvectors)



# smallest_eigenvector = eigenvectors[0, :]
# second_smallest_eigenvector = eigenvectors[1, :]
# third_smallest_eigenvector = eigenvectors[2, :]

#plot_two_eigenvectors(smallest_eigenvector, second_smallest_eigenvector, "1st
eigenvector", "2nd eigenvector", "Smallest Eigenvectors", "2b")
#plot_two_eigenvectors(second_smallest_eigenvector, third_smallest_eigenvector,
"2nd eigenvector", "3rd eigenvector", "Smallest Eigenvectors", "2b_2")
#plot_two_eigenvectors(eigenvectors[2, :], eigenvectors[3, :], "3rd eigenvector", "4th
eigenvector", "Smallest Eigenvectors", "2b_3")
#plot_two_eigenvectors(eigenvectors[6, :], eigenvectors[7, :], "7th eigenvector", "8th
eigenvector", "Smallest Eigenvectors", "2b_5")
#plot_two_eigenvectors(eigenvectors[7, :], eigenvectors[8, :], "8th eigenvector", "9th
eigenvector", "Smallest Eigenvectors", "2b_6")

#plot_eigenvector_vs_person(eigenvectors[7, :], "2nd Smallest Eigenvector", "2b_11")
#plot_eigenvector_vs_person(eigenvectors[14, :], "15th Eigenvector", "2b_15th")

#list of smallest eigenvalues
#List of eigenvalues:
#[-8.176427170721321e-14, -2.2035097765727694e-14, -8.880769415071853e-15,
#4.355158870870788e-15, 6.732637604348797e-14, 8.577736296227727e-14,
#0.014304016619435813, 0.05379565273704709, 0.07390297669255014,
0.0812896697122266,
#0.12022393183749233, 0.13283886699780015]
```

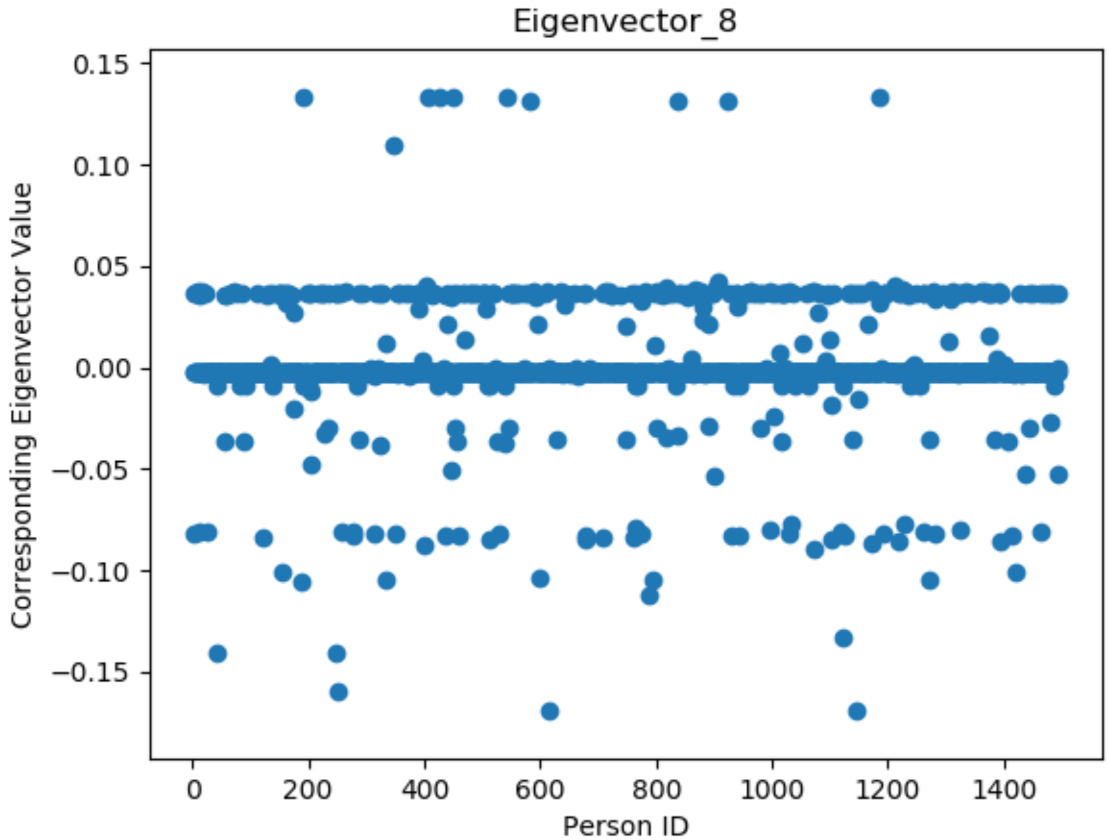
[-8.176427170721321e-14, -2.2035097765727694e-14, -8.880769415071853e-15, 4.355158870870788e-15, 6.732637604348797e-14, 8.577736296227727e-14, 0.014304016619435813, 0.05379565273704709, 0.07390297669255014, 0.0812896697122266, 0.12022393183749233, 0.13283886699780015]

C.
There seem to be 6 connected components. We can figure this out by looking at the smallest eigenvalues. According to the theorem from lecture, we know that the number of zero eigenvectors correspond to the number of connected components. The e-14 numbers are most likely 0 eigenvectors that have been slightly distorted by randomness and numerical instability. Using eigenvectors, we can also determine which nodes belong to which components. The eigenvector is about 1 divided by the size of the component for every node in that component and 0 elsewhere in the vector. Depending on implementation, numerical instability can cause it to be a very small number too such as 10e-14.

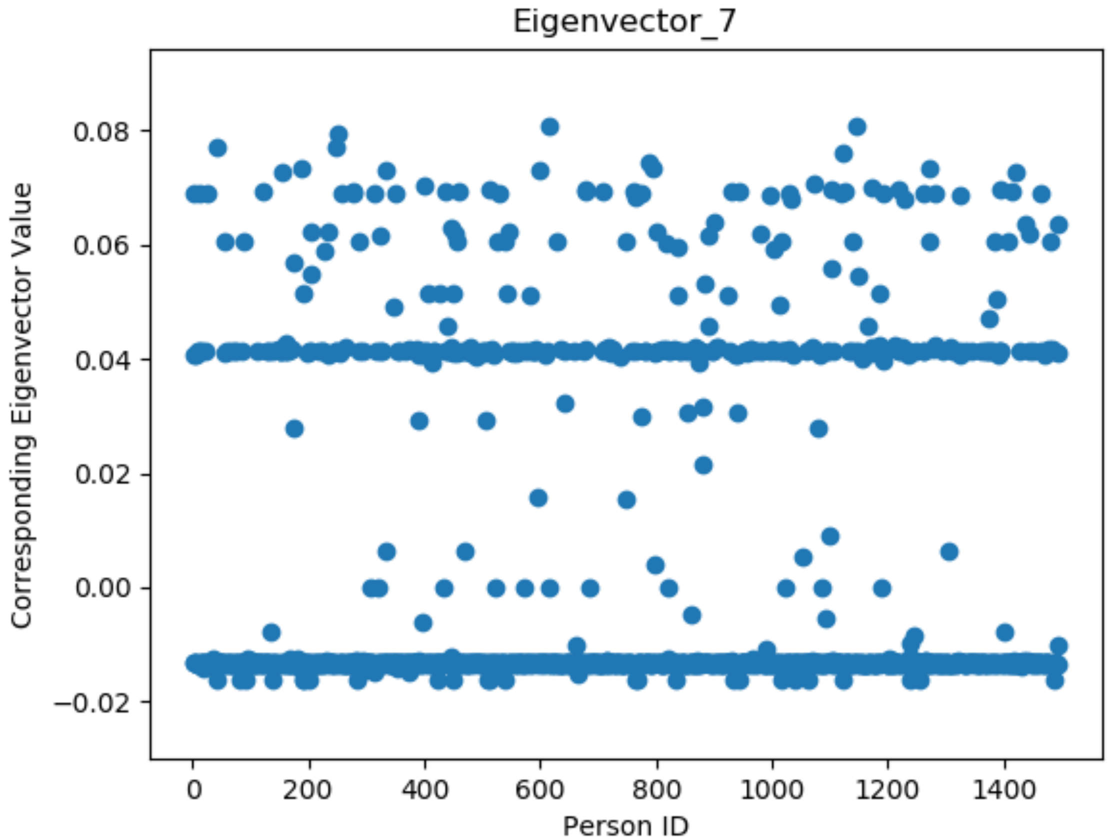
D.
Set 1 : Some nodes are [1, 5, 6, 12, 16, 20, 54, 59, 66, 70]. The size I found for this set was 190 nodes, with a conductance of ~0.0192957. I identified this set by looking at the straight bar of points located around 0.04 in the eigenvector plotted below. Since neighbors are likely to have similar eigenvalues, I thought this would be a good approach to use.



Set 2: Some nodes are [0, 2, 4, 6, 8, 9, 11, 13, 14, 15, 17]. The size I found for this set was 1162 nodes, with a conductance of ~0.006067. I used the same approach as above except with a different eigenvector, pictured below. I looked at the straight bar of points located around 0.00.



Set 3: Some nodes are [3, 7, 10, 1171, 710, 202, 201, 371, 888, 366]. The size I found for this set was 279 nodes, with a conductance of ~0.0427899. I returned back to a previous used eigenvector, but this time I was interested in the scattered points above the 0.04 bar. I calculated conductance on those points and found that they also met the threshold of maximum conductance.



E.

Conductance of a random set of 150 nodes: 0.936127122604767. Compared to this value, the sets found in Part D seem very tight knit as lower conductance equates to a more tightly knit group. Intuitively, a high conductance is likely when choosing random nodes because it is very plausible that the nodes come from different friend groups that are connected by a single edge or very few edges.

CODE APPENDIX:

```
import pandas as pd
import numpy as np
from scipy.sparse import identity
from scipy.sparse.linalg import eigs
from collections import Counter
import matplotlib
matplotlib.use('Agg')
```

```
import matplotlib.pyplot as plt


# QUESTION 1
n = 100

def get_ordered_evals(vals):
	enumerated = dict(enumerate(vals))
	counter = Counter(enumerated)
	ordered = counter.most_common()
	return ordered

def plot_eigenvecs(vals, vecs, title, filename):
	ordered = get_ordered_evals(vals)
	biggest_vals = ordered[0:2]
	smallest_vals = ordered[-2:]
	print(title)
	print("biggest: ", biggest_vals)
	print("smallest: ", smallest_vals)
	X = [i for i in range(n)]
	plt.plot(X, vecs[:,biggest_vals[0][0]], 'rs', label="Largest Eigenval")
	plt.plot(X, vecs[:,biggest_vals[1][0]], 'bs', label="2nd Largest")
	plt.plot(X, vecs[:,smallest_vals[0][0]], 'gs', label="2nd Smallest")
	plt.plot(X, vecs[:,smallest_vals[1][0]], 'ys', label="Smallest Eigenval")

	plt.title(title)
	plt.xlabel("i")
	plt.ylabel("v_i")
	plt.legend(shadow=True, loc = 0)
	plt.savefig(filename + ".png", format = 'png')
	plt.close()

def plot_embeddings_c(vals, vecs, title, filename):
	ordered = get_ordered_evals(vals)
	v2 = ordered[-2]
	v3 = ordered[-3]

	plt.plot(vecs[:,v2[0]], vecs[:,v3[0]], '-o')
	plt.title(title)
	plt.xlabel("v_2(i)")
	plt.ylabel("v_3(i)")
	plt.savefig(filename + ".png", format = 'png')
	plt.close()
```

```
def plot_embeddings_d(vals, vecs, points, title, filename):
	ordered = get_ordered_evals(vals)
	v2 = ordered[-2]
	v3 = ordered[-3]

	plt.title(title)
	plt.xlabel("v_2(i)")
	plt.ylabel("v_3(i)")

	v2_vec = vecs[:,v2[0]]
	v3_vec = vecs[:,v3[0]]
	plt.plot(v2_vec, v3_vec, 'rs')
	for i in range(len(points)):
		if points[i][0] < 0.5 and points[i][1] < 0.5:
			plt.plot(v2_vec[i], v3_vec[i], 'bs')
	plt.savefig(filename + ".png", format = 'png')
	plt.close()


# B
# line_a = np.zeros((n, n))
# for i in range(n - 1):
#	line_a[i][i + 1] = 1
#	line_a[i + 1][i] = 1
# line_d_vec = [1 if i == 0 or i == n - 1 else 2 for i in range(n)]
# line_d = np.diag(line_d_vec)
# line_l = line_d - line_a
# line_l_eig_vals, line_l_eig_vecs = np.linalg.eig(line_l)
# line_a_eig_vals, line_a_eig_vecs = np.linalg.eig(line_a)
# plot_eigenvecs(line_l_eig_vals, line_l_eig_vecs, "Graph A, Laplacian", "1b_a_i")
# plot_eigenvecs(line_a_eig_vals, line_a_eig_vecs, "Graph A, Adjacency", "1b_a_ii")

# line_add_a = np.zeros((n, n))
# for i in range(n - 1):
#	line_add_a[i][i + 1] = 1
#	line_add_a[i + 1][i] = 1
#	line_add_a[n - 1][i] = 1
#	line_add_a[i][n - 1] = 1
# line_add_d_vec = [2 if i == 0 or i == n - 2 else 3 for i in range(n)]
# line_add_d_vec[n - 1] = n - 1
# line_add_d = np.diag(line_add_d_vec)
# line_add_l = line_add_d - line_add_a
# line_add_l_eig_vals, line_add_l_eig_vecs = np.linalg.eig(line_add_l)
# line_add_a_eig_vals, line_add_a_eig_vecs = np.linalg.eig(line_add_a)
```

```
# plot_eigenvecs(line_add_l_eig_vals, line_add_l_eig_vecs, "Graph B, Laplacian",
"1b_b_i")
# plot_eigenvecs(line_add_a_eig_vals, line_add_a_eig_vecs, "Graph B, Adjacency",
"1b_b_ii")

# circle_a = np.zeros((n, n))
# for i in range(1, n - 1):
#         circle_a[i][i + 1] = 1
#         circle_a[i][i - 1] = 1
#         circle_a[i + 1][i] = 1
#         circle_a[i - 1][i] = 1
# circle_a[n - 1][0] = 1
# circle_a[0][n - 1] = 1
# circle_d = np.diag([2 for i in range(n)])
# circle_l = circle_d - circle_a
# circle_l_eig_vals, circle_l_eig_vecs = np.linalg.eig(circle_l)
# circle_a_eig_vals, circle_a_eig_vecs = np.linalg.eig(circle_a)
# plot_eigenvecs(circle_l_eig_vals, circle_l_eig_vecs, "Graph C, Laplacian", "1b_c_i")
# plot_eigenvecs(circle_a_eig_vals, circle_a_eig_vecs, "Graph C, Adjacency", "1b_c_ii")

# circle_add_a = np.zeros((n, n))
# for i in range(1, n - 2):
#         circle_add_a[i][i + 1] = 1
#         circle_add_a[i][i - 1] = 1
#         circle_add_a[i + 1][i] = 1
#         circle_add_a[i - 1][i] = 1
#         circle_add_a[i][n - 1] = 1
#         circle_add_a[n - 1][i] = 1
# circle_add_a[n - 2][0] = 1
# circle_add_a[0][n - 2] = 1
# circle_add_a[0][n - 1] = 1
# circle_add_a[n - 1][0] = 1
# circle_add_a[n - 2][n - 1] = 1
# circle_add_a[n - 1][n - 2] = 1
# circle_add_d = np.diag([3 if i != n - 1 else n - 1 for i in range(n)])
# circle_add_l = circle_add_d - circle_add_a
# circle_add_l_eig_vals, circle_add_l_eig_vecs = np.linalg.eig(circle_add_l)
# circle_add_a_eig_vals, circle_add_a_eig_vecs = np.linalg.eig(circle_add_a)
# plot_eigenvecs(circle_add_l_eig_vals, circle_add_l_eig_vecs, "Graph D, Laplacian",
"1b_d_i")
# plot_eigenvecs(circle_add_a_eig_vals, circle_add_a_eig_vecs, "Graph D, Adjacency",
"1b_d_ii")

# C
```

```
# plot_embeddings_c(line_l_eig_vals, line_l_eig_vecs, "Embedding of Graph A", "1c_a")
# plot_embeddings_c(line_add_l_eig_vals, line_add_l_eig_vecs, "Embedding of Graph
B", "1c_b")
# plot_embeddings_c(circle_l_eig_vals, circle_l_eig_vecs, "Embedding of Graph C",
"1c_c")
# plot_embeddings_c(circle_add_l_eig_vals, circle_add_l_eig_vecs, "Embedding of
Graph D", "1c_d")

# D
# rand_n = 500
# rand_points = np.random.uniform(size = (rand_n, 2))
# print(rand_points)
# rand_a = np.zeros((rand_n, rand_n))
# for i in range(rand_n):
#	for j in range(i + 1, rand_n):
#		dist = np.linalg.norm(rand_points[i] - rand_points[j])
#		if dist <= 0.25:
#			rand_a[i][j] = 1
#			rand_a[j][i] = 1
# rand_d = np.diag([sum(rand_a[k]) for k in range(rand_n)])
# rand_l = rand_d - rand_a
# rand_l_eig_vals, rand_l_eig_vecs = np.linalg.eig(rand_l)
# plot_embeddings_d(rand_l_eig_vals, rand_l_eig_vecs, rand_points, "Embedding of
Random Graph", "1d")


#QUESTION 2

#A

def read_csv(csv_name):
	df = pd.read_csv(csv_name, header = None)
	return df.as_matrix()


friendship_array = read_csv("cs168mp6.csv")
print("shape of array for 2: ", friendship_array.shape)


#B

unique_people = 1495
def process_array_into_D_and_A(array):
	D = np.zeros((unique_people, unique_people)) #degree matrixx
```

```
    A = np.zeros_like(D) #adjacency matrix
    for row in range(array.shape[0]): #iterating through each row
        person = array[row][0]
        #print("person: ", person)
        friend = array[row][1]
        D[person - 1, person - 1] += 1
        A[person - 1, friend - 1] = 1


    return D, A

def plot_two_eigenvectors(vector1, vector2, vector1_name, vector2_name, title,
filename):
    #vector 1 should be bigger eigenvector
    plt.scatter(vector1, vector2)
    plt.xlabel(vector1_name)
    plt.ylabel(vector2_name)
    plt.title(title)
    plt.savefig(filename + ".png", format = 'png')
    plt.close()

def plot_eigenvector_vs_person(eigenvector, title, filename):
    number_people = eigenvector.shape[0]
    people_list = [x for x in range(1, number_people + 1)]
    plt.scatter(people_list, eigenvector)
    plt.xlabel("Person ID")
    plt.ylabel("Corresponding Eigenvector Value")
    plt.title(title)
    plt.savefig(filename + ".png", format = 'png')
    plt.close()

#print("Shape of friendship array: ", friendship_array.shape)
D, A = process_array_into_D_and_A(friendship_array)

Laplacian = D - A
#print("Laplacian: ", Laplacian)

eigenvalues, eigenvectors = np.linalg.eig(Laplacian)
#eigenvalues_list = sorted(eigenvalues.tolist())
idx = eigenvalues.argsort() #[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
#eigenvectors = np.log(eigenvectors)
```

```
# smallest_eigenvector = eigenvectors[0, :]
# second_smallest_eigenvector = eigenvectors[1, :]
# third_smallest_eigenvector = eigenvectors[2, :]

#plot_two_eigenvectors(smallest_eigenvector, second_smallest_eigenvector, "1st
eigenvector", "2nd eigenvector", "Smallest Eigenvectors", "2b")
#plot_two_eigenvectors(second_smallest_eigenvector, third_smallest_eigenvector,
"2nd eigenvector", "3rd eigenvector", "Smallest Eigenvectors", "2b_2")
#plot_two_eigenvectors(eigenvectors[2, :], eigenvectors[3, :], "3rd eigenvector", "4th
eigenvector", "Smallest Eigenvectors", "2b_3")
#plot_two_eigenvectors(eigenvectors[6, :], eigenvectors[7, :], "7th eigenvector", "8th
eigenvector", "Smallest Eigenvectors", "2b_5")
#plot_two_eigenvectors(eigenvectors[7, :], eigenvectors[8, :], "8th eigenvector", "9th
eigenvector", "Smallest Eigenvectors", "2b_6")

#plot_eigenvector_vs_person(eigenvectors[7, :], "2nd Smallest Eigenvector", "2b_11")
#plot_eigenvector_vs_person(eigenvectors[14, :], "15th Eigenvector", "2b_15th")

#list of smallest eigenvalues
#List of eigenvalues:
#[-8.176427170721321e-14, -2.2035097765727694e-14, -8.880769415071853e-15,
#4.355158870870788e-15, 6.732637604348797e-14, 8.577736296227727e-14,
#0.014304016619435813, 0.05379565273704709, 0.07390297669255014,
0.0812896697122266,
#0.12022393183749233, 0.13283886699780015]


#plot a bunch of eigenvectors:

for i in range(100):
	plot_eigenvector_vs_person(eigenvectors[:, i], "Eigenvector_" + str(i + 1),
"2b_eigenvector_no_log_colum_" + str(i + 1))

# for i in range(100):
#	plot_two_eigenvectors(eigenvectors[i, :], eigenvectors[i + 1, :], "Eigenvector_" +
str(i + 1), "Eigenvector_" + str(i + 2), "Eigenvectors_" + str(i + 1) + "_and_" + str(i + 2),
"2b_eigenvectors_no_log_" + str(i + 1) + "_" + str(i + 2) )
#2D

#networkx time
import networkx as nx
```

```
#eigenvectors = 1000 * eigenvectors
#eigenvectors = np.log(eigenvectors)

#eigenvectors_summed = np.sum(eigenvectors, axis = 1)
# print("shape of summed: ", eigenvectors_summed.shape)
# plot_eigenvector_vs_person(eigenvectors_summed, "Eigenvector Summed",
"2b_SUMMED")

G = nx.from_numpy_matrix(A)

#nx.draw(G)

test_S = [x for x in range(0, 150)]
test_S = []
eigenvector_curr = eigenvectors[:, 6]
print("shape of eigenvector curr: ", eigenvector_curr.shape)
#eigenvector_curr = eigenvector_curr.T
#eigenvector_next = eigenvectors[9, :]
#eigenvectors_temp = eigenvector_curr + eigenvector_next
for i in range(len(eigenvector_curr)):


	if eigenvector_curr[i] > 0.041:
		test_S.append(i)

print("test_S: ", test_S)
other_S = [x for x in range(A.shape[0]) if  x not in test_S]
print("Len of test_S: ", len(test_S))
print("Len of other_S: ", len(other_S))
conduct = nx.algorithms.cuts.conductance(G, test_S, other_S)
print("conductance: ", conduct)

#2E
# print("random")
# random_S = random.sample(range(0, 1495), 150)
# print("Len of Test S: ", len(random_S))
# cond = calculate_conductance(A, random_S)
# print("conductance: ", cond)
print("random")

random_S = random.sample(range(0, 1495), 150)
other_S = [x for x in range(A.shape[0]) if  x not in random_S]
print("Len of test_S: ", len(random_S))
print("Len of other_S: ", len(other_S))
```

```
conduct = nx.algorithms.cuts.conductance(G, other_S, random_S)
print("conductance: ", conduct)

#conductance:  0.936127122604767
```