CS168 Spring Assignment 9

SUNet ID(s): johngold jsalloum
Name(s): John Gold Justin Salloum
Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1: Spectral Methods Intuition

(a) L = D - A

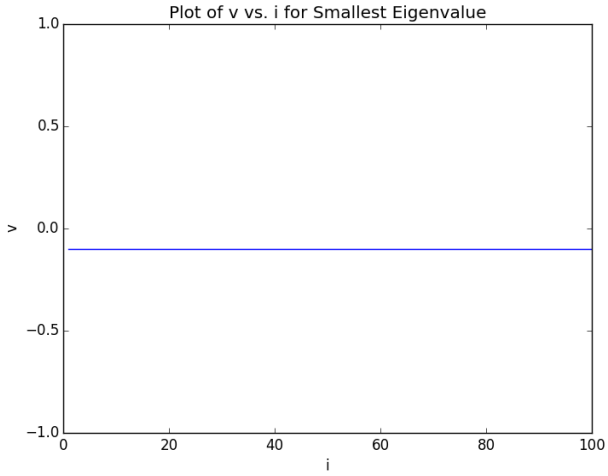
$$(a) = \begin{pmatrix} 1 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 1 \end{pmatrix}$$

$$(b) = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & -1 \\ -1 & 3 & -1 & 0 & \cdots & -1 \\ 0 & -1 & 3 & -1 & \cdots & -1 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ -1 & -1 & \cdots & -1 & -1 & (n-1) \end{pmatrix}$$

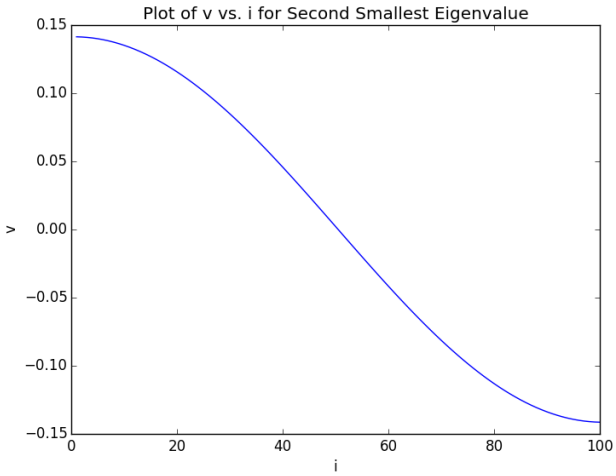
$$(c) = \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & -1 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ -1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

$$(d) = \begin{pmatrix} 3 & -1 & 0 & 0 & \cdots & -1 & -1 \\ -1 & 3 & -1 & 0 & \cdots & 0 & -1 \\ 0 & -1 & 3 & -1 & \cdots & 0 & -1 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots & \vdots \\ -1 & 0 & \cdots & 0 & -1 & 3 & -1 \\ -1 & -1 & \cdots & -1 & -1 & -1 & (n-1) \end{pmatrix}$$

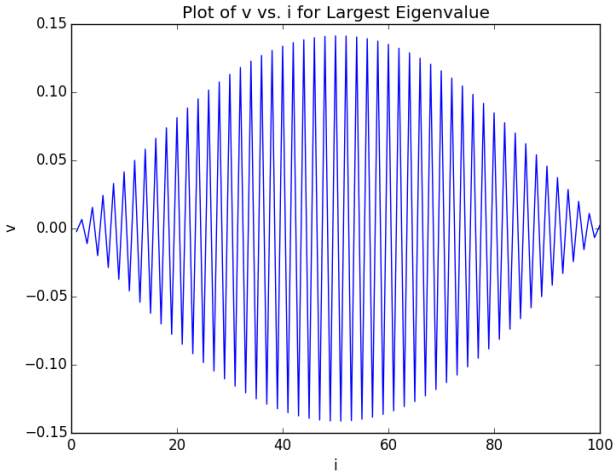
(b) (a) Smallest Eigenval = 3.28739950984e-16



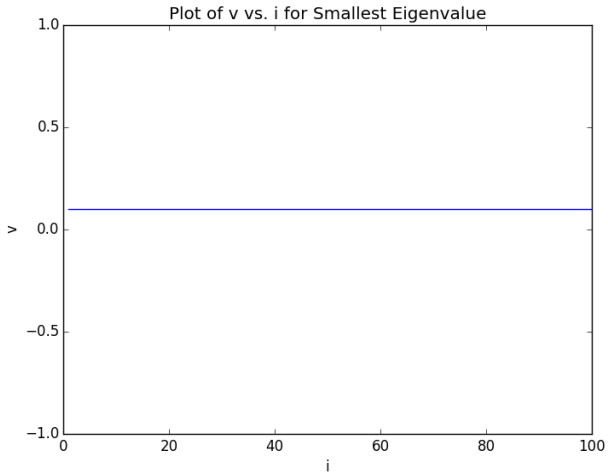
(a) 2nd Smallest Eigenval = 0.000986879268538



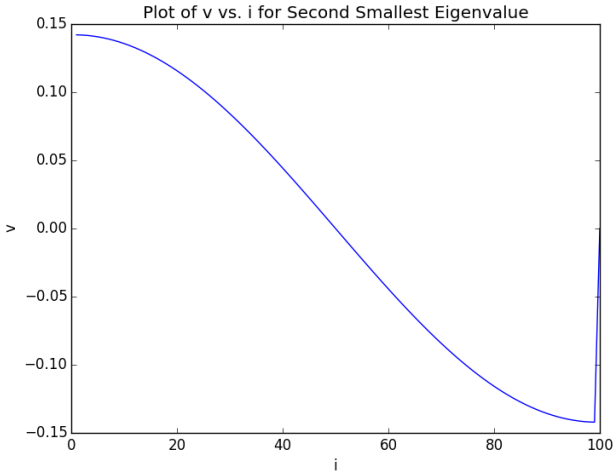
(a) Largest Eigenval = 3.99901312073



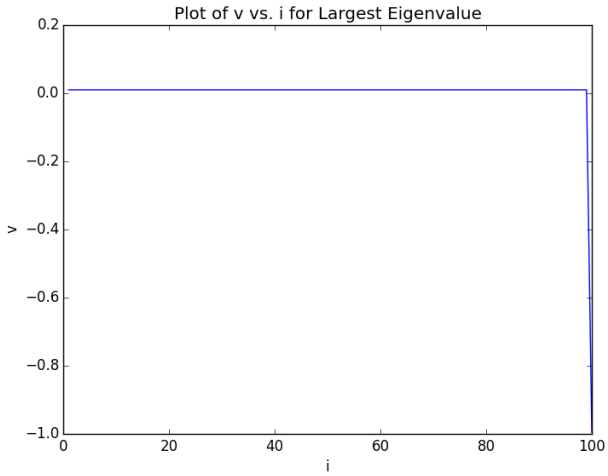
(b) Smallest Eigenval = -2.54088020966e-15



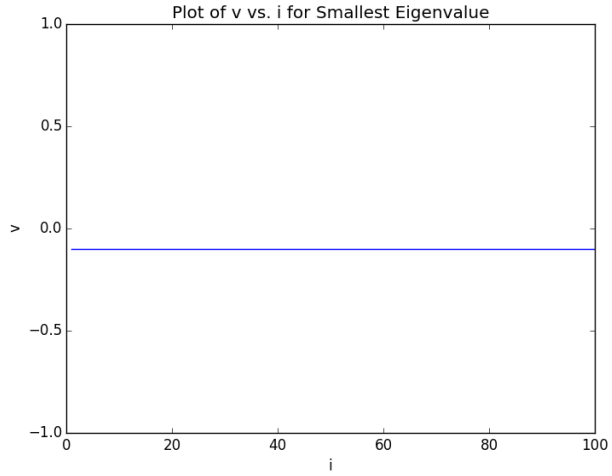
(b) 2nd Smallest Eigenval = 1.00100691523



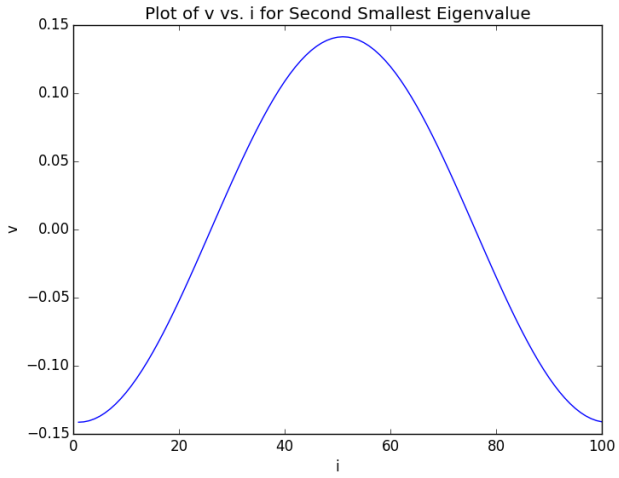
(b) Largest Eigenval = 100



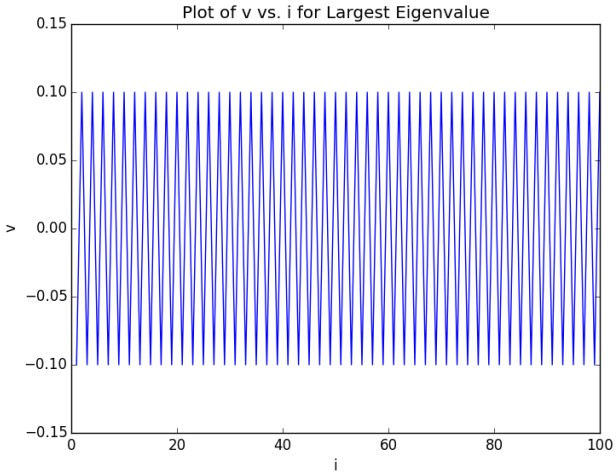
(c) Smallest Eigenval = -1.7763568394e-15



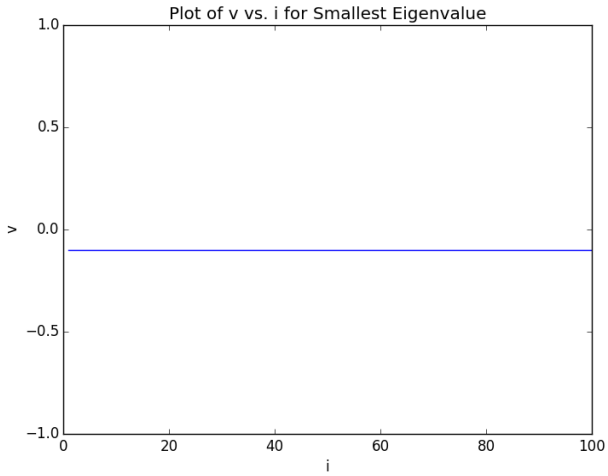
(c) 2nd Smallest Eigenval = 0.00394654314346



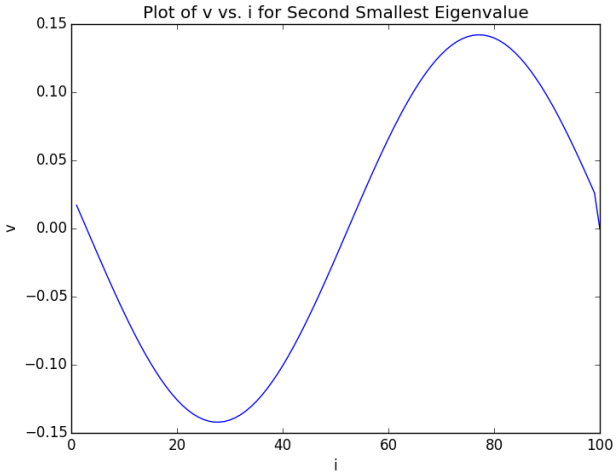
(c) Largest Eigenval = 4.0



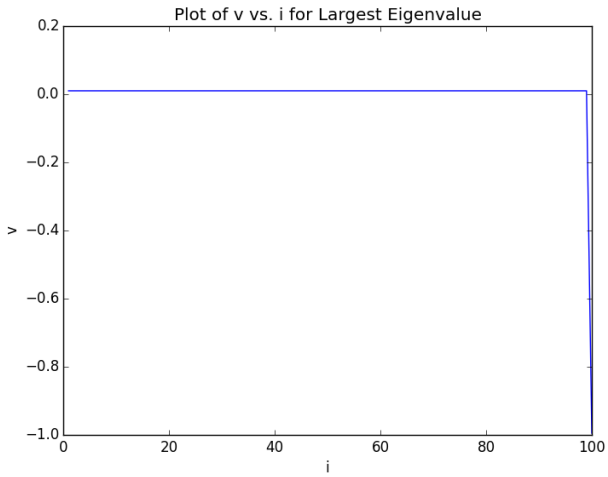
(d) Smallest Eigenval = 5.66864353828e-16



(d) 2nd Smallest Eigenval = 1.00402664706



(d) Largest Eigenval = 100.0

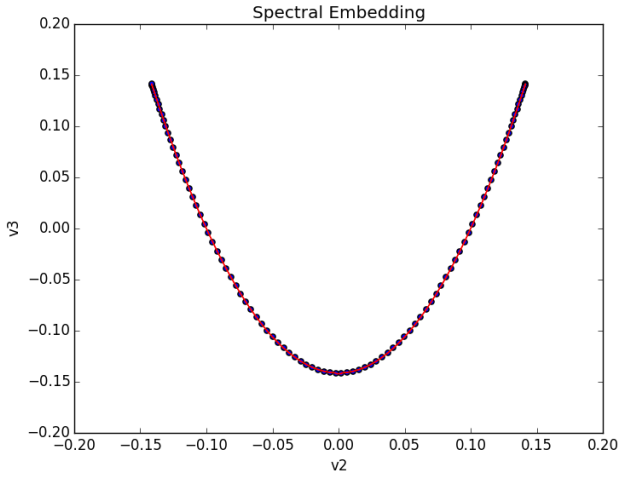



```
n = 100

def graph(graph_func, n):
  A, D, L = graph_func(n)
  print L
  w, v = la.eig(L)
  sorted_w = np.sort(w)
  sorted_indices = np.argsort(w)

  min_index = sorted_indices[0]
  second_min_index = sorted_indices[1]
```

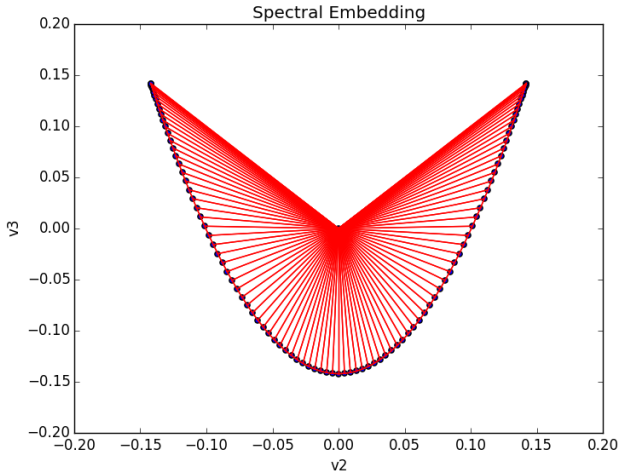
```
    max_index = sorted_indices[n - 1]

    v1, v2, v3 = v[:,min_index], v[:,second_min_index], v[:,max_index]
    x = np.arange(n) + 1
    y = v3

    plt.title('Plot of v vs. i for Largest Eigenvalue')
    plt.xlabel('i')
    plt.ylabel('v')
    plt.plot(x, y)
    plt.show()

graph(graph_a, n)
graph(graph_b, n)
graph(graph_c, n)
graph(graph_d, n)
```

(c) In light of the interpretation of $v^T Lv$, explain why these eigenvectors make sense...***

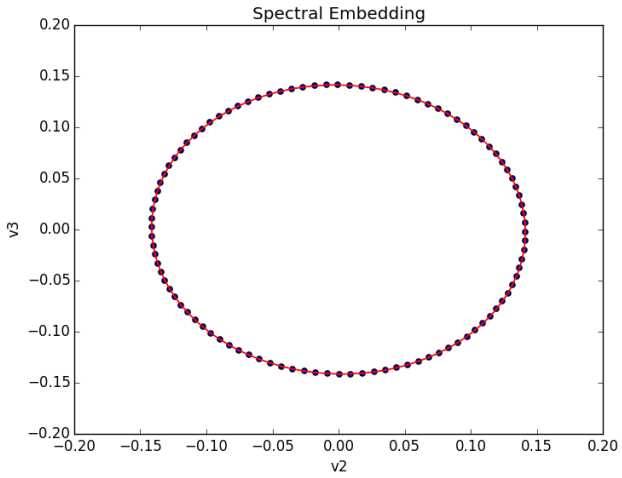
(a) Graph



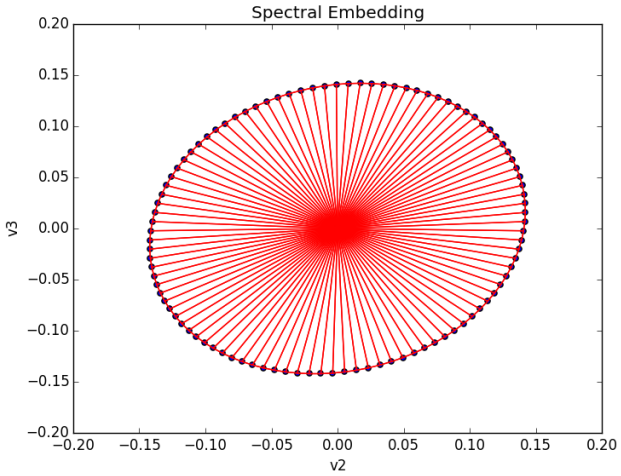
(b) Graph



(c) Graph



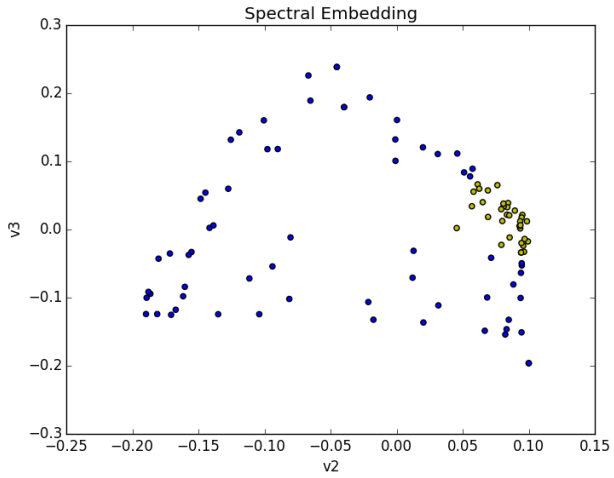
(d) Graph




```
n = 100

def graph(graph_func, n):
  A, D, L = graph_func(n)
  w, v = la.eig(L)
  sorted_w = np.sort(w)
  sorted_indices = np.argsort(w)

  second_min_index = sorted_indices[1]
  third_min_index = sorted_indices[2]

  v2, v3 = v[:,second_min_index], v[:,third_min_index]
  x = v2
  y = v3

  print x
  print y

  plt.title('Spectral Embedding')
  plt.xlabel('v2')
  plt.ylabel('v3')

  plt.scatter(x, y)

  A = np.array(A)
  for i in range(n):
    for j in range(n):
      if A[i][j] == 1:
```

```
        xp, yp = [x[i], x[j]], [y[i], y[j]]
        plt.plot(xp, yp, 'r')
  plt.show()

graph(graph_a, n)
graph(graph_b, n)
graph(graph_c, n)
graph(graph_d, n)
```

(d) Yes it makes sense that the colored points are clustered together....***




```
n = 100

points = np.random.uniform(0, 1, (n, 2))

A = np.zeros((n, n))
for i in range(n):
  for j in range(n):
    norm = la.norm(points[i] - points[j])
    if i != j and norm <= 0.25:
      A[i][j] = 1
d_array = np.zeros(n)

for i in range(n):
  d_array[i] = np.sum(A[i])
D = np.diag(d_array)
```

```
L = D - A
w, v = la.eig(L)
sorted_w = np.sort(w)
sorted_indices = np.argsort(w)

second_min_index, third_min_index = sorted_indices[1], sorted_indices[2]
v2, v3 = v[:,second_min_index], v[:,third_min_index]

plt.title('Spectral Embedding')
plt.xlabel('v2')
plt.ylabel('v3')

colors = ['b' for x in range(100)]
for i in range(n):
  if points[i][0] < 0.5 and points[i][1] < 0.5:
    colors[i] = 'y'
plt.scatter(v2, v3, c=colors)
plt.show()
```

Part 2: Finding Friends

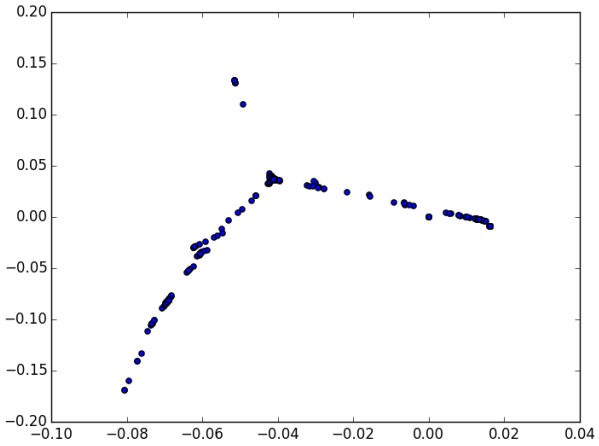
(a) No deliverables

(b)
```
input_file = "cs168mp9.csv"

friendships = {}
data = np.genfromtxt(input_file, dtype='int', delimiter=',')
for row in data:
  f1, f2 = row[0], row[1]
  if f1 not in friendships:
    s = Set()
    friendships[f1] = s
  friendships[f1].add(f2)
  if f2 not in friendships:
    s = Set()
    friendships[f2] = s
  friendships[f2].add(f1)

n = len(friendships)
A = np.zeros((n, n))
for i in friendships:
```

```
  for j in friendships[i]:
    A[i-1][j-1] = 1

d_array = np.zeros(n)
for i in range(n):
  d_array[i] = np.sum(A[i])
D = np.diag(d_array)

L = D - A
w, v = la.eig(L)
```

(c) There are 6 connected components because 6 of the eigenvalues are equal to 0. We can tell which nodes are in which components by plotting the eigenvectors of corresponding to the lowest eigenvalues. You can see the graphs in the next part, in particular the first one of the 6th vs 7th smallest eigenvectors. There is a cluster in the very middle of the inverted triangle object that has over 200 points. We can similarly plot these for other small eigenvectors to find more structure about the original graph.

(d) $S_1 = 304$ points and has a conductance of .0213
Plot of values of eigenvectors corresponding to 6th vs 7th smallest eigenvalues.



$S_2 = 202$ points and has a conductance of .007
$S_3 = 1190$ points and has a conductance of .0367

(e) On a 150 node random sample, we found a conductance of .9. This shows that the sets from part (d) are very close knit in comparison.

CODE APPENDIX - part1a.py

```
def graph_a(n):
  a_array = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      if np.fabs(i - j) == 1:
        a_array[i][j] = 1
  A = a_array
  # print A

  d_array = np.zeros(n)
  for i in range(n):
    if i == 0 or i == n - 1:
      d_array[i] = 1
    else:
      d_array[i] = 2
  D = np.diag(d_array)
  # print D

  L = D - A
  # print L
  return A, D, L

def graph_b(n):
  a_array = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      if np.fabs(i - j) == 1 or ((i == n - 1) != (j == n - 1)):
        a_array[i][j] = 1
  A = a_array
  # print A

  d_array = np.zeros(n)
  for i in range(n):
    if i == n - 1:
      d_array[i] = n - 1
    elif i == 0 or i == n - 2:
      d_array[i] = 2
    else:
      d_array[i] = 3
  D = np.diag(d_array)
```

```
  # print D

  L = D - A
  # print L
  return A, D, L

def graph_c(n):
  a_array = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      if np.fabs(i - j) == 1 or np.fabs(i - j) == n - 1:
        a_array[i][j] = 1
  A = a_array
  # print A

  d_array = np.zeros(n)
  for i in range(n):
    d_array[i] = 2
  D = np.diag(d_array)
  # print D

  L = D - A
  # print L
  return A, D, L

def graph_d(n):
  a_array = np.zeros((n,n))
  for i in range(n):
    for j in range(n):
      if np.fabs(i - j) == 1 or np.fabs(i - j) == n - 2 or ((i == n - 1) != (j == n - 1)
        a_array[i][j] = 1
  A = a_array
  # print A

  d_array = np.zeros(n)
  for i in range(n):
    if i == n - 1:
      d_array[i] = n - 1
    else:
      d_array[i] = 3
  D = np.diag(d_array)
  # print D
```

```
  L = D - A
  # print L
  return A, D, L


graph_d(8)
```