Activity 14

Damion Huppert

In [33]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Question 1-a

In [34]:
```
B = np.array([[3, -1], [-1, 3]])
eigenvalues, eigenvectors = np.linalg.eig(B)
# Define the eigenvectors
e1 = np.array([1, 2])
e2 = np.array([1, 1])
e3 = np.array([1, -1])
e4 = np.array([-1, 1])
e5 = np.array([2, 1])

def check_eigenvector(B, v):
    return np.allclose(B @ v, eigenvalues[0] * v) or np.allclose(B @ v, eige

check_eigenvector_results = {
    'e1': check_eigenvector(B, e1),
    'e2': check_eigenvector(B, e2),
    'e3': check_eigenvector(B, e3),
    'e4': check_eigenvector(B, e4),
    'e5': check_eigenvector(B, e5),
}
check_eigenvector_results
```

Out[34]:
```
{'e1': False, 'e2': True, 'e3': True, 'e4': True, 'e5': False}
```

Question 1-b

In [35]:
```
eigenvalues
```

Out[35]:
```
array([4., 2.])
```

Question 2

The sign of the singular vectors are not unique because we can make u and v negative and the answer would still be the same

3a)

In [36]:
```
# Circle topology
# Unweighted adjacency matrix
```

```
# # Option 1: Manually enter the entries
# Atilde = np.array(
#          [[0,1,0,0,0,0,0,1],
#           [1,0,1,0,0,0,0,0],
#           [1,1,0,1,1,0,0,0],
#           [0,0,1,0,1,0,0,0],
#           [0,0,0,1,0,1,0,0],
#           [0,0,0,0,1,0,1,0],
#           [0,0,0,0,0,1,0,1],
#           [1,0,0,0,0,0,1,0]])

# Option 2: or you can exploit the patterns
Atilde = np.zeros((8,8))
for i in range(8): #
    Atilde[i,(i+1)%8] = 1
    Atilde[i,(i-1)%8] = 1
Atilde[2,0] = 1
Atilde[2,4] = 1

print('Unweighted adjacency matrix')
print(Atilde)
print(' ')
```

```
Unweighted adjacency matrix
[[0. 1. 0. 0. 0. 0. 0. 1.]
 [1. 0. 1. 0. 0. 0. 0. 0.]
 [1. 1. 0. 1. 1. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 1.]
 [1. 0. 0. 0. 0. 0. 1. 0.]]
```

3b)

In [37]:

```
# Find weighted adjacency matrix
# option 1: normalize columns with a for loop
A = np.zeros((8,8), dtype=float)
for k in range(8):
    norm = np.sum(Atilde[:,k])
    A[:,k] = Atilde[:,k] / norm

# option 2: normalize using numpy.sum() and broadcasting, in a single line
# A = ???
np.set_printoptions(precision=3, suppress=True, linewidth=120)  # Control pr
print('Weighted adjacency matrix:')
print(A)
```

```
Weighted adjacency matrix:
[[0.    0.5   0.    0.    0.    0.    0.    0.5  ]
 [0.333 0.    0.5   0.    0.    0.    0.    0.   ]
 [0.333 0.5   0.    0.5   0.333 0.    0.    0.   ]
 [0.    0.    0.5   0.    0.333 0.    0.    0.   ]
 [0.    0.    0.    0.5   0.    0.5   0.    0.   ]
 [0.    0.    0.    0.    0.333 0.    0.5   0.   ]
 [0.    0.    0.    0.    0.    0.5   0.    0.5  ]
 [0.333 0.    0.    0.    0.    0.    0.5   0.   ]]
```

3c) and 3d)

In [38]:

```python
# Power method

b0 = 0.125*np.ones((8,1))
print('b0 = ', b0)
print(' ')

b1 = A @ b0
print('b1 = ', b1)
print(' ')

b = b0.copy()
for k in range(100):
    b =  A @ b

print('1000 iterations')
print('b = ',b)
```

```
b0 =  [[0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]
 [0.125]]

b1 =  [[0.125]
 [0.104]
 [0.208]
 [0.104]
 [0.125]
 [0.104]
 [0.125]
 [0.104]]

1000 iterations
b =  [[0.115]
 [0.154]
 [0.231]
 [0.154]
 [0.115]
 [0.077]
 [0.077]
 [0.077]]
```

3e) Explanation goes here.

Node 3 is more important than others which makes sense because it has the most edges pointing into it

4a)

In [47]:
```
# Hub topology

Atildehub = np.zeros((9,9))
for i in range(9): #
    Atildehub[i,8] = 1
    Atildehub[8,i] = 1
Atildehub[1,0] = 1
print('Unweighted adjacency matrix')
print(Atildehub)
print(' ')
```

```
Unweighted adjacency matrix
[[0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1.]]
```

4b)

In [48]:
```python
# find weighted adjacency matrix

Ahub = np.zeros((9,9), dtype=float)
for k in range(9):
    norm = np.sum(Atildehub[:,k])
    Ahub[:,k] = Atildehub[:,k] / norm

print('Weighted adjacency matrix')
print(Ahub)
```

```
Weighted adjacency matrix
[[0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.5   0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.    0.    0.    0.    0.    0.    0.    0.    0.111]
 [0.5   1.    1.    1.    1.    1.    1.    1.    0.111]]
```

4c) and 4d)

In [49]:
```python
b0 = (1/9)*np.ones((9,1))
print('b0 = ', b0)
print(' ')

bhub1 = Ahub @ b0
print('bhub1 = ', bhub1)
print(' ')

bhub = b0.copy()
for k in range(1000):
    bhub = Ahub @ bhub

print('1000 iterations')
print('bhub = ', bhub)
print(' ')
```

```
bhubr = b0.copy()
for k in range(100):
    bhubr = Ahub @ bhubr

print('100 iterations')
print('bhubr = ',bhubr)
```

```
b0 =  [[0.111]
 [0.111]
 [0.111]
 [0.111]
 [0.111]
 [0.111]
 [0.111]
 [0.111]
 [0.111]]

bhub1 =  [[0.012]
 [0.068]
 [0.012]
 [0.012]
 [0.012]
 [0.012]
 [0.012]
 [0.012]
 [0.846]]

1000 iterations
bhub =  [[0.057]
 [0.086]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.514]]

100 iterations
bhubr =  [[0.057]
 [0.086]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.057]
 [0.514]]
```

Complete 4e and 4f below.

E)

Node 9 is more important than the others and node 1 is more important than the other

non-important ones because they have a higher probability

F)
solved for 100 above