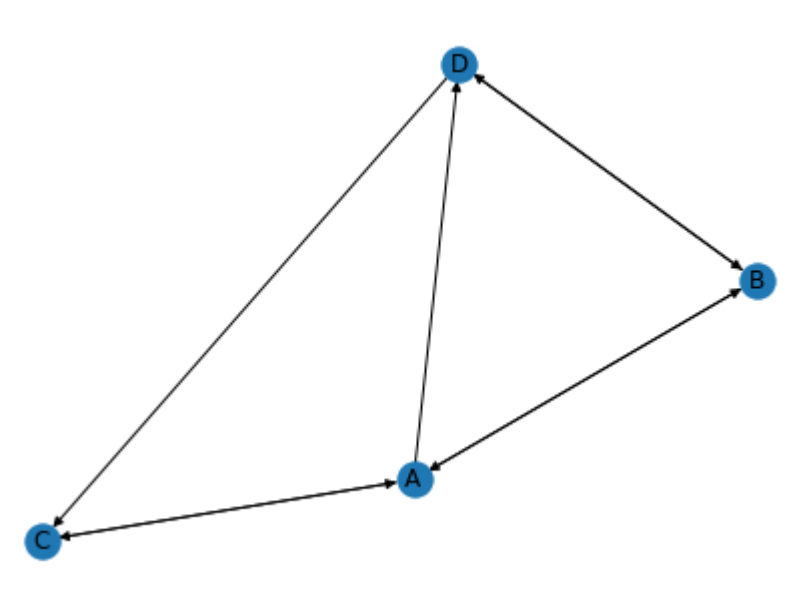
Question1

In [125…
```python
import networkx as nx
import numpy as np
# To create an empty directed graph
G = nx.DiGraph()

# To add a node
G.add_nodes_from('ABCD')

# To add an edge
G.add_edges_from([('A','B'),('A','C'),('A','D'),('B','D'),('B','A'),('C','A'),('D','C'),('D','B')])

node_list = G.nodes()
print("Nodes")
print(node_list)

# To get all the edges of a graph
edge_list = G.edges()
print("Edges")
print(edge_list)

nx.draw(G, with_labels = True)
```

```
Nodes
['A', 'B', 'C', 'D']
Edges
[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'D'), ('B', 'A'), ('C', 'A'), ('D', 'C'), ('D', 'B')]
```



In [126…
```python
A = nx.adjacency_matrix(G)
A = np.array(A.todense())
A = A.transpose()
A
```

Out[126…
```
array([[0, 1, 1, 0],
       [1, 0, 0, 1],
       [1, 0, 0, 1],
       [1, 1, 0, 0]], dtype=int32)
```

In [127…
```python
Tr = A / A.sum(axis=0, keepdims=True)
Tr
```

Out[127…
```
array([[0.        , 0.5       , 1.        , 0.        ],
       [0.33333333, 0.        , 0.        , 0.5       ],
       [0.33333333, 0.        , 0.        , 0.5       ],
       [0.33333333, 0.5       , 0.        , 0.        ]])
```

In [138…
```python
D=dict.fromkeys(G,1/4)
nlist = list(D.values())
x = np.array(nlist)
for i in range(20):
    y = np.matmul(Tr,x)
    print(y)
    s=np.subtract(x,y)
    print(s)
    if np.all((s)<0.001):
        print(i,' iterations')
        break
    else:
        x=y
```

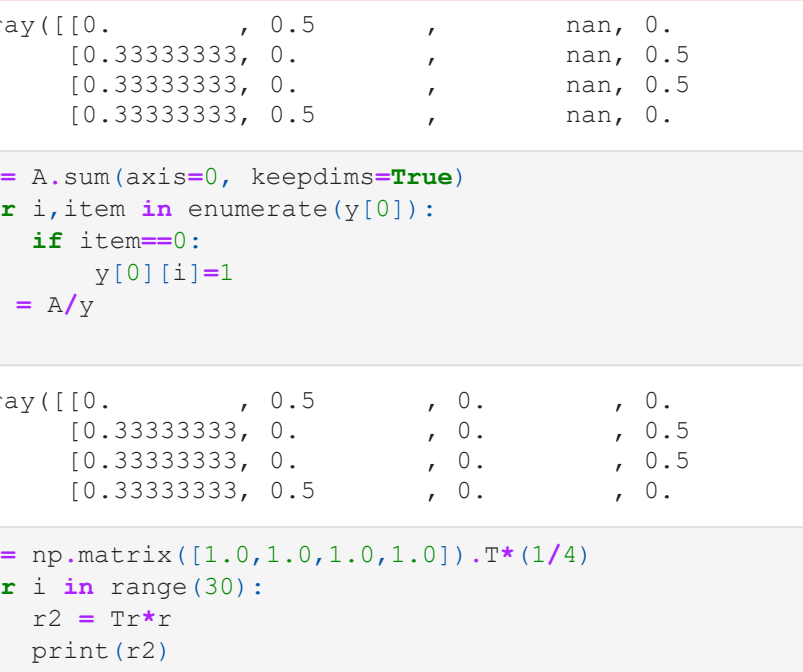
```
[0.375      0.20833333 0.20833333 0.20833333]
[-0.125       0.04166667  0.04166667  0.04166667]
[0.3125     0.22916667 0.22916667 0.22916667]
[ 0.0625     -0.02083333 -0.02083333 -0.02083333]
[0.34375 0.21875 0.21875 0.21875]
[-0.03125     0.01041667  0.01041667  0.01041667]
[0.328125   0.22395833 0.22395833 0.22395833]
[ 0.015625   -0.00520833 -0.00520833 -0.00520833]
[0.3359375  0.22135417 0.22135417 0.22135417]
[-0.0078125   0.00260417  0.00260417  0.00260417]
[0.33203125 0.22265625 0.22265625 0.22265625]
[ 0.00390625 -0.00130208 -0.00130208 -0.00130208]
[0.33398438 0.22200521 0.22200521 0.22200521]
[-0.00195312  0.00065104  0.00065104  0.00065104]
6  iterations
```

In [142…
```python
r = np.matrix([1.0,1.0,1.0,1.0]).T*(1/4)
for i in range(100):
    r2 = Tr*r
    print(r2)
    y = np.subtract(r2,r)
    if np.all(np.abs(y)<0.001):
        print(i,' iterations')
        break
    else:
        r = r2
```

```
[[0.375     ]
 [0.20833333]
 [0.20833333]
 [0.20833333]]
[[0.3125    ]
 [0.22916667]
 [0.22916667]
 [0.22916667]]
[[0.34375]
 [0.21875]
 [0.21875]
 [0.21875]]
[[0.328125  ]
 [0.22395833]
 [0.22395833]
 [0.22395833]]
[[0.3359375 ]
 [0.22135417]
 [0.22135417]
 [0.22135417]]
[[0.33203125]
 [0.22265625]
 [0.22265625]
 [0.22265625]]
[[0.33398438]
 [0.22200521]
 [0.22200521]
 [0.22200521]]
[[0.33300781]
 [0.22233073]
 [0.22233073]
 [0.22233073]]
7  iterations
```

Question2

In [143…
```python
G.remove_edge('C','A')
nx.draw(G, with_labels = True)
```



In [144…
```python
A = nx.adjacency_matrix(G)
A = np.array(A.todense())
A = A.transpose()
A
```

Out[144…
```
array([[0, 1, 0, 0],
       [1, 0, 0, 1],
       [1, 0, 0, 1],
       [1, 1, 0, 0]], dtype=int32)
```

In [145…
```python
Tr = A / A.sum(axis=0, keepdims=True)
Tr
```

```
<ipython-input-145-b8ca877ff8f1>:1: RuntimeWarning: invalid value encountered in true_divide
  Tr = A / A.sum(axis=0, keepdims=True)
```

Out[145…
```
array([[0.        , 0.5       ,        nan, 0.        ],
       [0.33333333, 0.        ,        nan, 0.5       ],
       [0.33333333, 0.        ,        nan, 0.5       ],
       [0.33333333, 0.5       ,        nan, 0.        ]])
```

In [146…
```python
y = A.sum(axis=0, keepdims=True)
for i,item in enumerate(y[0]):
    if item==0:
        y[0][i]=1
Tr = A/y
Tr
```

Out[146…
```
array([[0.        , 0.5       , 0.        , 0.        ],
       [0.33333333, 0.        , 0.        , 0.5       ],
       [0.33333333, 0.        , 0.        , 0.5       ],
       [0.33333333, 0.5       , 0.        , 0.        ]])
```

In [148…
```python
r = np.matrix([1.0,1.0,1.0,1.0]).T*(1/4)
for i in range(30):
    r2 = Tr*r
    print(r2)
    y = np.subtract(r2,r)
    if np.all(np.abs(y)<0.001):
        print(i,' iterations')
        break
    else:
        r = r2
```

```
[[0.125     ]
 [0.20833333]
 [0.20833333]
 [0.20833333]]
[[0.10416667]
 [0.14583333]
 [0.14583333]
 [0.14583333]]
[[0.07291667]
 [0.10763889]
 [0.10763889]
 [0.10763889]]
[[0.05381944]
 [0.078125  ]
 [0.078125  ]
 [0.078125  ]]
[[0.0390625 ]
 [0.05700231]
 [0.05700231]
 [0.05700231]]
[[0.02850116]
 [0.04152199]
 [0.04152199]
 [0.04152199]]
[[0.020761  ]
 [0.03026138]
 [0.03026138]
 [0.03026138]]
[[0.01513069]
 [0.02205102]
 [0.02205102]
 [0.02205102]]
[[0.01102551]
 [0.01606907]
 [0.01606907]
 [0.01606907]]
[[0.00803454]
 [0.01170971]
 [0.01170971]
 [0.01170971]]
[[0.00585485]
 [0.00853303]
 [0.00853303]
 [0.00853303]]
[[0.00426652]
 [0.00621813]
 [0.00621813]
 [0.00621813]]
[[0.00310907]
 [0.00453124]
 [0.00453124]
 [0.00453124]]
[[0.00226562]
 [0.00330198]
 [0.00330198]
 [0.00330198]]
[[0.00165099]
 [0.00240619]
 [0.00240619]
 [0.00240619]]
14  iterations
```

In [117…
```python
# We observe that the steady state probabilities tend to 0 values since the page C is a dead end
# with no output paths from C.
# So, the Random Walk probability to move to the next page is almost 0.
```

Question3

In [149…
```python
G.add_edge('C','C')
```

In [150…
```python
A = nx.adjacency_matrix(G)
A = np.array(A.todense())
A = A.transpose()
A
```

Out[150…
```
array([[0, 1, 0, 0],
       [1, 0, 0, 1],
       [1, 0, 1, 1],
       [1, 1, 0, 0]], dtype=int32)
```

In [151…
```python
Tr = A / A.sum(axis=0, keepdims=True)
Tr
```

Out[151…
```
array([[0.        , 0.5       , 0.        , 0.        ],
       [0.33333333, 0.        , 0.        , 0.5       ],
       [0.33333333, 0.        , 1.        , 0.5       ],
       [0.33333333, 0.5       , 0.        , 0.        ]])
```

In [152…
```python
r = np.matrix([1.0,1.0,1.0,1.0]).T*(1/4)
for i in range(100):
    r2 = Tr*r
    print(r2)
    y = np.subtract(r2,r)
    if np.all(np.abs(y)<0.001):
        print(i,' iterations')
        break
    else:
        r = r2
```

```
[[0.125     ]
 [0.20833333]
 [0.45833333]
 [0.20833333]]
[[0.10416667]
 [0.14583333]
 [0.60416667]
 [0.14583333]]
[[0.07291667]
 [0.10763889]
 [0.71180556]
 [0.10763889]]
[[0.05381944]
 [0.078125  ]
 [0.78993056]
 [0.078125  ]]
[[0.0390625 ]
 [0.05700231]
 [0.84693287]
 [0.05700231]]
[[0.02850116]
 [0.04152199]
 [0.88845486]
 [0.04152199]]
[[0.020761  ]
 [0.03026138]
 [0.91871624]
 [0.03026138]]
[[0.01513069]
 [0.02205102]
 [0.94076726]
 [0.02205102]]
[[0.01102551]
 [0.01606907]
 [0.95683634]
 [0.01606907]]
[[0.00803454]
 [0.01170971]
 [0.96854605]
 [0.01170971]]
[[0.00585485]
 [0.00853303]
 [0.97707908]
 [0.00853303]]
[[0.00426652]
 [0.00621813]
 [0.98329721]
 [0.00621813]]
[[0.00310907]
 [0.00453124]
 [0.98782845]
 [0.00453124]]
[[0.00226562]
 [0.00330198]
 [0.99113043]
 [0.00330198]]
[[0.00165099]
 [0.00240619]
 [0.99353662]
 [0.00240619]]
[[0.0012031 ]
 [0.00175343]
 [0.99529005]
 [0.00175343]]
[[8.76713192e-04]
 [1.27774557e-03]
 [9.96567796e-01]
 [1.27774557e-03]]
[[6.38872786e-04]
 [9.31110517e-04]
 [9.97498906e-01]
 [9.31110517e-04]]
17  iterations
```

In [123…
```python
# We observe that the steady state probabilities tend to 0 values for pages A,B and D while it is almost 1
# for page C.
# Since the page C is a spider trap with the output paths coming back to same page (self loops).
# So, the Random Walk is stuck at C and its probability to move to the other pages is almost 0.
```

Question 4

In [153…
```python
beta = 0.1
r = (1/4)*np.ones((4,1))
e = r

for i in range(20):
    r2 = np.matmul((1-beta)*Tr,r)+beta*e
    print(r2)
    y = np.subtract(r2,r)
    if np.all(np.abs(y)<0.001):
        print(i,' iterations')
        break
    else:
        r = r2
```

```
[[0.1375]
 [0.2125]
 [0.4375]
 [0.2125]]
[[0.120625]
 [0.161875]
 [0.555625]
 [0.161875]]
[[0.09784375]
 [0.13403125]
 [0.63409375]
 [0.13403125]]
[[0.08531406]
 [0.11466719]
 [0.68535156]
 [0.11466719]]
[[0.07660023]
 [0.10219445]
 [0.71901086]
 [0.10219445]]
[[0.0709875 ]
 [0.09396757]
 [0.74107735]
 [0.09396757]]
[[0.06728541]
 [0.08858166]
 [0.75555127]
 [0.08858166]]
[[0.06486175]
 [0.08504737]
 [0.76504351]
 [0.08504737]]
[[0.06327132]
 [0.08272984]
 [0.771269  ]
 [0.08272984]]
[[0.06222843]
 [0.08120982]
 [0.77535193]
 [0.08120982]]
[[0.06154442]
 [0.08021295]
 [0.77802968]
 [0.08021295]]
[[0.06109583]
 [0.07955915]
 [0.77978587]
 [0.07955915]]
[[0.06080162]
 [0.07913037]
 [0.78093765]
 [0.07913037]]
[[0.06060867]
 [0.07884915]
 [0.78169303]
 [0.07884915]]
13  iterations
```

In [71]:
```python
# Teleportation induces the probability of going to the other pages which was
# the concern in above ques 2 and 3.
# So, we observe that using teleportation we have reduced the probability of
# page C from 0.99 to 0.78 and increased the prob of other pages as well.
```

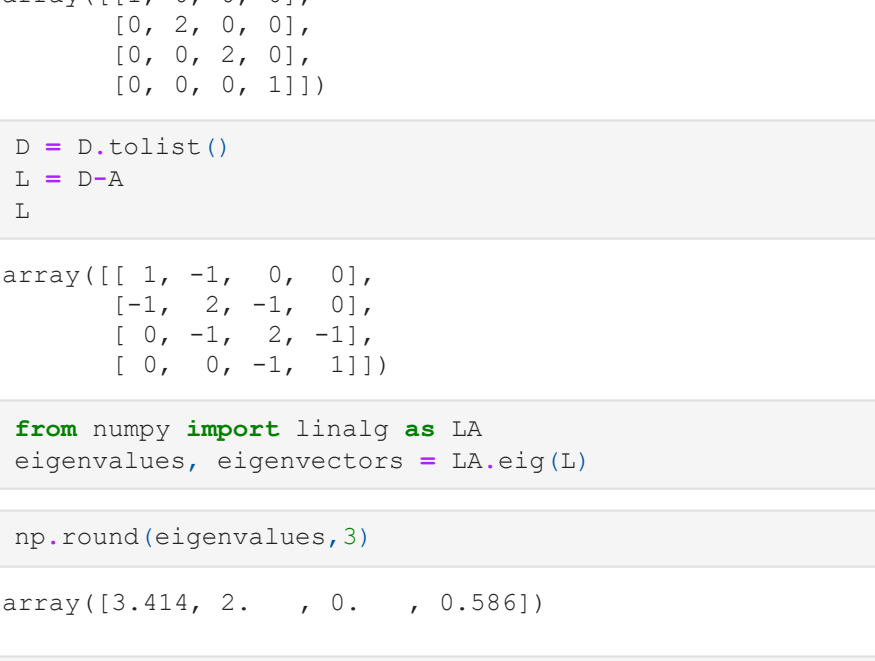
In [154…
```python
# Alternate soln
beta = 0.1
r = np.matrix([1.0,1.0,1.0,1.0]).T*(1/4)

for i in range(20):
    r2 = (1-beta)*Tr*r + np.matrix([1,1,1,1]).T*(1/4)*beta
    print(r2)
    y = np.subtract(r2,r)
    if np.all(np.abs(y)<0.001):
        print(i,' iterations')
        break
    else:
        r = r2
```

```
[[0.1375]
 [0.2125]
 [0.4375]
 [0.2125]]
[[0.120625]
 [0.161875]
 [0.555625]
 [0.161875]]
[[0.09784375]
 [0.13403125]
 [0.63409375]
 [0.13403125]]
[[0.08531406]
 [0.11466719]
 [0.68535156]
 [0.11466719]]
[[0.07660023]
 [0.10219445]
 [0.71901086]
 [0.10219445]]
[[0.0709875 ]
 [0.09396757]
 [0.74107735]
 [0.09396757]]
[[0.06728541]
 [0.08858166]
 [0.75555127]
 [0.08858166]]
[[0.06486175]
 [0.08504737]
 [0.76504351]
 [0.08504737]]
[[0.06327132]
 [0.08272984]
 [0.771269  ]
 [0.08272984]]
[[0.06222843]
 [0.08120982]
 [0.77535193]
 [0.08120982]]
[[0.06154442]
 [0.08021295]
 [0.77802968]
 [0.08021295]]
[[0.06109583]
 [0.07955915]
 [0.77978587]
 [0.07955915]]
[[0.06080162]
 [0.07913037]
 [0.78093765]
 [0.07913037]]
[[0.06060867]
 [0.07884915]
 [0.78169303]
 [0.07884915]]
13  iterations
```

Question5

In [90]:
```python
import networkx as nx
import numpy as np
# To create an empty undirected graph
G1 = nx.Graph()

# To add a node
G1.add_nodes_from('1234')

# To add an edge
G1.add_edges_from([('1','2'),('2','3'),('3','4')])

node_list = G1.nodes()
print("Nodes")
print(node_list)

# To get all the edges of a graph
edge_list = G1.edges()
print("Edges")
print(edge_list)

nx.draw(G1, with_labels = True)
```

```
Nodes
['1', '2', '3', '4']
Edges
[('1', '2'), ('2', '3'), ('3', '4')]
```



In [91]:
```python
A = nx.adjacency_matrix(G1)
A = np.array(A.todense())
A
```

Out[91]:
```
array([[0, 1, 0, 0],
       [1, 0, 1, 0],
       [0, 1, 0, 1],
       [0, 0, 1, 0]], dtype=int32)
```

In [92]:
```python
D = [G1.degree[node] for node in G1.nodes()]
D = np.diag(D)
D
```

Out[92]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 2, 0],
       [0, 0, 0, 1]])
```

In [93]:
```python
D = D.tolist()
L = D-A
L
```

Out[93]:
```
array([[ 1, -1,  0,  0],
       [-1,  2, -1,  0],
       [ 0, -1,  2, -1],
       [ 0,  0, -1,  1]])
```

In [94]:
```python
from numpy import linalg as LA
eigenvalues, eigenvectors = LA.eig(L)
```

In [97]:
```python
np.round(eigenvalues,3)
```

Out[97]:
```
array([3.414, 2.   , 0.   , 0.586])
```

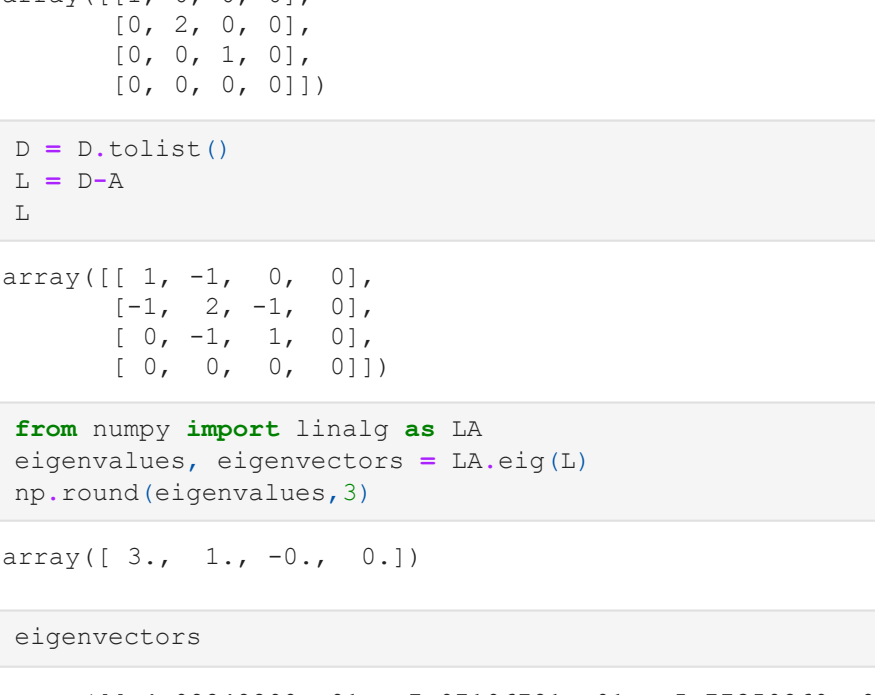
In [96]:
```python
eigenvectors
```

Out[96]:
```
array([[ 0.27059805,  0.5       , -0.5       , -0.65328148],
       [-0.65328148, -0.5       , -0.5       , -0.27059805],
       [ 0.65328148, -0.5       , -0.5       ,  0.27059805],
       [-0.27059805,  0.5       , -0.5       ,  0.65328148]])
```

In [98]:
```python
# We observe that one eigen value is 0 and the corresponding eigenvector has same
# constant value for all nodes
```

In [99]:
```python
import networkx as nx
import numpy as np
# To create an empty undirected graph
G2 = nx.Graph()

# To add a node
G2.add_nodes_from('1234')

# To add an edge
G2.add_edges_from([('1','2'),('2','3')])

node_list = G2.nodes()
print("Nodes")
print(node_list)

# To get all the edges of a graph
edge_list = G2.edges()
print("Edges")
print(edge_list)

nx.draw(G2, with_labels = True)
```

```
Nodes
['1', '2', '3', '4']
Edges
[('1', '2'), ('2', '3')]
```



In [100…
```python
A = nx.adjacency_matrix(G2)
A = np.array(A.todense())
A
```

Out[100…
```
array([[0, 1, 0, 0],
       [1, 0, 1, 0],
       [0, 1, 0, 0],
       [0, 0, 0, 0]], dtype=int32)
```

In [102…
```python
D = [G2.degree[node] for node in G2.nodes()]
D = np.diag(D)
D
```

Out[102…
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 0]])
```

In [103…
```python
D = D.tolist()
L = D-A
L
```

Out[103…
```
array([[ 1, -1,  0,  0],
       [-1,  2, -1,  0],
       [ 0, -1,  1,  0],
       [ 0,  0,  0,  0]])
```

In [104…
```python
from numpy import linalg as LA
eigenvalues, eigenvectors = LA.eig(L)
np.round(eigenvalues,3)
```

Out[104…
```
array([ 3.,  1., -0.,  0.])
```

In [105…
```python
eigenvectors
```

Out[105…
```
array([[-4.08248290e-01, -7.07106781e-01,  5.77350269e-01,
         0.00000000e+00],
       [ 8.16496581e-01,  2.61239546e-16,  5.77350269e-01,
         0.00000000e+00],
       [-4.08248290e-01,  7.07106781e-01,  5.77350269e-01,
         0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         1.00000000e+00]])
```

In [106…
```python
# We observe that 2 eigen values are 0 implying 2 connected components.
# last eigen value which is 0 has eigenvector with same values for nodes 1, 2 and 3
# but different for node 4. This indicates 1,2 and 3 are connected.
```