Table of Contents

CS4423-Networks: Week 6 (19+20 Feb 2025)

Part 3: Eigenvector Centrality - Computation

Niall Madden, School of Mathematical and Statistical Sciences
University of Galway

This Jupyter notebook, and PDF and HTML versions, can be found at https://www.niallmadden.ie/2425-CS4423/#Week06

This notebook was written by Niall Madden, adapted from notebooks by Angela Carnevale.

Modules for this notebook

In [1]:
```
import networkx as nx
import numpy as np
opts = { "with_labels": True,  "node_color": "xkcd:sky blue"} # show labels; nodes are

np.set_printoptions(precision=3)    # just display arrays to 3 decimal places
np.set_printoptions(suppress=True)  # avoid scientific notation (better for matrices)
```

Eigenvector Centrality

So now we know (see `CS4423-W06-Part-2.pdf`)

- The adjacency matrix, A of a connected graph, G, is an irreducible non-negative matrix.
- So the F-B Therom applies to it
- So A has an eigenvalue that is real and positive, and greater than the modulus of any other.
- It has a corresponding positive eigenvector, $\vec{v}$.
- v_i is the Eigenvector Centrality node i.

Normalisation

One minor issue is that any multiple of $\vec{v}$ is also an eigenvector for the same eigenvalue. This is not a major problem: we are mainly interested in if, for example $v_i > v_j$ which would mean that Node i has greater centrality than Node j.

Nonetheless, by convection we choose $\vec{v}$ so that

- $\vec{v} > 0$ (already discussed)
- $\vec{v}^T\vec{v} = 1$ (equivalently, $\|\vec{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \dots v_n^2} = 1$

We say such an eigenvector is **normalised**.

Computing Eigenvalues

Presently, we'll learn about a method called the **Power Method**

For now, though, we'll use the `np.linalg.eig()` which computes the eigenvalues and eigenvectors of a matrix:
`l, V = np.linalg.eig(A)` computes

- `l` : an array of length n containing the eigenvalues of A. (Note: we can't call this array `lambda` , since that is a keyword in Python.
- `V` : a $n \times n$ matrix; column i of V is the eigenvector corresponding to the eigenvalues λ_i.

Example: Find the eigenvalues and corresponding eigenvectors of

$$A = \begin{pmatrix} 2 & 2 \\ 3 & 1 \end{pmatrix}$$

In [2]:
```
A = np.array([[2,2],[3,1]])
l, V = np.linalg.eig(A)
print(f"The eigenvalues of A are {l[0]} and {l[1]}.")
print(f"The corresponding eigenvectors are {V[:,0]} and {V[:,1]}")
```

```
The eigenvalues of A are 4.0 and -1.0.
The corresponding eigenvectors are [0.707 0.707] and [-0.555  0.832]
```

Let's check if this worked:

In [3]:
```
print(A@V[:,0])
print(l[0]*V[:,0])
```

```
[2.828 2.828]
[2.828 2.828]
```

Let's check if the columns of `V` are normalised:

In [4]:
```
print(f" ||v|| = {np.linalg.norm(V[:,1])}")
```

```
 ||v|| = 1.0
```

Eigenvalues of adjacency matrices

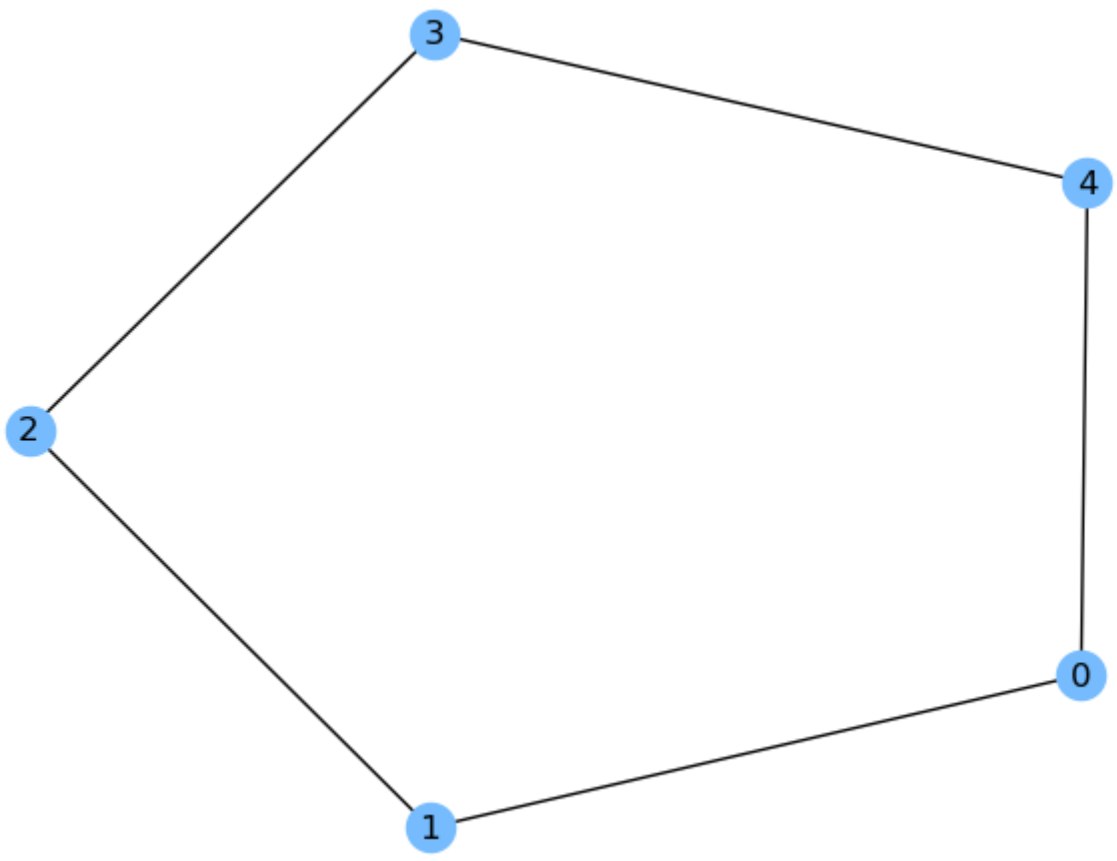
Let's look at some examples, for which we may have an intuition for the centrality. we want to check that

- there is a dominant positive eigenvalue
- there is a corresponding positive eigenvector
- the centrality values seem "sensible"

(Note: *Extra details given on the white board!*)

Example 1: $G = C_5$

```
In [5]: G = nx.cycle_graph(5)
        nx.draw(G, **opts)
```

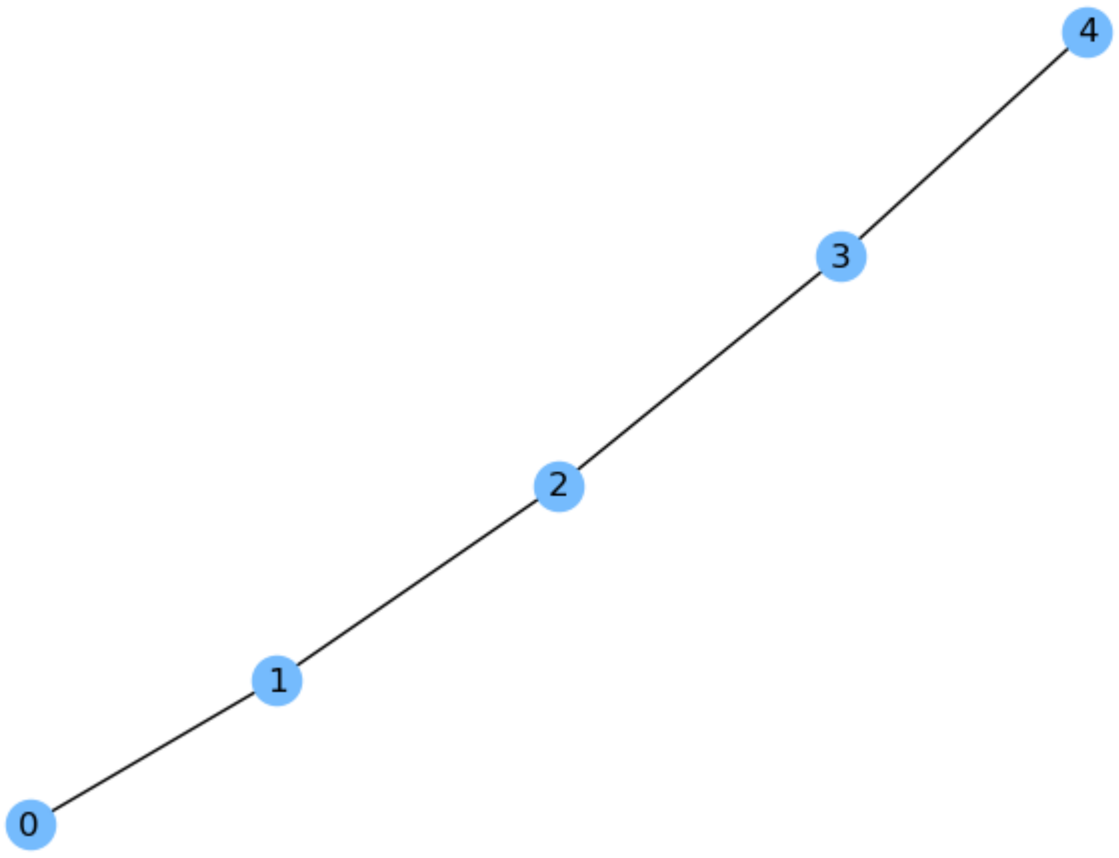


```
In [6]: A = nx.adjacency_matrix(G).toarray()
        l, V = np.linalg.eig(A)
        for i in range(G.order()):
            print(f"eigenvalue {l[i]:6.3f} has eigenvectors are {V[:,i]}")
```

```
eigenvalue -1.618 has eigenvectors are [ 0.632 -0.512  0.195  0.195 -0.512]
eigenvalue  0.618 has eigenvectors are [-0.632 -0.195  0.512  0.512 -0.195]
eigenvalue  2.000 has eigenvectors are [-0.447 -0.447 -0.447 -0.447 -0.447]
eigenvalue -1.618 has eigenvectors are [-0.032  0.397 -0.611  0.591 -0.345]
eigenvalue  0.618 has eigenvectors are [ 0.074  0.62   0.309 -0.429 -0.575]
```

Example 1: $G = P_5$

```
In [7]: G = nx.path_graph(5)
        nx.draw(G, **opts)
```



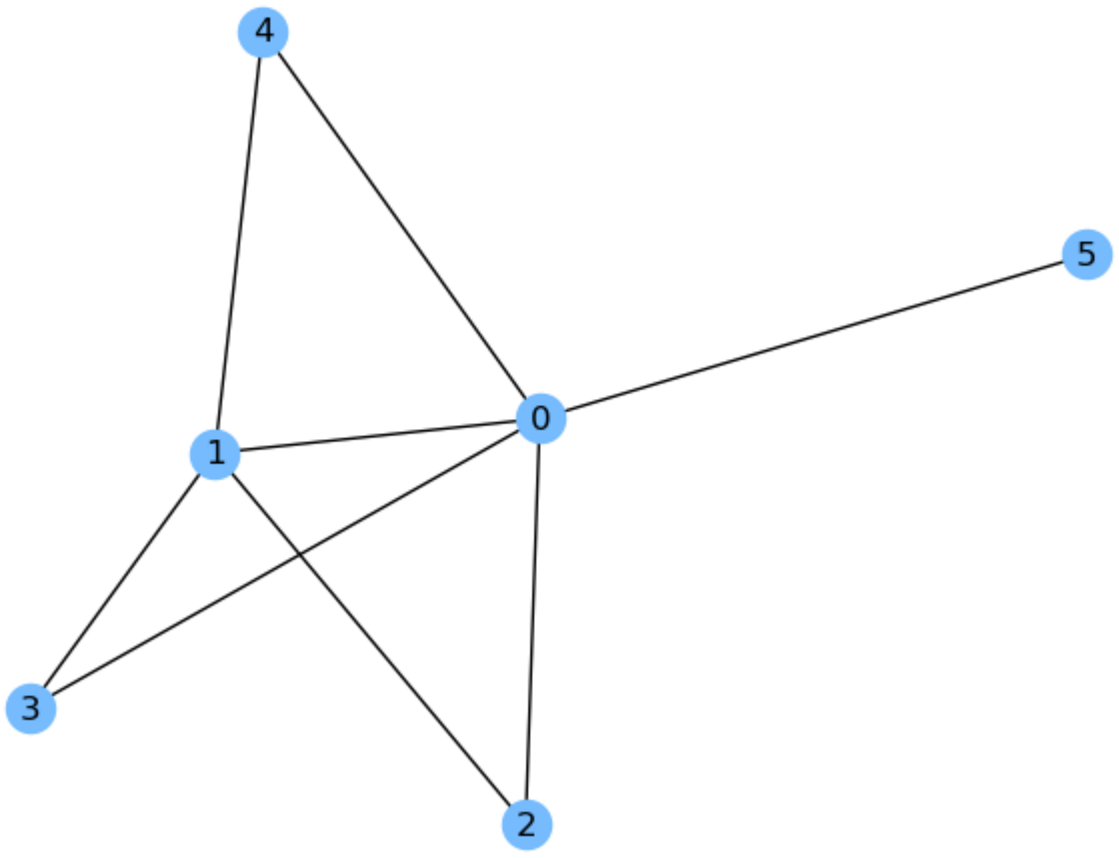
In [8]:
```
A = nx.adjacency_matrix(G).toarray()
l, V = np.linalg.eig(A)
for i in range(G.order()):
    print(f"eigenvalue {l[i]:6.3f} has eigenvectors are {V[:,i]}")
```

```
eigenvalue  1.732 has eigenvectors are [0.289 0.5   0.577 0.5   0.289]
eigenvalue -1.732 has eigenvectors are [-0.289  0.5   -0.577  0.5   -0.289]
eigenvalue -1.000 has eigenvectors are [-0.5  0.5 -0.   -0.5  0.5]
eigenvalue -0.000 has eigenvectors are [ 0.577 -0.    -0.577  0.     0.577]
eigenvalue  1.000 has eigenvectors are [-0.5 -0.5 -0.   0.5  0.5]
```

Example 3

Let's look at the eigenvalues of an adjacency matrix of a graph. It is constructed so that node 0 is more "central" than any of the others, node 5 is the least "central".

In [9]:
```
G = nx.Graph(["01", "02", "03", "04", "05", "12", "13", "14"])
nx.draw(G, **opts)
```



In [10]:
```
A  = nx.adjacency_matrix(G).toarray()
print(A)
```

```
[[0 1 1 1 1 1]
 [1 0 1 1 1 0]
 [1 1 0 0 0 0]
 [1 1 0 0 0 0]
 [1 1 0 0 0 0]
 [1 0 0 0 0 0]]
```

In [11]:
```
l, V = np.linalg.eig(A)
print(f"The eigenpairs of A are:")
for i in range(5):
    print(f"Eigenvalue {l[i]:8.3f} with eigenvector {V[:,i]}")
```

```
The eigenpairs of A are:
Eigenvalue    3.102 with eigenvector [0.568 0.523 0.352 0.352 0.352 0.183]
Eigenvalue    0.344 with eigenvector [ 0.296 -0.343 -0.138 -0.138 -0.138  0.859]
Eigenvalue   -2.123 with eigenvector [-0.557 -0.324  0.415  0.415  0.415  0.262]
Eigenvalue   -1.323 with eigenvector [ 0.529 -0.71   0.137  0.137  0.137 -0.4  ]
Eigenvalue   -0.000 with eigenvector [-0.    -0.    -0.509  0.807 -0.298 -0.   ]
```

The Power Method

There are subfields in the *Numerical Linerar Algebra* dedicated to computing estimates for eigenvalues and eigenvectors. When we only need one eigenvalue, and it is the largest, use the **Power method**:

1. start with any $u = (1, 1, \dots, 1)$, say;

2. keep replacing $u \leftarrow Au$ until $u/\|u\|$ becomes stable ...

Questions Does this work? Meaning:

- Does the sequence actually converge?
- Does it return the correct values?

```
In [12]: n = G.order()
         u = np.ones((n,1)); u=u/np.linalg.norm(u)
         for i in range(10):
             u = A @ u
             u = u/np.linalg.norm(u)
```

```
In [13]: print(u)
```

```
[[0.564]
 [0.521]
 [0.354]
 [0.354]
 [0.354]
 [0.185]]
```

```
In [14]: print(V)
```

```
[[ 0.568  0.296 -0.557  0.529 -0.     0.   ]
 [ 0.523 -0.343 -0.324 -0.71  -0.     0.   ]
 [ 0.352 -0.138  0.415  0.137 -0.509  0.816]
 [ 0.352 -0.138  0.415  0.137  0.807 -0.408]
 [ 0.352 -0.138  0.415  0.137 -0.298 -0.408]
 [ 0.183  0.859  0.262 -0.4   -0.     0.   ]]
```

Finished here **Thursday**