

ECS253 - Homework 1

Name: Camille Scott
Email: camille.scott.w@gmail.com

May 3, 2016

This is the common problem set for Homework 1 from the spring quarter Network Theory class at UC Davis taught by Prof. Raissa D'Souza. The original assignment is at <http://mae.engr.ucdavis.edu/dsouza/Courses/253-S16/hw1.pdf>. Source code for this notebook is on github at <https://github.com/camillescott/ucd-ecs253>.

```
In [1]: %matplotlib inline
import numpy as np
import networkx as nx
import seaborn as sns
sns.set_style('ticks')
sns.set_context('poster')
```

1 Adjacency Matrices

1.1 The Matrix

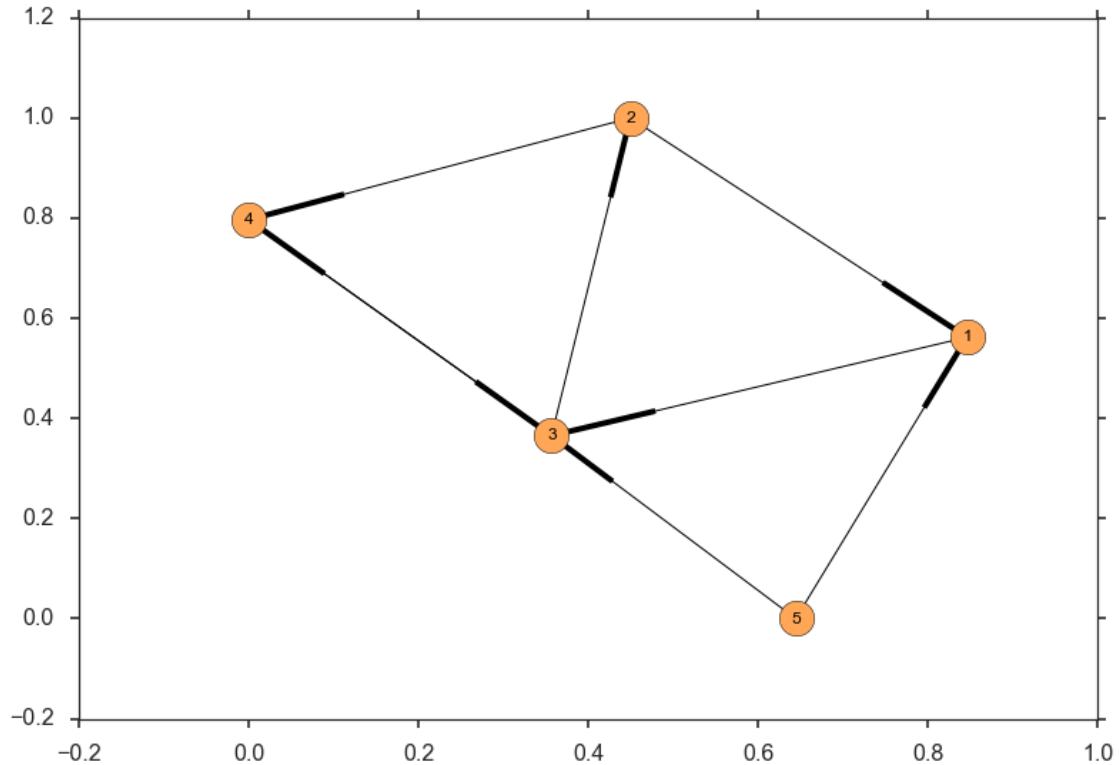
```
In [2]: A_directed = np.array( [[0, 1, 0, 0, 1],
                                [0, 0, 1, 0, 0],
                                [1, 0, 0, 1, 1],
                                [0, 1, 1, 0, 0],
                                [0, 0, 0, 0, 0]] )

print(A_directed)
```

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

A little visualization, just to double check.

```
In [3]: nx.draw_networkx(nx.DiGraph(data=A_directed.T),
                          labels=dict(zip(range(len(A_directed)), range(1, len(A_directed)+1))),
                          node_size=600,
                          node_color=sns.xkcd_rgb["pale orange"])
```



1.2 Steady-State Probability of Random Walker

We can calculate the steady state probabilities quite simply by dividing out the column sums of the adjacency matrix A to convert it into a state transition matrix M and then computing the matrix power M^i . With i as a reasonably high number, the result will converge to the steady state probabilities.

```
In [4]: def calc_steady_state(A, i=100):
        M = A / A.sum(axis=0)
        M = np.linalg.matrix_power(M, i)
        return M

In [5]: def print_probs(probs):
        print('\n'.join('Node {0}: {1:.4f}'.format(node, p) for node, p in \
            zip(range(1, len(probs)+1), probs)))
```

The resulting probabilities:

```
In [6]: probs = calc_steady_state(A_directed)[: ,0]
        print_probs(probs)
```

Node 1: 0.1000

Node 2: 0.2000

```
Node 3: 0.4000
Node 4: 0.3000
Node 5: 0.0000
```

1.3 Steady-State Probabilities for Undirected Graph

For the undirected variant, we just mirror across the diagonal.

```
In [7]: A_undirected = np.array( [[0, 1, 1, 0, 1],
                                   [1, 0, 1, 1, 0],
                                   [1, 1, 0, 1, 1],
                                   [0, 1, 1, 0, 0],
                                   [1, 0, 1, 0, 0]] )

print(A_undirected)

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

```
In [8]: probs = calc_steady_state(A_undirected)[:,0]
print_probs(probs)
```

```
Node 1: 0.2143
Node 2: 0.2143
Node 3: 0.2857
Node 4: 0.1429
Node 5: 0.1429
```

2 Rate equations: Network growth with uniform attachment

2.1 Rate Equations

$$k > m: n_{k,t+1} = n_{k,t} + \frac{1}{t}n_{k-1,t} - \frac{1}{t}n_{k,t}$$

$$k = m: n_{m,t+1} = n_{m,t} + 1 - \frac{1}{t}n_{m,t}$$

2.2 Probabilistic Treatment

$$k > m: (t+1)P_{k,t+1} = tP_{k,t} + \frac{1}{t}tP_{k-1,t} - \frac{1}{t}tP_{k,t}$$

$$\Rightarrow (t+1)P_{k,t+1} = (t-1)P_{k,t} + P_{k-1,t}$$

$$k = m: (t+1)P_{m,t+1} = tP_{m,t} + 1 - \frac{1}{t}tP_{m,t}$$

$$\Rightarrow (t+1)P_{m,t+1} = (t-1)P_{m,t} + 1$$

2.3 Solving for P_k

$$\begin{aligned}k > m: tP_k + P_k &= tP_k - P_k + P_k - 1 \\ \Rightarrow P_k &= \frac{P_{k-1}}{2} \\ k = m: tP_m + P_m &= tP_m - P_m + 1 \\ \Rightarrow P_m &= \frac{1}{2}\end{aligned}$$

2.4 Final Expression

$$\begin{aligned}P_k &= \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \dots \times P_m \\ \Rightarrow P_k &= \frac{1}{2^k}\end{aligned}$$