# Exercise 9: Networks 1

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

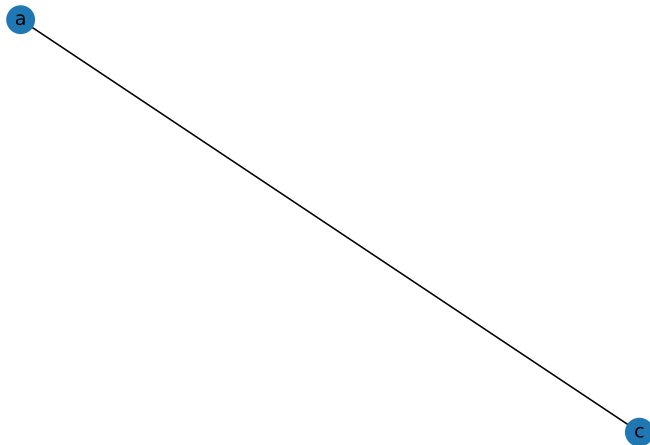April 6, 2020

## Problem 9.1.1. - making a `nx` graph



**Fig 1.** A very simple two-node network.

The first step of creating our first graph will be to create an instance of nx.Graph. Because we use nx.Graph and not nx.DiGraph our graph will be *undirected*.

```
g = nx.Graph()
```

I will add three nodes to the graph, 'a', 'a', and 'c'. You can do this one at a time with g.add_node, or all at once as i do here

```
g.add_nodes_from(['a', 'b', 'c'])
```

Notice that there is no reassignment here, i.e. we don't write g = g.add_nodes_from(...). To understand why, consider a dummy-class that mimicks the functionality of nx.Graph, it might look something like

```python
class DummyGraph:
    def __init__(self):
        self.nodes = []
        self.edges = []

    def add_nodes_from(self, nodes):
        for node in nodes:
            self.nodes.append(node)
```

Notice that DummyGraph.add_nodes_from does not return anything (and thus defaults to returning **None**). If we tried to write g = g.add_nodes_from(...), g would be **None**. However when add_nodes_from is called it directly modifies the instance self. Many modules implement their classes differently. For instance in Pandas, the reassignment pattern is fine and we use it all the time. Which one to use depends on the package you are using so you will have to get used to both.

With this detour out of the way, let us add some edges and remove the 'b' node.

1

```
g.add_edges_from([('a', 'b'), ('a', 'c')])
g.remove_node('b')
```

Finally we can draw the network; all of the networkx plots are made with matplotlib so you can use all of the matplotlib code you know to modify the figure.

```
nx.draw(g, with_labels = True)
plt.savefig('writeup/first_network.pdf')
```
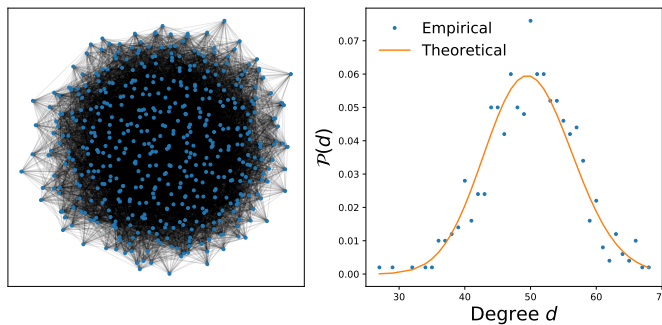
## Problem 9.1.2 - the Erdős-Rényi graph

**Fig 2.** Erdős-Rényi network with $n = 500$ and $p = 0.1$.

The ER graph $G_{n,p}$ is a random graph constructed by fixing $n$ and then independently for each pair of nodes forming a connection with probability $p$. Notice that even for $p$ close to 1 there is still a non-zero probability that the network is very loosely connected. Let us begin by creating an ER-network. I will give it 500 nodes instead of the 100 mentioned in the exercise to reduce the role of randomness a little.

```
ER = nx.gnp_random_graph(n=500, p=0.1)
```

Now i want both the empirical degree distribution, which is simply the percentage of nodes with a given degree $k$, calculated for all relevant values of $k$, and the theoretical degree distribution given by

$$p(k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}. \tag{1}$$

To compute the empirical distribution I will need to count the number of occurences of each degree $k$ in the network, and for the theoretical distribution I need a function that computes $n$ choose $k$, so i will import a counter and a function that computes $\binom{n}{k}$,

```
from collections import Counter
from scipy.special import comb
```

with these two tools i am ready to write a function that computes all of the things I need for my figure. It will need a network as input, and the value of $p$, since this enters in the equation above, and is not stored in the network object.

```
def compute_degree_distribution(network, p):
    N = er.number_of_nodes()

    # Compute the empirical degree distribution
    degrees = sorted([d for n, d in network.degree()])  # Degree of each node
    counts = Counter(degrees)  # Count the number of times each value of d appears
    degrees, counts = zip(*counts.items())  # extract into two lists
    empirical_probs = [c/N for c in counts]  # convert from counts to probabilities.
```

```
# Compute the theoretical degree distribution
theoretical_probs = [comb(N - 1, k) * (p**k) * (1-p)**(N - 1 - k) for k in degrees]

return degrees, empirical_probs, theoretical_probs
```

With this function we can compute the degree distribution of the ER network we created above.

```
degr, probs, ther = compute_degree_distribution(ER, 0.1)
```

Now we have all the ingredients we need to create the figure. I first set up a two-panel figure and plot the network on the first panel. I use some of the more detailed drawing methods to control the size and position of the nodes and edges.

```
fig, ax = plt.subplots(1,2, figsize = (10,5))
pos = nx.spring_layout(er)
nx.draw_networkx_nodes(er, pos, node_size = 10, with_labels=False, ax = ax[0])
nx.draw_networkx_edges(er, pos, alpha = 0.1, width=0.1, ax = ax[0])
```

Then on the second panel we can draw the empirical and theoretical degree distribution. I plot the empirical distribution as dots to reduce clutter in the figure.

```
ax[1].plot(degr, probs, '.', label = 'Empirical')
ax[1].plot(degr, ther, label = 'Theoretical')
ax[1].set_xlabel('Degree $d$', fontsize = 20)
ax[1].set_ylabel('$\mathcal{P}(d)$', fontsize = 20)
ax[1].legend(frameon=False, fontsize = 16)
```