Exercise set #2 (31 pts)

- The deadline for handing in your solutions is Nov 21th 2016 23:55.
- Return your solutions (one `.pdf` file and one `.zip` file containing Python code) in MyCourses (Assignments tab). Additionally, submit your pdf file also to the Turnitin plagiarism checker in MyCourses.
- Check also the course practicalities page in MyCourses for more details on writing your report.

1. Properties of Erdős-Rényi networks (9 pts)

Erdős-Rényi networks are random networks where N nodes are randomly connected such that the probability that a pair of nodes is linked is p. In network science, the E-R random graphs are important because they provide the simplest reference to which one can compare real-world networks. In this exercise, we will analyze some of the properties of E-R graphs.

a) (1 pt) The degree distribution $P(k)$ of E-R networks is binomial:

$$P(k) = \binom{(N-1)}{k} p^k (1-p)^{(N-1)-k}, \tag{1}$$

where N equals the number of nodes in the network. **Explain in detail** the origin of each of the three factors in the above formula.

b) (1 pt) **Motivate** why in E-R networks, the average clustering coefficient C equals p (on expectation).

Hint:

- What is the expected value of the clustering coefficient for one node?

c) (1 pt) **Explain**, what happens to C, if $N \to \infty$ with $\langle k \rangle$ bounded.

Hint:

- Mathematically, if x is 'bounded' then x is always smaller than some possibly big, but finite, number M.

d) (3 pts) Use NetworkX to (loosely) verify that the so-called percolation threshold of E-R networks is at an average degree of $\langle k \rangle = 1$. That is, for $\langle k \rangle < 1$ the largest connected component is small (size being measured as number of participating nodes), and for $\langle k \rangle > 1$ it quickly reaches the network size.

Do this by generating E-R networks of size $N = 10^4$ with different average degrees: $\langle k \rangle = [0.00, 0.05, ..., 2.45, 2.50]$). For each of the E-R networks, **compute** the size of the *largest* and the *second largest* connected component and **plot** them against $\langle k \rangle$.

Hints:

- Use function `nx.fast_gnp_random_graph(N, p)` for generating E-R networks. (The standard function `nx.erdos_renyi_graph(N, p)` is slow with large graphs.)

- Test your code first with small values of N.
- For looping through a range of non-integer values, you may find `numpy.arange(first_value, last_value, stepsize)` useful.
- `coms=list(nx.connected_components(G))` gives a list containing all connected components of G (as lists of nodes).
- You can find out the largest component size by building from the above `coms` a list of component sizes `sizes` measured as numbers of nodes (loop through the elements of `coms` and apply `len()` on them).
- Then, sort the list of the sizes, and find out the maximum and second largest element of the list.

e) (3 pts) Use NetworkX to calculate estimates for the ensemble averages $\langle k \rangle$, $\langle C \rangle$, and $\langle d^* \rangle$ defined in the Exercise 6 of the Exercise set #1. Do this by generating 100 E-R networks for each value of p in range $p = [0.00, 0.05, ..., 0.95, 1]$ and $N = 3$. An estimate for the ensemble average $\langle X \rangle$ can be calculated for each value of p by calculating the average value of X over the 100 realisations. **Plot** each quantity in a single plot such that p values are on the x-axis and $\langle X \rangle$ values are on the y-axis. If you solved the part b of Exercise 6 of the Exercise set #1 you can check the correctness of your results by including your analytical solution to these plots. Repeat the same exercise with $N = 100$. **Explain** the benefits and downsides of this method as compared to the analytical method of Exercise 6?

Hints:

- You can use the function `nx.diameter` to calculate the diameter of a network. However, this function only accepts graphs that have a single connected component. Use the function `nx.connected_component_subgraphs` to iterate over the components, calculate the diameter for each of them, and find the largest value d^*.

2. Implementing the Watts-Strogatz small-world model (7 pts)

In this exercise, you will implement the Watts-Strogatz small-world model [1], which is a very simple network model that yields small diameter as well as high level of clustering. In practice, the Watts-Strogatz model is a ring lattice where some of the links have been randomly rewired. The model has three parameters: network size N, m (a node on the ring connects to m nearest neighbors both to the left and to the right), and p, the probability of rewiring one end of each link to a random endpoint node.

a) (4 pts) **Implement** the Watts-Strogatz small world model using $N = 15$, $m = 2$, and **visualize** the network with $p = 0.0$ and $p = 0.5$ using a circular layout algorithm (`nx.draw_circular(G)`), and check that the networks look right.

Hints:

- Begin with generating an empty network with N nodes: `G=nx.Graph()`, then add nodes with `G.add_nodes_from(range(0,N))` (the command `range(0,N)` gives you a list `[0, 1, ..., N-1]`).
- Then, connect each node to its m neighbours to the left and right (*i.e.* join node i with nodes $i-m, i-m+1, \ldots, i+1, \ldots, i+m$). Here the remainder-operation might come handy, see e.g. what `np.remainder(1030, 1000)` does.

- Check that your ring lattice looks ok with `nx.draw_circular(G)`.
- Then, for rewiring the network's links, the easiest way is to first generate a separate list of all links (`edges=G.edges()`) and then go through this list one by one (`for edge in edges:`), choosing to rewire if a random number (e.g. `numpy.random.rand()`) is less than p. When rewiring, remove the original link from `G` and create a new one where one of the endpoints has been replaced by a randomly chosen vertex in the network.

  Note: NetworkX has a ready-made function for small-world networks but here the task is to program your own function, so do not use it (except for checking results, if in doubt).

b) (3 pts) **Plot** the *relative* average clustering coefficient $C(p)/C(p=0)$ and average shortest path length $\langle l(p)\rangle/\langle l(p=0)\rangle$ vs. p in your network, for $p = 0.001, 0.002, 0.004, \ldots, 0.512$ (relative=average value for given p divided by the same value for $p = 0$). **Use** $N = 1000$ and $m = 5$. Use a logarithmic x-axis in your plot (`ax.semilogx`). **Check** that your results are in line with the plots in the lecture slides.

Hints:

- With large value of p the network may splits into different parts. However, with $N = 1000$ this should not happen often. In case you face such a problem, run your code again so that the rewiring step results in a connected graph.
- Use `nx.average_clustering(G)` and `nx.average_shortest_path_length(G)`. First, initialize empty lists for clustering, path length, and p: `c_list=[]`, `l_list=[]`, etc.
- To loop over the range of p-values, use *e.g.*

```
p=0.001
while p<0.6:
    ...
    p=2.0*p
```

- For each value, calculate clustering and path length, and append their values and the value of p to the lists discussed above. Then, just plot the clustering/path length lists against the p-list.

3. Implementing the Barabási-Albert (BA) model (7 pts)

The Barabási-Albert scale-free network model is a model of network growth, where new nodes continuously enter the network and make links to existing nodes with a probability that is linearly proportional to their degree. The steps required for generating a Barabási-Albert scale-free network with N nodes are as follows:

- Create a small seed network which has m nodes, where m is the number of links a new node creates to already existing nodes.
- Add new nodes to the network until your network has N nodes, such that each entering node has m links and connects to existing nodes proportional to their degrees.

In this exercise, we will implement the model and investigate the networks it generates.

a) (3 pts) **Implement** a Python function for generating Barabási-Albert networks. Then **generate** a network with $N = 200$ and $m = 1$ (a tree!) and **visualize** it with e.g. `nx.draw_spring(G)`. You should be able to spot some nodes that have many connections, while most of the nodes have few connections.

Hints:

- The seed network can be anything, *e.g.* a clique of size $m+1$, as long as it has enough nodes for the first incoming node to attach to with m links.
- The easiest way of picking nodes with probability proportional to their degree is to maintain a list of node labels where each node appears as many times as its degree is, and then just pick a random element from the list. E.g. if node 1 has degree 3 and node 2 degree 1, the list would look like `degree_list=[1,1,1,2]`. You can pick a random element from a list e.g. with `target=np.random.choice(degree_list)`.

b) (3 pts) **Generate** a new network using parameters $N = 10^4$ with $m = 2$ and **plot** the logarithmically binned degree distribution $P(k)$ (on double logarithmic axes, `ax.loglog`). **Compare** your result with the theoretical prediction of $P(k) = 2m\,(m + 1)\,/\,[k\,(k + 1)\,(k + 2)]$ (shown in the next exercise). To this end, **plot** both the experimental and theoretical distributions to the same axes.

Hints:

- You can get a list of the degrees of a network `G` with `degrees=nx.degree(G).values()`.
- For plotting the binned degree distribution, please have a look at the materials for the binning tutorial in MyCourses.
- There is no simple rule of thumb for selecting the number of bins. However, ideally there should be no empty bins, but on the other hand one would like to have as many bins as possible to best present the shape of the true distribution.
- The generation of results should take only a few seconds. If your code takes too long time to run, there should ways to improve its efficiency.
- For the interested: When plotting the node degree distribution, you may end up with empty bins with small values of k. (Consider e.g. if you had a bin [3.1, 3.9]: this bin would always have value zero.) To circumvent this, it is often practical to bin degree-distributions using "lin-log" bins: [0.5, 1.5, 2.5, ..., 9.5, 10.5, 12.5, 16.5, ...] so that one does not end up with empty bins with small values of k. **If you wish**, you may bin your data using logarithmic bins or this more sophisticated approach of "lin-log" bins.

c) (1 pt) By reading from the plot of the experimental degree distribution, **estimate** the probability for a randomly picked node to have a degree value between 10 and 20. **Explain** how you obtained your estimate.

4. Deriving the degree distribution for the BA-model (8 pts, pen and paper)

In this exercise, we show that the degree distribution of the Barabási-Albert scale-free model is $P(k) = 2m\,(m + 1)\,/\,[k\,(k + 1)\,(k + 2)]$ in the limit of infinite size, *i.e.* it becomes a power law only for very large values of k.

As a reminder, the BA scale-free network growth algorithm goes as follows:

1. Start the network growth from a small "seed" network of N_0 fully connected vertices.
2. Pick m different vertices from the existing network so that the probability of picking vertex v_i of degree k_i equals $p(k_i) = \frac{k_i}{\sum_j k_j}$, i.e. the degree of that vertex divided by the sum of the degrees of all vertices.
3. Create a new vertex and connect it to the m vertices which were chosen above.
4. Repeat steps 2.-3. until the network has grown to the desired size of $N_{final} = N_0 + I$ vertices, where I denotes the number of iterations.

The exact degree distribution for the Barabási-Albert model in the limit of infinite network size[1] can be derived using the so-called *master equation* approach (see, *e.g.* [2]). This approach makes use of the fact that the BA model is a model of network growth, *i.e.* the network is continuously expanding. The key idea of the master equation approach is to write an equation for the changes in the fraction of vertices of degree k, p_k, as function of time and find stationary solutions. In such solutions, p_k does not change anymore when the network grows, corresponding to an infinite network size. This stationary solution for p_k when $N \to \infty$ equals the degree distribution $P(k)$ of the network.

In this exercise, you will derive the distribution step-by-step.

Throughout this exercise, show all intermediate steps and motivate your reasoning either mathematically or verbally.

a) (2 pts) Let $p_{k,N}$ be the density of vertices of degree k in a network that, at time t, has altogether $N(t)$ vertices. Thus, $n_{k,N} = N(t)p_{k,N}$ is the number of vertices of degree k in the network. At each time step, one vertex is added, and hence $N(t) = t + N_0$, where $t \in \mathbb{Z}$ denotes the time step of the network growth process. Since N_0 is small, we can approximate $N \approx t$. In the following, N will be used for $N(t)$ for readability.

   In the BA model, the probability Π_i that a new edge attaches to a *particular* vertex of degree k_i equals

$$\Pi_i = \frac{k_i}{\sum_{j=1}^{N} k_j}. \tag{2}$$

   As our first intermediate result, we will need the probability $\Pi(k)$ that a new edge attaches to *any* vertex of degree k in a network of N vertices. This equation reads

$$\Pi(k) = \frac{k p_{k,N}}{2m}. \tag{3}$$

   Your first task is to **derive** (3) from (2).

   Hint: Formulate the sum of degrees $\sum_j^N k_j$ in terms of m and N (How much the total degree grows when a new vertex is added? You can approximate $N_0 \approx 0$.) and note that there are $Np_{k,N}$ vertices of degree k.

b) (2 pts) Next, we will construct the master equations for the changes of the *average* numbers of vertices of degree k. From Eq. (3), the average number[2] of vertices of degree k that

[1]Some words of explanation might be helpful here. This approach is typical of statistical physics - very large systems are usually well approximated by results derived for infinite systems. This also applies in the case of complex networks. The exact degree distribution can (and has been) derived for finite-sized networks as well, but the calculations are extremely cumbersome.

[2]This is the so-called *mean-field* approach. Instead of keeping track on what happens to each vertex, we will focus on what *on the average* happens to vertices of some degree k.

gain an edge when a single new vertex with m edges is added is $m \times kp_{k,N}/2m = \frac{1}{2}kp_{k,N}$. This means that the number $n_{k,N}$ of vertices with degree k must *decrease* by this amount, since these vertices become vertices of degree $k+1$. Let's mark this as

$$n_k^- = \frac{1}{2}kp_{k,N}. \tag{4}$$

But at the same time, there is an increase n_k^+ as well. For vertices with degree $k > m$ this is equal to the average number of vertices that used to have degree $k-1$ and became vertices of degree k by gaining an edge. For vertices with $k = m$, $n_k^+ = 1$. **Explain why?**

Now, we're ready for the master equation! With the help of the above results, **write down** equations for the *net change* of the number of vertices of degree k as the network grows in size from N to $N+1$,

$$\begin{aligned}(N+1)p_{k,N+1} - Np_{k,N} &= n_k^+ - n_k^-. \\ &= ?\end{aligned} \tag{5}$$

Write separate equations for both cases ($k > m$, $k = m$). In the case $k = m$, denote the densities by $p_{m,N+1}$ and $p_{m,N}$ as this will make things easier in what follows. (Note that the left-hand side in the above equation is simply due to $n_{k,N+1} = (N+1)p_{k,N+1}$ and $n_{k,N} = Np_{k,N}$.)

c) (2 pts) Now, let the network grow towards the infinite network size limit and consider stationary solutions of the two equations you just wrote. In this case, there are no longer changes in the probability density p_k, and so you can write $p_{k,N+1} = p_{k,N} = p_k$, and $p_{k-1,N} = p_{k-1}$, and $p_{m,N+1} = p_{m,N} = p_m$.

Write down equations for p_k and p_m. The density p_k should now be of the form $F(k) \times p_{k-1}$, where $F(k)$ is some prefactor depending on k alone. p_m should be a function of m only, $p_m = G(m)$.

d) (2pts) We're almost there! Now things get recursive: p_k depends on p_{k-1}. At the same time, we have a formula for p_m which depends only on m, which is the smallest degree in our network. Your final task is to **derive** a formula for p_k, so that p_k is a function of k and m only.

Hint: First, write a formula for p_{m+1} using the formulas $p_k = F(k) \times p_{k-1}$ and $p_m = G(m)$, then to write a formula for p_{m+2} etc.. Continue until you see which terms cancel out.

Feedback (1 pt)

To earn one bonus point, give feedback on this exercise set and the corresponding lecture latest two day after the report's submission deadline.
Link to the feedback form: `https://goo.gl/forms/mWAqj05ANk73IbrJ2`.

References

[1] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[2] M. E. J. Newman, "The Structure and Function of Complex Networks," *Siam Review*, vol. 45, pp. 167–256, 2002.