Graphs

Author(s): Rick Quax

- The first part are all analytical questions.
- Always show the steps which you made to arrive at your solution.
- Make sure you answer all parts of each question.
- Always label your axes in each figure. Your figures should all be plotted (using code) and not hand-drawn.
- You must submit your answers to these questions as a PDF on Canvas. Do not compress the files, but hand them in separately. Try to write concisely and limit your submission to 10 pages.
- You must also submit all your code needed for this assignment (and *only* for this assignment). This code should be Python and when run it should successfully generate[1] and output all figures that you use in your PDF, either to screen or written to suitably named files in the same directory.

Grading

The points for each part of a question are shown alongside every step; there are 151 points in total.

Note that this means that your code is not directly graded. It is only used to verify that your code is functional and is not copied from someone else. However, if a (part of) your code is not functional, i.e., does not produce a given figure that you use in an answer, or it's not structured/commented well enough for us to understand it, then you risk not receiving any points for the corresponding answer.

Background material

This exercise sheet requires knowledge of the following skills.

- Basic graph theory.
- Binomial distribution.
- Proof by induction.

You will also need to know about centralities. The different types of centralities are briefly explained in this slide deck: `https://www.cl.cam.ac.uk/teaching/1213/L109/stna-lecture3.pdf`, but feel free to search for other sources if you feel like it.

[1] It is fine if this takes a while to run, due to the simulations.

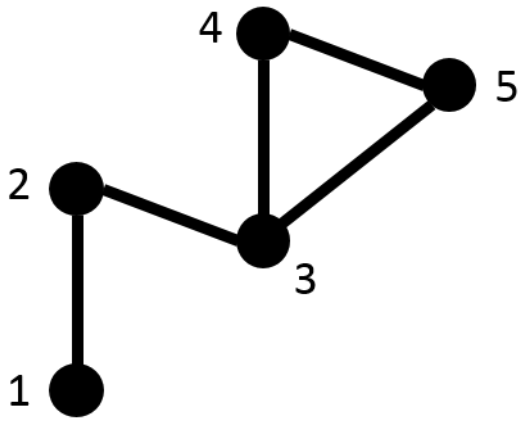

Figure 1: A simple network.

Questions

1. Consider the undirected network depicted in Figure 1.
   (a) Draw a histogram of the degrees in this network. 2 points
   (b) Write the adjacency matrix of this network. 4 points
   (c) Compute two different types of centrality for each vertex [2] (you can choose which two; see the link on the first page) by hand, and describe how you did it. 10 points
   (d) The vector of the degree per vertex can be obtained by a suitable, single matrix-vector multiplication. Show how, and explain why. 4 points
   (e) We can convert this undirected network to a directed one by choosing one of two possible directions for every edge. Prove that no such 'directionalization' exists such that from all vertices you could reach (or infect) the entire network. 4 points
   (f) Compute the *local*[3] clustering coefficient for every vertex. 7 points
   (g) What is the minimum number of edges that must be added so that the clustering coefficient is 1 for every vertex? Why? 4 points
2. Consider a random (Erdos-Renyi) network of N nodes and a probability p per possible edge to be present in the network.
   (a) What is the expression for the probability distribution for the degree of a single vertex? Why is this the right type of probability distribution?

   5 points

[2] Note: for attempting the eigenvector centrality and the betweenness centrality you will earn +0.1 bonus credit (each) for this entire assignment (but the maximum possible score remains 10.0).

[3] `http://www.wikiwand.com/en/Clustering_coefficient`

(b) For a given vertex that has degree k, what is the expression for the probability that its clustering coefficient equals exactly t?

7 points

(c) Explain why the expression for the expected local clustering coefficient for a vertex in the entire network equals $\sum_{k=1}^{N-1} p(k) \cdot \sum_{i=0}^{k(k-1)/2} \frac{i}{k(k-1)/2} \cdot \text{Binom}\,[i; k(k-1)/2, p]$.

7 points

(d) Use this expression to prove that the expected clustering coefficient actually simply equals p.

7 points

(e) Suppose you pick a random vertex from the network. What is the expected degree of this vertex, denoted $< k >$? Explain briefly why.

2 points

Epidemic spreading, introduction

Epidemic spreading on networks have been studied a lot in the past few decades. This research started with epidemics on random (Erdos-Renyi) networks and then progressed to other models of networks, mostly Barabasi ('scale-free') networks and Watts-Strogatz ('small-world'). Most studies focused on situations where an ODE ('mean-field') approximation would no longer yield accurate results, since otherwise Occam's razor tells us that we should prefer the ODE model over the networked model. In this assignment, you will investigate an example such situation for yourself, answering some theoretical questions as well as performing computer simulations.

Note: when we say 'epidemics', this might naturally suggest infectious diseases. However, the term is used more broadly to represent all kinds of 'spreading phenomena' which also includes spreading of, for example, gossip in online social networks, water in rock cavities, information in sensor networks, or a memo in an organization.

Epidemics, question 1

We begin by modelling an epidemic using a network, using the simplest model possible, the so-called SI (Susceptible-Infected) model. As you may have guessed, the model consists of two types of vertices; the vertices could, for example, represent humans, rats, wireless sensors in a dyke or cavities in a rock.

A vertex in the 'susceptible' state changes to the 'infected' state if (at least) one of its neighors infects it. A vertex in the 'infected' state will, for each timestep, infect a neighbour vertex with a probability of i for each neighbour.

Let us denote the total number of susceptible vertices as S and the total number of infected vertices as I. Naturally, $I + S = N$ at all times. In epidemiology jargon, $I(t)$ is referred

to as 'prevalence' of the epidemic at time step t.

(a) Suppose that a susceptible vertex has r infected neighbors in a certain time step. What is the probability that the vertex is not infected in the next time step? What is the probability that it *does* become infected?

3 points

(b) Implement this model for random (Erdos-Renyi) networks, using Python. Review Appendix A for advice on how to do this. Show one figure with the expected evolution of I/N (i.e., normalized 'prevalence') over time for two cases: (i) $N = 10^5, i = 0.01, \langle k \rangle = 5.0$, and (ii) $N = 10^5, i = 0.1, \langle k \rangle = 0.8$.

Initialize the network with a proportion of 0.1 of the vertices infected. Note that we want to see the *expected* behavior, so you'll need to average over enough simulations for each case to get a smooth curve, preferably including confidence intervals or error bars.[4] 10 points

(c) We can see that, initially, one case grows faster than the other. Could we have predicted this beforehand, by looking only at the values $N, i, \langle k \rangle$?

2 points

(d) We see two 'opposing forces', namely a higher infection probability but a lower connectivity, so it is not so clear at first sight. Let's look more closely. In the early phase of the epidemic, it is highly likely that all neighbors of an infected vertex are susceptible. Assume this is the case, and write the expression for the probability that a vertex with (say) $\langle k \rangle$ edges infects at least one neighbor.

2 points

(e) Compute this probability numerically in each of the two cases, for the first time step. Check this result by comparing it to the initial ordering[5] of the two curves for question (2b); does the ordering correspond?

1 points

(f) Derive (analytically) the expression for the expected number of new infections in the first time step *per infected vertex* in both cases, i.e., $I'(t)/I(t) = (I(t+1) - I(t))/I(t)$. This number is called the 'reproduction number' in epidemiology and often denoted R_0.

3 points

(g) Use the ratio of the two reproduction numbers to quantify exactly how much faster/slower case (i) is expected to grow compared to case (ii) in the beginning.

2 points

[4] You will be able to verify your implementation later using the mean-field approximation.

[5] I.e., which of the two curves is above the other one in the early steps of the simulation

(h) Can you visually verify the absolute value of this ratio for question (2g) using your numerical results? Try it, by plotting one figure with two curves which approximate the R_0 over time for each case, using your numerically calculated (averaged) $I(t)$.

Does it seem to correspond well, in the beginning? (If your curves are too jumpy, run additional simulations for better averaging.)

7 points

(i) Why do these curves drop off to zero? And why does the top curve drop to zero faster than the bottom one?

5 points

(j) Now, we want to consider the final phase of the two epidemics. In your figure for question (2b), we see that they settle to different final outbreak sizes. Explain, based on graph theory, why you would expect this to happen.

3 points

(k) As $N \to \infty$, how will both final outbreak sizes grow as function of N? Your answer may omit constants.[6]

4 points

Epidemics, question 2

Let's try making a mean-field approximation of these two epidemic models. We'll need to remove any 'spatial' structure that the network induces. We can easily do this by making the network complete, so that all $N(N-1)/2$ edges are present.

Then, we could approximate how $I(t)$ and $S(t)$ behave over time using the following (continuous-time, continuous-state) coupled ODE:

$$\frac{dI}{dt} = \left(1 - (1-b)^{I(t)}\right) \cdot S(t), \tag{1}$$

$$\frac{dS}{dt} = -\left(1 - (1-b)^{I(t)}\right) \cdot S(t). \tag{2}$$

We want our models to be the same as those from Question 1, so we'll use the same values for N, and use the following (equivalent) initial conditions:

$$I(0) = 0.001N, \tag{3}$$

$$S(0) = N - 0.001N, \tag{4}$$

[6] You may also use big-O notation if you wish.

(a) The first thing to note is that there is now a b parameter which is *not* equivalent to the i parameter above. That is, we cannot simply set $b = i$ and then expect that this ODE will be an approximation of the networked models above. Explain briefly why.

4 points

(b) To make the ODE approximate the networked models above, we need to compute a value for b, based on i. We can require that the two models will have the same behaviour in the beginning, by making the initial values of R_0 equal, using this equation:

$$\left(1 - (1-b)^I\right) \cdot S = \left(1 - (1-i)^{\frac{\langle k \rangle}{N} I}\right) \cdot S. \tag{5}$$

Derive the expression for $b = ...$ from this equation.

4 points

(c) Use this expression to compute the two numerical values corresponding to the above two networked cases; write the results up to 2 decimals.[7]

2 points

Epidemics, question 3

Now, we want to evaluate whether our mean-field approximation sufficiently represents the original models.

(a) In the first exercise sheet (ODEs) of the Networks part of this course you should have implemented the Euler algorithm for numerically approximating the solution of an ODE. Extend this algorithm to support a coupled ODE of two functions (namely, I and S).

Show one figure with four curves. Start with the two from question 1 (2b); the other two should be your numerically approximated solutions for the ODEs that correspond to these two networked cases. Clearly distinguish the ODE approximations from the network results, and make them as accurate as possible.

10 points

(b) If you did everything correctly, then the ODE should approximate one case relatively well, but the other case should deviate more significantly, at least in the final phase. Explain (briefly) what could cause the difference.

5 points

(c) Looking at the previous figure, it looks like both the ODE and the random network model have an initial phase of a certain type of growth of the prevalence. Explain

[7] If you fail to derive b from i, you can use the values $b = 5.00 \cdot 10^{-7}$ for case (i) and $8.40 \cdot 10^{-7}$ for case (ii) instead, for the remaining questions.

what type of growth it is, and justify your answer using the formulas.

5 points

Epidemics, question 4

One of the parameter values $N, i, \langle k \rangle$ from Question 1 gives us a Erdos-Renyi network model with a giant component. Instead of using Erdos-Renyi for the model using these parameters, we can try using the *scale-free network* model instead.

(a) For the $N, i, \langle k \rangle$ for which a giant component exists in the Erdos-Renyi network model, repeat the simulations, using the *scale-free network* model instead, using exponent $\gamma = 2.5$.[8] Show one figure in which you compare the 'scale-free' case with the 'random' case.

8 points

(b) Can you explain the difference in the shape of the prevalence curve?

2 points

(c) In every time step of the simulation, keep track of every vertex that is being newly infected. Show one figure with the average degree of the newly infected vertices for the scale-free network and the random network.

4 points

(d) Based on this figure, what can you say about how the epidemic typically spreads through a scale-free network?

2 points

[8] see the Appendix if you need a hint on how to generate scale-free networks with an arbitrary exponent.

A Creating an Erdos-Renyi (random) network

Some of the code given here may be deprecated.

In order to do research on random graphs, you have to be able to create one. You can either write your own code in Python (probably slow), or you can use `networkx` to generate your networks. The `networkx` package is recommended, as it is relatively efficient and easy to use.

You can create a random graph with `networkx` using the following example code:

```
import networkx as nx

# Create a graph with N = 10^5 and <k> = 1.0
G = nx.fast_gnp_random_graph(10**5, 1.0/10**5)

# Print all neighbors of the first node
neighbors = nx.all_neighbors(G, 0)
for neighbor in neighbors:
    print(neighbor)
```

B Creating a power law network

The package `networkx` does not have a method to directly create a scale-free graph, but it's still advised to use it. You can create a scale-free graph using the following code. It draws a number of degrees from the powerlaw distribution and uses that to create a random graph.

```
import networkx as nx
import numpy as np

# generate sequence of expected degrees, one for each vertex,
# with p(k) ~ k^(-2.5)
s = nx.utils.powerlaw_sequence(100000, 2.5)

# rescale the expected degrees to match the given average degree
s = s / np.mean(s) * avgk

# generate network probabilistically
graph_sf = nx.expected_degree_graph(s, selfloops=False)
```

Or alternatively:

```
import networkx as nx
import matplotlib.pyplot as plt

N = 100000
e = 2.5
k = 5

# Create a graph with degrees following a power law distribution
s = []
while len(s) < N:
    nextval = int(nx.utils.powerlaw_sequence(k, e)[0])
    if nextval != 0:
        s.append(nextval)

# TODO: s must be scaled and rounded such that the average degree equals k

# Sum of degrees must be even. One way to solve this is to add a
# single node with degree 1
if sum(s) % 2:
    s.append(1)

G = nx.configuration_model(s)
G = nx.Graph(G)

# Remove self-loops.
G.remove_edges_from(G.selfloop_edges())
```