# Exercise 11: Networks 3

Kristian Urup Olesen Larsen[1]

[1]kuol@econ.ku.dk

May 6, 2020

## Problem 11.1.1. - SIS model

Building the SIS model is relatively straight forward. First of all we require $S + I = k$ where $k$ is some fixed constant. This restraints the dynamic to fixed populations. We also take some initial levels $S_0, I_0$ as given. Then, the number of infected at any specific time will change due to *i)* new infections $\beta SI$ with the number of new infections being proportional in both the number of susceptible and the number of currently infected, and *ii)* the number if recoveries from infected to suceptible (again), $\gamma I$. Thus in total

$$\frac{dI}{dt} = \beta SI - \gamma I \tag{1}$$

Since there are only two states and the population is of a fixed size naturally it must be that

$$\frac{dS}{dt} = -\frac{dI}{dt} \tag{2}$$

## Problem 11.1.2. - SIRS model

In the SIRS model the removed don't nessecarily stay removed but instead transition into becoming suceptible again with a fixed probability. Note that this model does not include deaths because all of the removed can reenter the suceptible population. If death was possible the reentry rate would depend on the composition of the removed. In any case, this model looks very much like the SIR model but with an added $\phi R$ term, where $\phi$ is the reentry rate from the recoved to suceptible populations

$$\begin{aligned} \frac{dI}{dt} &= \beta SI - \gamma I \\ \frac{dR}{dt} &= \gamma I - \phi R \\ \frac{dS}{dt} &= -\beta SI + \phi R \end{aligned} \tag{3}$$

That is in words

- The number of infected increase by $\beta SI$ at any given time (i.e. is proportional to the number of suceptible "candidates" and to the number of current infected "donators"), while it decreases by $\gamma I$ which is the constant fraction being removed.
- The number of removed increase by the constant fraction entering from the infected compartment and decreases by the constant fraction reentering succeptibility.
- The number of suceptible decrease by the amount that becomes infected, and increase by the amount that reenters from having recovered.

## Problem 11.1.3. - Explorables play

I will skip this and let you play with the buttons yourself.

## Problem 11.1.4. - Explorables play #2

I will skip this and let you play with the buttons yourself.

## Problem 11.1.5. - SI model simulations

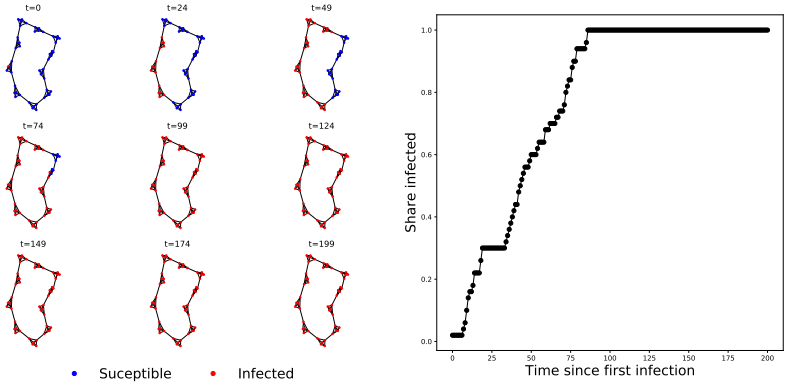


**Fig 1.** SI model simulations

In this problem we will modify Ulf's code to store the fraction infected over time when we simulate from the SI model. Since this is straight forward we will also draw the network as the disease spreads through it to get a better understanding of how the process looks in the network. The first thing we need is to import some things from matplotlib, in particular gridspec which will help us set up complicated subplots and Line2D which we will need for making a custom legend.

```
from matplotlib import gridspec
from matplotlib.lines import Line2D
```

Next, the same setup that Ulf provided in the exercise text

```
G = nx.connected_caveman_graph(10, 5)
p_I = 0.1
n_iter = 200
I = set()
S = set(G.nodes())
patiento = np.random.choice(list(S))
I.add(patiento)
S.remove(patiento)
```

And a list that can store the fraction infected over time as we run the simulation. I prepopulate this with the initial share infected as we would otherwise miss the first time period in our results.

```
N = len(G.nodes())
frac_I = [len(I)/N]
```

Now to draw the networks I will write a simple function that uses the network G and the list of suceptible S to draw a color coded network on a matplotlib axis ax

```
def gdraw(G, S, ax):
    nx.draw(G,
            ax = ax,
            pos = nx.spring_layout(G, seed = 10),
            node_color = ['blue' if n in S else 'red' for n in range(N)],
            node_size = 10)
```

The function doesn't return anything since it modifies ax *in place*. Now we are ready to begin modifying the loop Ulf provided. First, before even entering the loop I will set up a new figure fig and a grid gs. I can then plot the $t = 0$ infection pattern as a network in the upper left corner of the figure,

```
fig = plt.figure(figsize = (16,8))
gs = gridspec.GridSpec(3, 6)
x = 0
y = 0

ax = plt.subplot(gs[x,y])
gdraw(G, S, ax)
ax.set_title('t=0')
```

Now entering Ulf's loop the first lines are unchanged

```
for t in range(n_iter):
    for infected_node in list(I):
        neighbors = G.neighbors(infected_node)
        infected_neighbors = set([n for n in neighbors if np.random.random() < p_I])
        I |= infected_neighbors
        S -= infected_neighbors
```

After each full run through the infected nodes I first update frac_I to reflect any change in infection counts that occured at time $t$

```
    frac_I.append(len(I)/N)
```

and then, if the current time is one of 8 evenly spaced time periods i add a network figure to the grid by

```
    if ((t+1)/(n_iter/8)).is_integer():
        y += 1
        if y > 2:
            y = 0
            x += 1
        ax = plt.subplot(gs[x,y])
        gdraw(G,S, ax)
        ax.set_title(f't={t}')
```

This is it for the loop body. After the loop terminates we still need to plot the large panel of the figure containing the share infected over time. To do this we pick out all rows of the figure grid and all columns after column 2 and create a subplot from these grid points. Then plotting the data is as simple as calling plt.plot

```
ax = plt.subplot(gs[:,3:])
ax.plot(frac_I, 'o-', color = 'black')
ax.set_xlabel('Time since first infection', fontsize = 20)
ax.set_ylabel('Share infected', fontsize = 20)
```

Finally I want to add a legend to the networks to show readers that blue dots are suceptible and red dots are infected individuals. To do this set up a list of custom legend entries

```
custom_entries = [Line2D([0], [0], marker = 'o', color='blue', label='Suceptible', lw=0),
                  Line2D([0], [0], marker = 'o', color='red', label='Infected', lw=0)]
```

in this case two Line2D objects with circle markers and zero line width. Then grab the lower middle grid point $(2, 1)$ of the network plots and plot a legend here.

```
ax = plt.subplot(gs[2,1])
ax.legend(handles=custom_entries,
fontsize = 20,
frameon = False,
loc='upper center',
bbox_to_anchor=(0.5, -0.05),
ncol=2)
```

For the last question you are asked to repeat the above but this time making infected nodes recover and become resuceptible again after a fixed number of time steps. I will not retype the whole code here but instead give you some indication about how I have solved this problem. First, to know which nodes were infected $T_I$ periods ago we need a memory that extends back $T_I$ periods. That is we need a datastructure that looks something like $[I_t, I_{t-1}, I_{t-2}, ..., I_{t-T_I}]$. When we take a step forward in time the oldest value should be discarded and a new one added to the front of the data. One way to implement this is by a *ringbuffer* which in python could look something like

```
class RingBuffer:
    def __init__(self, length):
        self.length = length
        self.data = [None]*length

    def add(self, value):
        if len(self.data) < self.length:
            self.data.insert(0,value)
        else:
            self.data = self.data[:-1]
            self.data.insert(0,value)

    def take(self):
        return self.data.pop()
```

In the setup code we can then initialize a ringbuffer to store the history of infections

```
T_I = 10
rb = RingBuffer(T_I)
```

We will also need to modify the code to keep track of new infections in each time period. Assuming we store the newly infected in a set called `new_infections` the updating is then as simple as running

```
rb.add(new_infections)
recoveries = rb.take()
if recoveries is not None:
    S |= recoveries
    I -= recoveries
```

once every time step.