Study Project: Epidemic Modelling - A new Compartmental Model May 2023

End-Semester Report

Astitva Sehgal

1 Learning basics of Epidemic theory

We started of the project by an article reading :

Physics today

We learnt about a few basic concepts in epidemic modelling, most relevantly:

- Compartmental (Non-Network) based modelling
- Network Based modelling
- R_o = The unitless number is defined as the average number of new cases, or secondary infections, caused by a typical infected individual in a susceptible population.

2 The SIS model

The SIS model is a compartmental model that has only 2 compartments of people, Susceptible and Infected.

Susceptible people can become infected again, and infected people can recover to become susceptible again.

The governing equations of this compartmental model are:

$$\frac{dI}{dt} = \beta SI - \gamma I$$
$$N = I + S$$

which is a system of equations in two equations with two unknowns, I (number of infected people) and S (number of susceptible people).

Substituting $S = N - I$ in the system gives us a single nonlinear differential equation that governs the SIS model:

$$\frac{dI}{dt} = \beta(N - I)I - \gamma I$$

This euation has been solved analytically, to give results for the compartmental model.

3 Novel Model: SIS-m

In this report, we discuss a novel compartmental model based on the original SIS model. The novel idea being that an SIS model doesn't differentiate between nfected people based on how likely they are to spread the infection. All infected people are considered equally likely to spread the infection forward, and all we have is one value for β. We would like to take this into account and build a model that hopefully is closer to the real situation.

We define the SIS-m model as having (m+1) compartments. 1 compartment is reserved for all the Susceptible individuals. There are m infected compartments, each of the m compartments having individuals with varying likelihood of spreading the infection to others, thus having their own specific $beta_i$'s.

In this project, we focus only on the SIS-2 model, as a starting point to further study, and to check whether SIS-m models should be explored in more detail.

4 SIS-2 Model

Since we just have 2 infected compartments, we consider the population itself to be separable into two parts, Critical individuals (more likely to spread infection if infected) and Non-critical individuals (less likely to spread infection if infected).

The "known" quantities we have are:

- N - Total number of individuals
- β_c, $\beta_n c$ - The β values for the critical and non critical compartments
- γ - The γ value for infected individuals (Note that this is the same for both critical and non critical individuals, as it depends on the disease type and not on how likely and individual is to spread the disease further).

The unknowns we have are:

- S_c - The number of susceptible people of criticial type.
- S_{nc} - The number of susceptible people of infected type.
- I_c - The number of infected people of critical type
- I_{nc} - The number of infected people of non critical type

We choose to convert these above unknowns into ratios rather than absolute numbers. Thus the converted unknowns, in terms of the original unknowns are:

$$\rho_c^s = S_c/N$$
$$\rho_{nc}^s = S_{nc}/N$$
$$\rho_c^i = I_c/N$$

$$\rho^i_{nc} = I_{nc}/N$$

These are the unknowns we will be dealing with from now on. Now, we have 4 unknowns, that need 4 equations to solve.

The first two equations are fairly easy to see.

$$\rho^i_c + \rho^s_c = N_c/N = \rho_c$$

where N_c is the total number of critical people in the population.

$$\rho^s_{nc} + \rho^i_{nc} = N_{nc}/N = \rho_c$$

where N_nc is the total number of non critical people in the population.

The next two equations are the interesting ones. These deal with the rate of change of ρ^i_c and ρ^i_{nc} based on the β_c, β_{nc} and γ values.

$$\frac{d\rho^i_c}{dt} = \beta_c(\rho^i_c + \rho^i_{nc})\rho^s_c - \gamma\rho^i_c$$

$$\frac{d\rho^i_{nc}}{dt} = \beta_{nc}(\rho^i_c + \rho^i_{nc})\rho^s_{nc} - \gamma\rho^i_{nc}$$

These two equations are inspired by the SIS model, while keeping the exact new model in mind.

The first two equations can be used to substitute ρ^s_c and ρ^s_{nc} in the last two equations.

Thus we get the last two equations as:

$$\frac{d\rho^i_c}{dt} = \beta_c(\rho^i_c + \rho^i_{nc})(\rho_c - \rho^i_c) - \gamma\rho^i_c$$

$$\frac{d\rho^i_{nc}}{dt} = \beta_{nc}(\rho^i_c + \rho^i_{nc})(\rho_{nc} - \rho^i_{nc}) - \gamma\rho^i_{nc}$$

These are the two defining equations for the SIS-2 model.

5 Solution to SIS-2 model

The two defining equations is a system of 2 coupled non-linear ordinary differential equations. An analytical solution to these equations was attempted, however not achieved. More time wasn't given to the first aim of the project was to verify if SIS-m models are even a good representation of epidemic modelling. Thus, a numerical solution would suffice.

For this, the RK-4 method was used in MATLAB.

The two code files for this (written by Toby Sir and taken as the standard for the project):

SIS2 Equations

SIS2 runner script

This script will be used for checking the validity of the SIS-2 model, against network simulation data.

Note that in the simulation γ is set to be 1, since β values will be adjusted accordingly.

6 Network Simulation

The project had 3 parts:

- Setup the differential equations for SIS-2
- Setup a network simulation
- Compare the results of the two.

We have done Part 1 above. For Part 2, an original Python code was written (Jupyter notebook):

Network Simulation

The colab notebook has a well-documented and modularised. It has custom functions to make simulating for a specific network much quicker and easier. These functions are described below:

- makeGraph(N,Pk,Pe1,Pe2)

  This function is used to make a biased random graph with N nodes.

  Pk is the probability of a node being more biased to make edges (defined as Type 1 node)

  Pe1 is the probability of a Type 1 Node of making an edge with any of the other vertices.

  Pe2 is the probability of a Type 0 Node of making an edge with any of the other vertices.

  $Pe1 >= Pe2$ is a logical condition that should hold.

  The function returns an object of the class networkx.Graph() networkx is a well documented python module:

  networkx Documentation

  The function also initialises a few global variables that can be used if needed.

  - network - adjacency list of the graph
  - edges - list of edges u,v in the graph
  - infected - a set of infected nodes

- resetGraph(G) This takes a graph G (networkx.Graph() object) to it's initial state, essentially setting all nodes as susceptible without altering the structure of the graph.

- drawGraph(G)

  This plots the graph G (networkx.Graph() object).

- countCritical(G,k)

  This counts the number of "Critical" nodes in the graph and returns that integer. From a network perspective, the definition we have employed for a "Critical" node is a node having more than k edges. This definition may vary for different simulation techniques. "Critical" only means that the node/agent is more likely to spread the infection than others.

  This function also initialises a global set "criticals" that holds all the critical nodes.

- runSim(G,steps, mu, r)

  This function runs the epidemic simulation on a graph G (networkx.Graph() object), for steps number of timesteps.

  Here

  - r - Probability that an infected node spreads the disease through an edge (i.e. to a neighbour) in one unit time.

    Note that this is the probability of infection spreading through an edge. Not the probability of an infected person spreading the infection to at least one susceptible neighbour.
  - mu - The probability of an infected node recovering in one unit time.

  Note that the simulation works in the manner than in one timesteps an infected node first has the chance of infecting its neighbours through its edges, and then the chance of recovering.

  This function uses a lot of intermediate functions like startInfection(G) (to randomly start an infection in a node of G), timestep(mu,r) (for one timestep of the epidemic simulation), resetGraph(G), etc.

  The user need not worry about these internally used functions.

  The functions returns an array infectedList, which captures the ratio of infected people (to total population) at every timestep of the simulation.

  This can be easily plotted using matplotlib.

6.1 How to run the simulation

1. Create a static Graph, with the appropriate parameters:

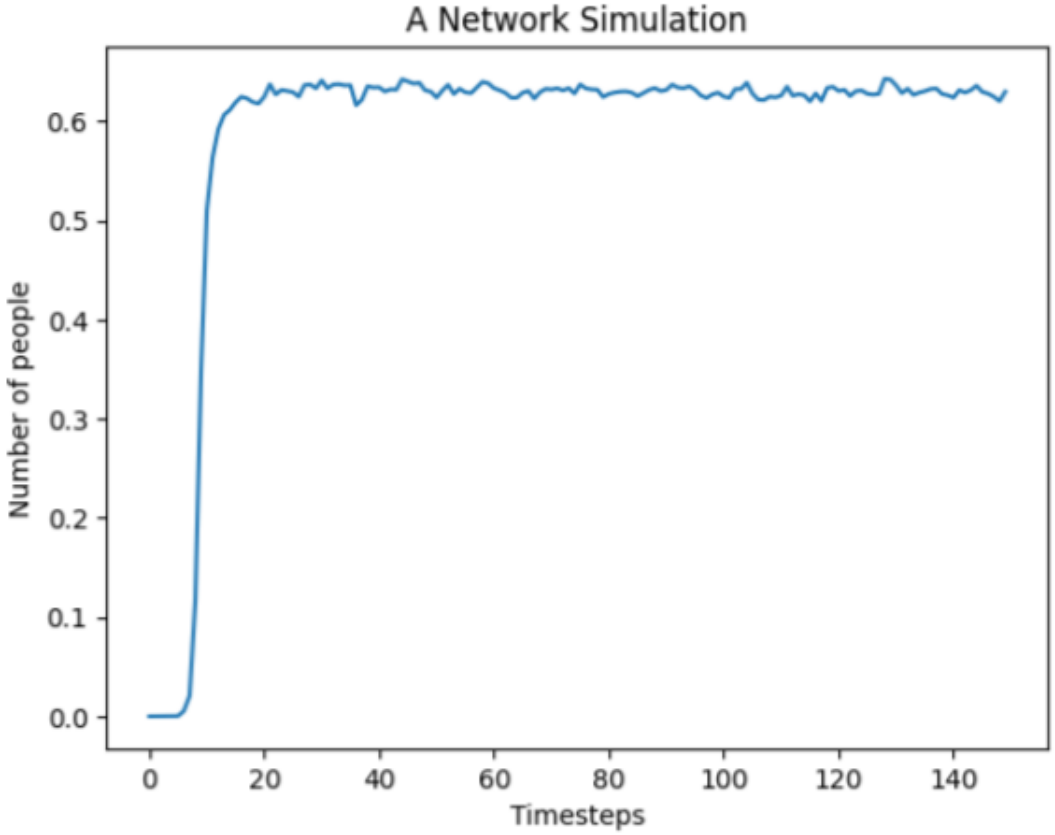
   G = makeGraph(10000,0.025,0.0001,0.1)

2. Run the simulation, with appropriate parameters.

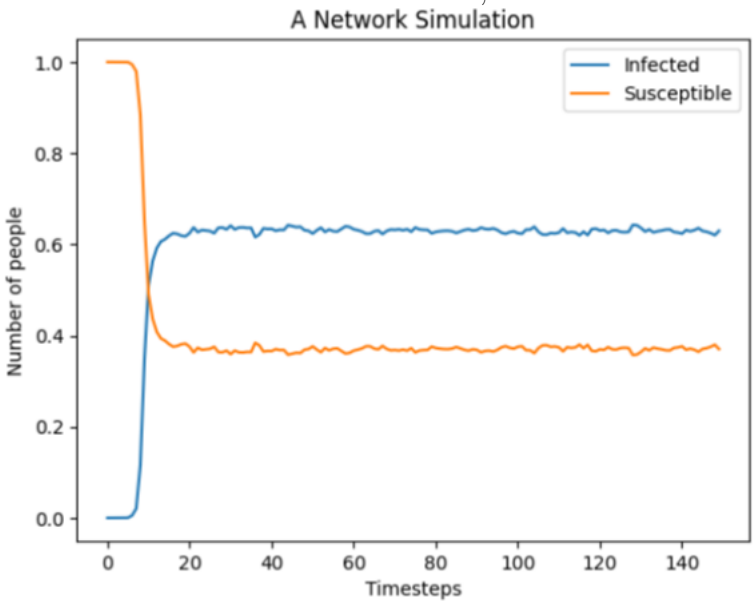
   infectedData = runSim(G,3000,0.2,0.05)

3. Plot the results (infectedData):

plt.plot(infectedData[:150], label = 'infected')
plt.title("A Network Simulation")
plt.xlabel("Timesteps")
plt.ylabel("Number of people")



This simulation code has been tested, and has shown no errors till now.



7 Going further

We have setup the differential equations and created a simulation environment for the project.

What is left for us to do is Part 3, i.e. analyzing simulation results on appropriate parameters, and comparing (and fitting) them to our SIS-2 model results (that we obtain numerically through MATLAB).

Thank you.