Lab #2 – simulating Covid in a population

Goals of the lab:

In this lab, we'll

- Learn about infectious-disease simulation – how it works and (hopefully) when to (dis)trust it.
- Write simple Python code to implement a simulation. It builds on our previous assignment by adding loops, 1D arrays (slightly) and graph plotting.

Equations of the model

Your code should implement the following representation of a population:

- There are *N_PEOPLE* people (set it to 100 for our simulations).
- People can be in any of 5 categories: Susceptible, Infected/Latent, Infected/Invisible, Infected/Visible and Removed.
- Track the number of people in each category with the variables S, IL, II, IV and R. You should always have S+IL+II+IV+R = *N_PEOPLE*.

On day 1, start with one person in (IL) and the rest in Susceptible (S). I.e., IL=1 and S=*N_PEOPLE*-1.

Every day after that, move people between groups as follows:

- The number of people contracting the disease (and hence moving from S to IL) on any day is given as follows. Each person in II comes into close contact with *beta* new people. Of those people, the fraction *S*/*N_PEOPLE* actually catch it. However, if (e.g.,) II=60, BETA=2 and there are only 100 susceptible people, then you cannot have 120 people being infected! Thus, you should cap the calculations as per slide 10 of the lecture.
- The fraction $\frac{1}{N\_DAYS\_IN\_IL}$ of the IL group moves from IL to II.
- The fraction $\frac{1}{N\_DAYS\_IN\_II}$ of the II group moves from II to IV.
- The fraction $\frac{1}{N\_DAYS\_IN\_IV}$ of the IV group moves from IV to R.
- R is roach motel of simulation – once you're in it you never leave.
- Even though there is no such things in real life as a fractional person 😊, you should not round your numbers to the nearest integer. The reason is that, with the simulation starting with only 1 person in IL, rounding the number of people moving from IL → IL to the nearest integer will typically result in zero – which means your simulation will quickly get stuck!

You should run the simulation for *N_SIM_DAYS* days. Set it to 100 for this lab.

Coding guidelines

Please follow good coding guidelines. Specifically:

- Declare and initialize all of your main disease parameters BETA, DAYS_IN_IL, DAYS_IN_II and DAYS_IN_IV all together near the top of the program.
- Initialize your population variables S, IL, II, IV and R together as well.

- For consistency across the class, please indent loops by 4 spaces for each loop level.
- Comment any code whose purpose may not be obvious.

Accumulating the results for plotting

You will be plotting your results vs. time. In order to do so, you must accumulate the results into a Python list. We've only started to learn about lists, so we'll give you some code for this.

First, *outside of your main simulation loop*, initialize your lists:

```
S_by_day  = []
IL_by_day = []
II_by_day = []
IV_by_day = []
R_by_day  = []
```

Then, inside your simulation loop after you compute the new values of S, IL and R for any given day, update your lists as follows:

```
S_by_day.append (S)
IL_by_day.append (IL)
II_by_day.append (II)
IV_by_day.append (IV)
R_by_day.append (R)
```

Main outline

In general, the big outline of the code should look something like this:

```
Declare and initialize all of your constants
Declare and initialize your main simulation variables, and your lists that track them
For each day:
        Figure out how many people will move from group to group (no rounding)
        Update the group sizes accordingly
        Save the day's results for plotting
Do your final plotting
```

This type of high-level program description is typically called *metacode*, and can be quite helpful! In particular, note that it's quite helpful to *first* compute how many people move from group to group, and then implement the moves. If you mix those two steps together, you may have subtle issues.

Plotting your results

For each of the first three simulations that you run, please turn in a single graph that shows the number of people in *S*, *IL*, *II*, *IV* and *R* vs. time.

The course web page has a link for basic tutorial on how to plot in Python with Matplotlib, and we can also discuss it during our recitation. Finally, here's a small recipe that will get you *most* (but not all) of the way there:

```
import matplotlib.pyplot as plt
plt.figure ()
```

```
plt.plot (range(n_days), S_by_day,  label='S')
plt.legend (stuff…)
plt.show ()
```

What simulations to run:

Run the simulations indicated in the table below, and then fill in the rest of the table.

beta	days_in_IL	days_in_II	days_in_IV	Total infections	R_0	Herd immunity
.25	3	3	5	*3.72*	*.75*	*N/A*
.25	3	4	5			
.5	3	3	5			
.5	3	4	5			
.5	8	4	5			
.5	3	4	10			

The first four columns are the simulation parameters for you to set. "Total infections" is the total number of people who have been infected by the end of the simulation, which your code should calculate.

R_0 is the basic infectivity discussed in class. "Herd immunity" is the number of infected people required to reach herd immunity. You can either compute these by hand or have your code do it. The relevant equations are in the slides, and the top row is already filled in for you as a means of checking yourself (it's possible that your number for total infections will be slightly different than mine).

Questions

After filling in the table, please answer the first two questions, and then choose any one of the next three:

1. Compare the top four lines of the table. Consider the large change in the total number of people infected across the four lines, and the relatively small changes in the various parameters. What does this say about our ability to predict the course of an epidemic such as Covid?
2. A little bit of math says that if each person contacts R_0 other people, then at the beginning of a pandemic when |S|≈N_PEOPLE the number of infections vs. time is roughly ${R_0}^{t/\tau}$, where τ is the average time lag for one person transmitting the virus to another (i.e., the *generation* time). The "new" B.1.1.7 strain has roughly R_0=1.5x higher than the original strain. If $\tau \approx 4$ days, and if our current measures are effectively creating R_0=1 for our original strain, then how long would it take until one infected person infects 1M others?

Pick any **one** of these three questions:

3a. Does the total number of infected people change when you change *days_in_IV*? How about *days_in_IL*? Explain why or why not.

3b. Even though herd immunity is the point where the epidemic is essentially under control, clearly it does not exactly match with the total number of infections (as shown by the two relevant columns in the table). Why not?

3c. The slides show how to compute the herd-immunity threshold (which is presumably how you filled in the table above). Is there a way you can look at the graph and read the herd-immunity threshold reasonably well, without needing the equation?

Challenge problem

The next step up in simulation accuracy is called a *individual-based simulation*. The prerecorded video talked about these and gave some basics about why they can be more accurate. If you had all the time in the world to write a better simulator, and it was individual based, what types of effects would you like to model?

What to turn in:

You Python code *lab2_covid_population_sim.py*, as well as a lab report with the table above, the three graphs and the answers to three questions.

***Grading*:**

- Code correctness: 45 pts
- Code clarity: 10 pts
- Questions: 15 pts each