Simulating Covid-19 with 6 lines of code — the SIR epidemic model

Bhaskar Krishnamachari
Mar 22 · 6 min read

We have been talking about mathematical modeling and python programming in my undergraduate course on Internet of Things this semester. I would like to tell my students how these tools we are discussing in class have a practical relevance in today's world, by showing them how they can help them gain some more concrete understanding of Covid-19 epidemic propagation and the implication of different policies such as the "shelter in place" directive currently in effect in California.

I will use the classic SIR model of epidemics , which is originally attributed to the work of Kermack and McKendrick from around 1927. To keep the presentation as simple as possible, instead of the conventional continuous differential equation model, I'm going to use a simpler discrete-time difference equation approach that will track the growth of the infection at one day intervals.

The SIR model is named after the three categories into which the population is categorized: **S for susceptible** (these are individuals that may get infected but are not yet infected), **I for infected** (the currently infected individuals), and **R for recovered** (individuals that are no longer infected and also, in this model, no longer susceptible — i.e. the model assumes people cannot get reinfected). Over the course of the epidemic, individuals can go from being in the S class, to being the I class, to being in the R class. The number of individuals in the S class decreases monotonically over time, and the number of individuals in the R class increases monotonically over time. The number of infected individuals initially increases, and then decreases.

We are going to ask three questions, resulting in one equation each, each a recurrence relation. And that will be the entirety of the mathematical model. We can then write-up these three recurrence relation inside a for-loop, and along with a line to initial variables and one to print the result, this will yield us a full simulation of how an infectious disease propagates in just 6 simple lines of Python code!

Here are the three questions:

1. On the nth day of the epidemic, by how much does the number of susceptible individuals decrease?

2. On the nth day of the epidemic, by how much does the number of recovered individuals increase?

3. On the nth day of the epidemic, by how much does the number

of infected individuals change?

Let's consider each question in turn, starting with the first. Say there are S(n-1), I(n-1), and R(n-1) individuals respectively in each of the three classes at the start of the nth day. The number of susceptible individuals will decrease exactly as the number of new infections that happen among them, and these new infections must be the result of a susceptible person "encountering" an infected person and having the misfortune of then getting infected. It can be seen by the multiplication principle from combinatorics that the total number of possible pairs that could possibly meet is the product S(n-1)•I(n-1).

Say that the probability of an infection-spreading encounter happening between any particular pair of individuals during a single day is denoted by α. This probability α includes both the likelihood of the two individuals, one infected and one susceptible, coming into contact with each other, as well as the likelihood that the contact results in one infecting the other. Then the average number of new infections, and thus the *expected number by which the susceptible population will decrease* will be given by applying the principle of linearity of expectation from probability theory as α•S(n-1)•I(n-1). This gives us our first equation showing the decrease:

$$S(n) = S(n-1) - \alpha \bullet S(n-1) \bullet I(n-1) \quad \ldots\ldots\ldots\ldots (1)$$

Now let's consider the second question of how the *expected number of recovered individuals increases*. Essentially this happens only from the population of infected individuals. If we assume that the average recovery time is T, then a simple way to model this increase is to assume that on average a fraction (1/T) of the currently infected individuals will become newly recovered each day. This gives us the second equation:

R(n) = R(n-1) + I(n-1)/T ………. (2)

Finally, to figure out the change in the infected population to answer the third question, we could simply consider how we have already answered the first two questions. The first answer told us how many susceptible individuals became infected, and the second told us how many infected individuals became recovered; thus the difference between those two numbers is how much the *expected number of infected changes*. This yields us the third equation:

I(n) = I(n-1) + α•S(n-1)•I(n-1) - I(n-1)/T ………. (3)

And…. we're done!

Putting the above three equations down in a for-loop that keeps track of days, we can simply run the model over time to see what happens. This results in the following six lines of python code:

```
S, I, R, alpha, T, days = [999], [1], [0], 0.0002,
12, 200
for n in range(1,days):
    S.append(S[n-1]-alpha*S[n-1]*I[n-1])
    R.append(R[n-1]+I[n-1]/T)
    I.append(I[n-1]+alpha*S[n-1]*I[n-1]-I[n-1]/T)
print(list(map(round,I)))
```

Here the first line initializes the simulation with a total of 1000 individuals, one of whom is infected and 999 susceptible at the start. It sets a recovery time of 12 days and also sets the infection rate so that each infected person can infect 0.02% of the susceptible population in one day. (In news reports about covid-19, you may have heard about a key parameter for epidemics called R0, which is the average number of individuals infected by someone who is sick at the initial phase of the epidemic, given the parameters alpha, T, and total population N, it can be determined to be alpha*N*T, here alpha has been chosen so that the R0 = 0.0002*1000*12 =2.4). The last line prints the infected population sequence over 200 days, rounded to the nearest integer.

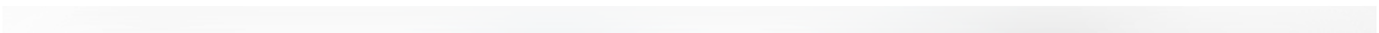
The output of the above simulation, appears as follows, showing the infection rising to a peak of 225 infected individuals, then falling off as the patients start to recover:

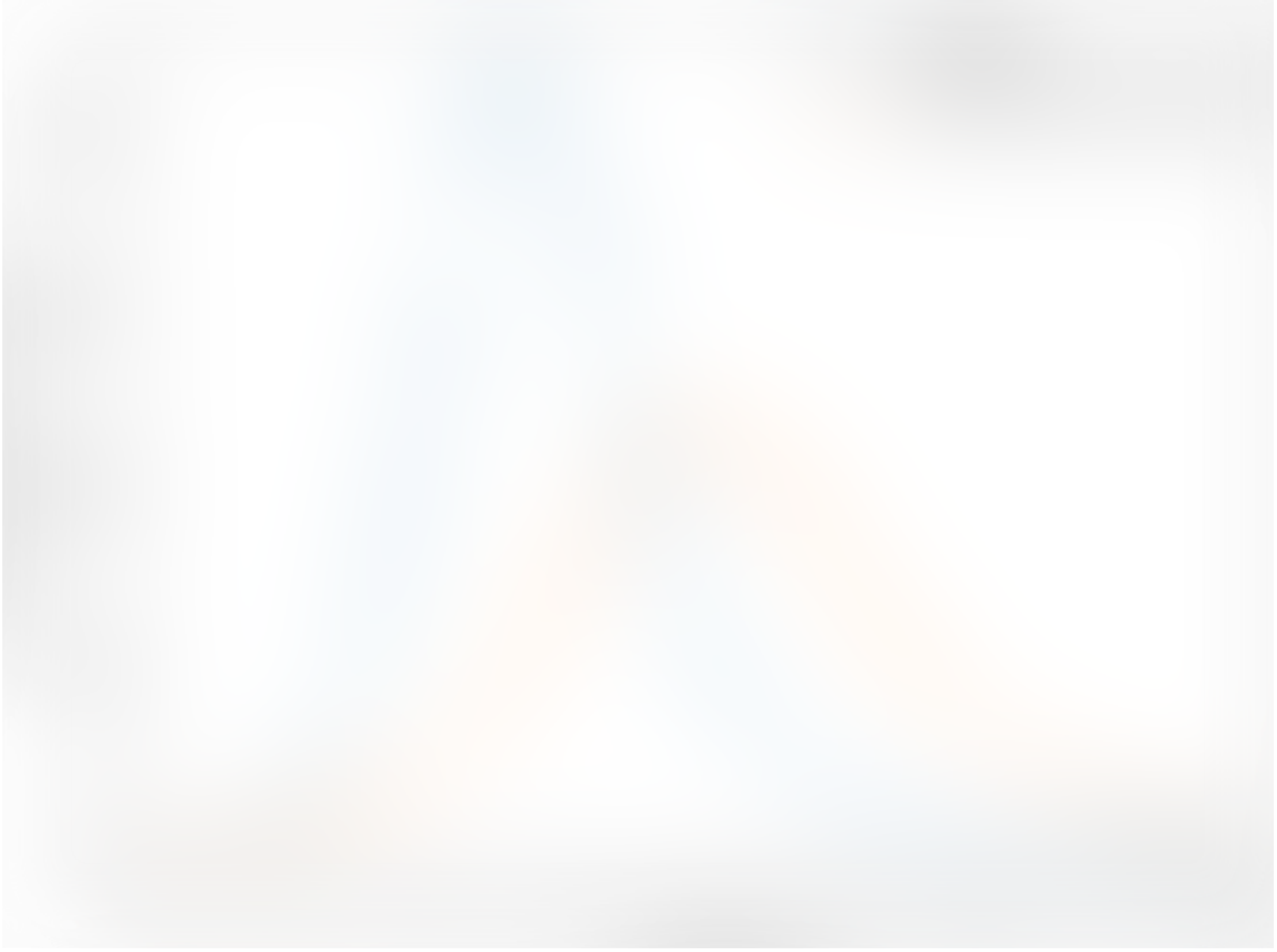
```
[1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 4, 4, 5, 5, 6,
6, 7, 8, 9, 10, 11, 12, 14, 15, 17, 19, 21, 23, 26,
28, 32, 35, 38, 42, 47, 51, 57, 62, 68, 74, 81, 88,
95, 103, 111, 120, 128, 137, 146, 155, 163, 172,
180, 187, 195, 201, 207, 212, 216, 220, 222, 224,
225, 224, 223, 222, 219, 216, 212, 208, 203, 198,
192, 187, 181, 175, 168, 162, 156, 149, 143, 137,
131, 125, 119, 114, 108, 103, 98, 93, 88, 84, 79,
75, 71, 67, 64, 60, 57, 54, 51, 48, 45, 43, 40, 38,
36, 34, 32, 30, 28, 27, 25, 24, 22, 21, 20, 19, 18,
16, 16, 15, 14, 13, 12, 11, 11, 10, 10, 9, 8, 8, 7,
7, 7, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 3, 3,
2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

To visualize the model, we can plot all three classes of individuals (just a few additional matplotlib commands can achieve this, omitted in the above), which yields the following remarkably elegant figure.

I have also made the code for the simulator available on an online python interpreter, for you to play with: https://trinket.io/python/91fb8b17ba.

Now, we can also run what-if scenarios on these models. For example, if we cut the encounter rates by 25% (such as through social distancing policies), what would that do to the infection curve? The figure below shows what this looks like (shifting the infection peak to the right and reducing its height).





I have intentionally avoided showing illustrations of the simulation with very large populations such as what might happen in an area as large as Los Angeles, though the model can certainly be simulated at any scale. However, to get simulation results that are a better fit with real-world observations, the model should be further enhanced to consider additional factors such as incubation period, geographical movements etc. Such more sophisticated simulation tools have been developed and are being deployed and used by professional epidemiologists across the world to track the current epidemic and to make policy prescriptions, such as the widely-shared Imperial college report on non-pharmaceutical interventions for Covid-19 .

Covid 19 Epidemic Epidemiology Python Mathematical Modeling