Simulating Covid-19 spread with a SIR Model

Chad Olsen

December 2024

1 Introduction

The SIR(Survival Infection Recovery) Model is a very simplistic and common model within epidemic modeling. However, this model is only equipped to predict a single outcome based on the system's initial conditions and parameters. However, disease spread is inherently random. This brings up a large flaw within the SIR model. The model itself only allows for the singular trajectory of the variables since it is a deterministic system of first-order differential equations. To better model and predict disease transmission, it would be best to include some stochastic element added to the system.

So for this project, we will perform a Monte Carlo simulation on the SIR model with its coefficients as randomly generated variables. For this to work we would need first to choose a distribution that the coefficients should follow based off of real-world transmission data, then use Likelihood estimators to find the estimated parameters for such a distribution. Once we have the MLEs, the model can be easily simulated with the randomly generated coefficients from that distribution.

At each iteration of the simulation, we get one possible randomly generated trajectory of the variables. From that, we can get the total percentage of the population that was infected, how many recovered, how many people are still susceptible, and the current growth rate of the illness. For this simulation it is focusing on building a probability distribution of infection percentages. So the system will be ran a large number of times, and after each iteration, we will record the percentage of the population that was infected.

This approach allows for a more realistic representation of transmission dynamics, providing deeper insights into the variability and uncertainty in epidemic outcomes. This distribution can serve as a valuable tool for understanding and predicting the spread of diseases in diverse scenarios.

2 Methods

The SIR model is a system of first-order nonlinear differential equations defined as follows,

$$\begin{cases} \frac{dS}{dt} = -\beta IS \\ \frac{dI}{dt} = \beta ISdt - \gamma I \\ \frac{dR}{dt} = \gamma I \end{cases} \tag{1}$$

S - Number of Susceptible people
I- Number of Infected People
R- Number of Recovered People

β - The infection rate
γ - Recovery rate

A natural choice for the distribution of β and γ is the beta distribution since they are both percentages. Then taking a set of real-world time series data on confirmed cases, and recovered cases, we can construct the Maximum Likelihood estimators. The log-likelihood function for the beta distribution is,

$$l(\alpha,\beta) = (\alpha-1)\sum^{n}\ln(X_i) + (\beta-1)\sum^{n}\ln(1-Xx_i) - n\ln(B(\alpha,\beta)) \tag{2}$$

2.1 Fisher's scoring

Then to find the values of the estimators numerically we will use Fisher scoring, where the score function is,

$$l'(\alpha,\beta) = \left(\frac{\partial l(\alpha,\beta)}{\partial\alpha}, \frac{\partial l(\alpha,\beta)}{\partial\beta}\right) = (\sum^{n}\ln(x_i) + n\psi(\alpha+\beta) - n\psi(\alpha), \sum^{n}\ln(1-x_i) + n\psi(\alpha+\beta) - n\psi(\beta)) \tag{3}$$

Where $\psi(\alpha) = \frac{\partial \ln\Gamma(\alpha)}{\partial\alpha}$ is the digamma, which can easily be calculated in Python using the digamma function within the scipy.special package.

The information matrix is

$$I(\alpha,\beta) = E[-l''(\alpha,\beta)] = \begin{bmatrix} -E[\frac{\partial^2 l(\alpha,\beta)}{\partial\alpha^2}] & -E[\frac{\partial^2 l(\alpha,\beta)}{\partial\alpha\partial\beta}] \\ -E[\frac{\partial^2 l(\alpha,\beta)}{\partial\alpha\partial\beta}] & -E[\frac{\partial^2 l(\alpha,\beta)}{\partial\beta^2}] \end{bmatrix} = \begin{bmatrix} \psi_1(\alpha) - \psi_1(\alpha+\beta) & -\psi_1(\alpha+\beta) \\ -\psi_1(\alpha+\beta) & \psi_1(\beta) - \psi_1(\alpha+\beta) \end{bmatrix} \tag{4}$$

Where $\psi_1(\alpha) = \frac{\partial^2 \ln\Gamma(\alpha)}{\partial\alpha^2}$ is the digamma, which can easily be calculated in Python using the function polygamma within scipy.special package.

Then the fisher scoring update is defined as $\theta^{(t+1)} = \theta^{(t)} + [I(\theta^{(t)})]^{-1} l'(\theta^{(t)})$

2.2 Euler's Method

Then once we have the values for the MLEs, the simulation can be run. Since the system is nonlinear, we are going to need to use some sort of numerical method to find the trajectories. For this simulation Euler's method for numerically approximating ordinary differential equations is used. Euler's method for equation 1 is defined as,

$$S(t+\Delta t) = S(t) + \Delta t(-\beta I(t)S(t)) \tag{5}$$
$$I(t+\Delta t) = I(t) + \Delta t(\beta I(t)S(t) - \gamma I(t)) \tag{6}$$
$$R(t+\Delta t) = R(t) + \Delta t(\gamma I(t)) \tag{7}$$

Where $\beta \sim Beta(\hat{\alpha_\beta}, \hat{\beta_\beta})$,$\gamma \sim Beta(\hat{\alpha_\gamma}, \hat{\beta_\gamma})$

This numerical method works perfectly for the simulation because it has time steps (Δt), that

will allow us to regenerate β and γ at each step. This would more realistically match real-world scenarios because the infection and recovery rates will vary from day to day as they do in real life. And by the nature of the SIR model these perturbations in the parameters will cause all of the trajectories to change.

Then if each run corresponds to a real-world scenario, we are going to do a Monte Carlo approach and run this simulation many times. At the end of each simulation, since we are interested in the total population that was infected over some time interval, those will be recorded. At the end of the simulation, all of those percentages will form a probability distribution of the total infected population percentage. This will give a better idea of the variation in the transmission of the disease.

3 Results

The data used for this simulation is the time series data on COVID-19 cases from Mexico between August 2020 and August 2021. After the likelihood estimators were found for β and γ the trajectories of equation 1 can be plotted using $E[\beta]$, and $E[\gamma]$. To do this Euler's method was used with beta and gamma set as their expected values (no random parameters), and the initial conditions $S_0 = 0.9, I_0 = 0.1, R_0 = 0.1$ over 365 time steps, which results in the plot in Figure 1.

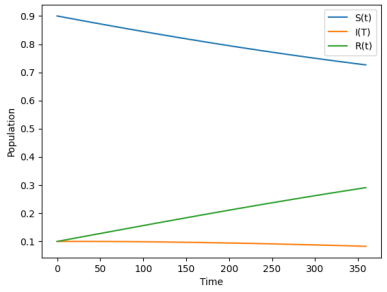

Figure 1: Expected Trajectories

The phase plot of the original system(no random parameters) based off of $E[\beta]$ and $E[\gamma]$ is shown in Figure 2.

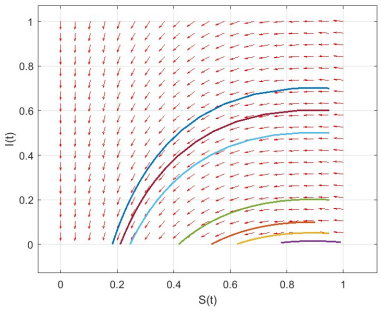

Figure 2: Phase Plot

From Figure 2 it is clear that when the random parameters generate values that are lower or higher

than their expected values, the trajectory of the infected individuals will change dramatically, even if the expected value was generated for the rest of time. The plot illustrates the influence of these random variables on the system, one really high infection rate changes the trajectory in a massive way, consistent with real-life transmission dynamics.

Then using randomly generated parameters at each time step, figure 3 shows examples from four different runs with the initial conditions $S_0 = 0.9, I_0 = 0.1, R_0 = 0.1$ over 365 time steps.

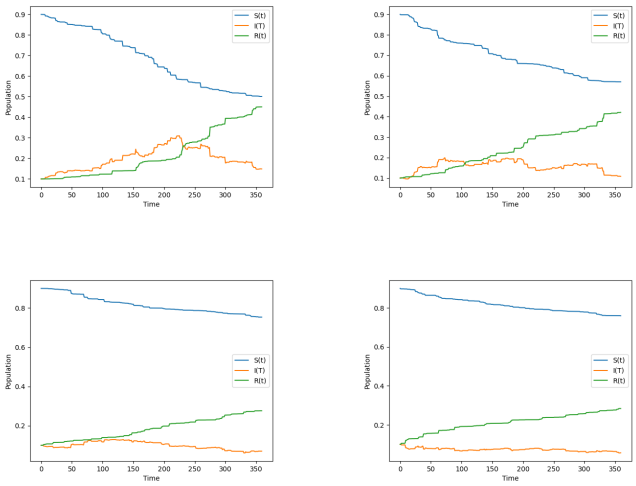

Figure 3: Randomly Generated Runs

This highlights the variability in trajectories. As you can see some will almost immediately move towards zero percent of the population being infected, and some will have big perturbations that cause the infection rate to grow very quickly.

Then using more realistic initial conditions to run the simulation of interest, $S_0 = 0.98, I_0 = 0.01, R_0 = 0.01$, the simulation is run 10000 times it gives the following distribution in Figure 4 of total infection percentage over a year period.

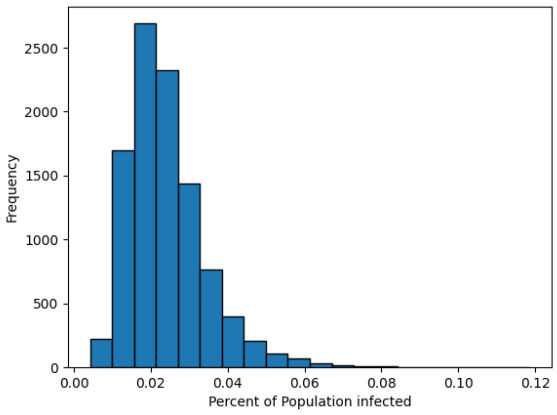

Figure 4: Distribution of Infected Population in One Year

This distribution had a mean percentage of 0.02411, which would imply that there are 3086080

infected individuals, which is close to the amount from the real-world data from Mexico at the end of the one year period ($\approx$ 3 million).

The same simulation over two and three years is shown in Figure 5. As you can see the shift in the distribution between one year and two years is larger than that of two and three years. This is because in the SIR model, the trajectories of $I(t)$ eventually converge to zero after some point.

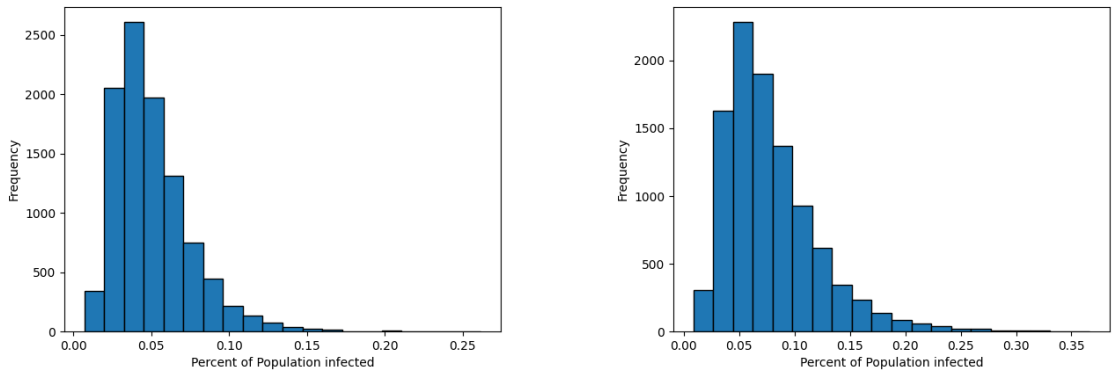

Figure 5: Distribution of Infected Population in Two and Three Years

4 Discussion

The variance in the trajectory behavior and the infected population is what I was hoping for as it seems realistic. There were some extreme cases (though uncommon), that would seem like they could be within the realm of possibility as well. The ranges in the infected population seem to be reasonable as well, and the expected values from each simulation are fairly close to the real-world data from Mexico. The simulation as a whole is a really strong tool for analyzing the possibilities of transmission.

The goal of this simulation was to show how dramatic perturbations in daily infection rates can change the overall percentage of the infected population significantly. The SIR model is very sensitive to increased infection rates, so when only using the deterministic model with real-world data, you really aren't getting to do an in-depth analysis of transmission. This is because there are normally spikes and dips in infection rates, which appear to happen randomly. While the deterministic model should be a close approximation most of the time, it doesn't factor in the case that there is a string of abnormal rates, causing the trajectory of the disease transmission to change dramatically.

While in this report only the percentage of the population infected was explored, there is a lot of other analysis that can be done with this simulation. Such as how many time steps would it take for the infection percentage to fall below a certain amount (predicting the lifespan of the disease), or calculating a confidence interval of the total infected percentage. Similar methods can also be used on other epidemic models for stronger analysis.

5 Appendix

All of the project was done in Python,
Here are the Packages used,

```
import numpy as np
import scipy
from scipy.special import betaln
from scipy.special import psi, polygamma
import pandas as pd
import matplotlib.pyplot as plt
from scipy.special import beta
from scipy.special import betaln
from scipy.stats import beta
from scipy.special import betaln, digamma
```

Here is the code for loading in and processing the data,

```
# Load the data
file_path = r'C:\Users\chado\Downloads\time-series-19-covid-combined.csv'
data = pd.read_csv(file_path)

#Process Data
def preprocessdata(data):
    data = np.array(data)
    data = data[data != 0]                #Get rid of zeros
    return data

rows = data.iloc[150538:150705]              #Lines for the time interval

confirmed_cases = rows['Confirmed'].values  #Used number of Confirmed cases
confirmed = confirmed_cases.tolist()
confirmeddiff = [confirmed[i] - confirmed[i - 1] for i in range(1, len(confirmed))]
confirmeddiff = [x / 128000000 for x in confirmeddiff]        #Divide by the total population
confirmed2 = preprocessdata(confirmeddiff)           #Change to numpy array

recovered_cases = rows['Recovered'].values          #Used number of Recovered cases
recovered = recovered_cases.tolist()
recovereddiff = [recovered[i] - recovered[i - 1] for i in range(1, len(recovered))]
recovereddiff = [x / 128000000 for x in recovereddiff]         #Divide by the total population
recovered2 = preprocessdata(recovereddiff)           #Change to numpy array
```

Here is the code implementing fisher scoring

```
#Find MLES
def fisher_scoring(data, tol=0.001, max_iter=100):
    update = np.array([1, 1])      #Initial update
    for i in range(max_iter):
        score_alpha = np.sum(np.log(data) - digamma(update[0]) + digamma(update[0] + update[1]))
        score_beta = np.sum(np.log(1 - data) - digamma(update[1]) + digamma(update[0] +update[1]))
```

```
        score = np.array([score_alpha, score_beta])                 #Find Score Funtion

        info_alpha = -polygamma(1, update[0]) + polygamma(1, update[0] + update[1])
        info_beta = -polygamma(1, update[1]) + polygamma(1, update[0] + update[1])
        info_alphabeta = -polygamma(1, update[0] + update[1])
        info = np.array([[info_alpha, info_alphabeta],[info_alphabeta, info_beta]]) #Find information m

        old = update
        update = update + np.linalg.inv(info)@score     #Update fisher scoring
        if np.linalg.norm(abs(update-old)) < tol:       #Check for convergance
            return update[0], update[1]
    return


alpha_confirmed, beta_confirmed = fisher_scoring(confirmed2)
print(alpha_confirmed, beta_confirmed)
alpha_recovered, beta_recovered = fisher_scoring(recovered2)
print(alpha_recovered, beta_recovered)
```

Here is the code I was using to plot random trajectories using Euler's method,

```
# Initial conditions
S0 = 0.98
I0 = 0.01
R0 = 0.01

# Time parameters
T = 365 # total time steps
dt = 1  # time step

S = np.zeros(T)
I = np.zeros(T)
R = np.zeros(T)
S[0], I[0], R[0] = S0, I0, R0

#Eulers Method
for t in range(1, T):
    beta1 = np.random.beta(alpha_confirmed, beta_confirmed)    #randomly generate beta
    gamma1 = np.random.beta(alpha_recovered, beta_recovered)   #randomly generate gamma

    S[t] = S[t-1] + (-beta1 * S[t-1] * I[t-1] ) * dt    #equation 5
    I[t] = I[t-1] + (beta1 * S[t-1] * I[t-1] - gamma1 * I[t-1] ) * dt   #equation 6
    R[t] = R[t-1] + (gamma1 * I[t-1] ) * dt    #equation 7

#Plot trajectories
plt.plot(S, label="S(t)")
plt.plot(I, label="I(T)")
plt.plot(R, label="R(t)")
plt.xlabel("Time")
plt.ylabel("Population")
plt.legend()
plt.show()
```

This is the code that I wrote to perform the simulation and plot the percentages of infected people in a histogram.

```
# Simulate and make a histogram
percentage = []
for i in range(1, 100):
    S = np.zeros(T)
    I = np.zeros(T)
    R = np.zeros(T)
    S[0], I[0], R[0] = S0, I0, R0
    for t in range(1, T):
        beta1 = np.random.beta(alpha_confirmed, beta_confirmed)    #randomly generate beta
        gamma1 = np.random.beta(alpha_recovered, beta_recovered)   #randomly generate gamma

        S[t] = S[t-1] + (-beta1 * S[t-1] * I[t-1] ) * dt    #equation 5
        I[t] = I[t-1] + (beta1 * S[t-1] * I[t-1] - gamma1 * I[t-1] ) * dt   #equation 6
        R[t] = R[t-1] + (gamma1 * I[t-1] ) * dt    #equation 7
        percentage.append(0.98 - min(S))   #Find the amount infected from t0

plt.hist(percentage, bins=20, color='blue')         #Plot histogram
plt.xlabel('Percentage of Population Infected')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()
```