Technical University of Denmark

02443 - Stochastic Simulation

Project 2: Simulation of epidemics

Gunnhildur Katrín Erlendsdóttir - s212510

Helga Dís Halldórsdóttir - s212457

David Miles-Skov - s204755

Patrick Jensen Martins - s204748

Professor:

Bo Friis Nielsen

Contents

1 Contributions

The work was distributed equally between all group members.

2 Introduction

In this project, the focus will be on disease modeling, used for simulating and analysing epidemics. We aim to demonstrate and discuss the simulation of outbreaks, as well as explore measures to reduce their spread. Our primary focus will be on the Susceptible, Infected, Recovered (SIR) model, a well-known approach for studying disease transmission. Throughout our investigation, we enhance the complexity of the model to gain deeper insights into disease modeling. In order to do that we will look at the SEIR model which adds another component E (Exposed) to the model. We also investigate the effects of vaccines with different effectiveness.

3 Part 1: Basic Modeling

In this first part, our aim is to provide an introductory overview of the SIR (Susceptible-Infectious-Recovered) model, focusing on its theoretical foundations and exploring various parameter values, which enable us to examine a wide range of scenarios.

To illustrate the application of the SIR model, we will initially consider a simple example of simulating the spread of flu using both the ordinary differential equation (ODE) approach and a stochastic approach. Furthermore, we will extend our analysis to a more complex scenario by simulating the spread of COVID-19, specifically for Brazil and Japan.

3.1 The SIR Model

The SIR model is a mathematical model used to understand the spread of infectious diseases in a population. It divides the population into three compartments, Susceptible (S), Infected (I) and Recovered (R). This model assumes that individuals can only belong to one of these compartments at any given time and that the population is well-mixed, meaning that individuals have an equal chance of coming into contact with anyone in the population.

The equations for the SIR model are as follows:

$$\frac{dS}{dt} = -\frac{\beta SI}{N}$$

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

$$\frac{dR}{dt} = \gamma I$$

In these equations, S represents the number of susceptible individuals, I represents the number of infected individuals, R represents the number of recovered individuals, and N represents the total population size.the parameter $\beta > 0$ signifies the transmission rate of the disease, indicating how easily the infection spreads from infected individuals to susceptible ones. On the other hand, $\gamma > 0$ represents the recovery rate, reflecting the rate at which infected individuals recover or transition to the recovered state. The duration of an infection is given by $D = \frac{\gamma}{1}$, which simplifies to $D = \gamma$.

The term βSI represents the number of new infected individuals per unit of time, reflecting the rate at which susceptible individuals become infected. In the specific problem formulation mentioned above, this implies that the disease always dies out as time approaches infinity ($t \to \infty$). Consequently, the basic model suggests that the disease will not exhibit cyclical behavior, indicating the absence of recurring outbreaks [3].

3.1.1 Deterministic SIR Model using Ordinary Differential Equations

A simple example of a SIR model using differential equations was implemented. This scenario shows a disease epidemic where the total population was 1000. To initiate the simulation, we assume that at time t=0, there are 10 infected individuals (I=10) and the remaining individuals are considered susceptible (S = N-10). The simulation was allowed to run for a duration of 100 days, capturing the progression of the epidemic over time. Here the parameters used are $\beta = 0.6$, $\gamma = 0.2$ and $N = 1000$ and the results are shown in figure 1.

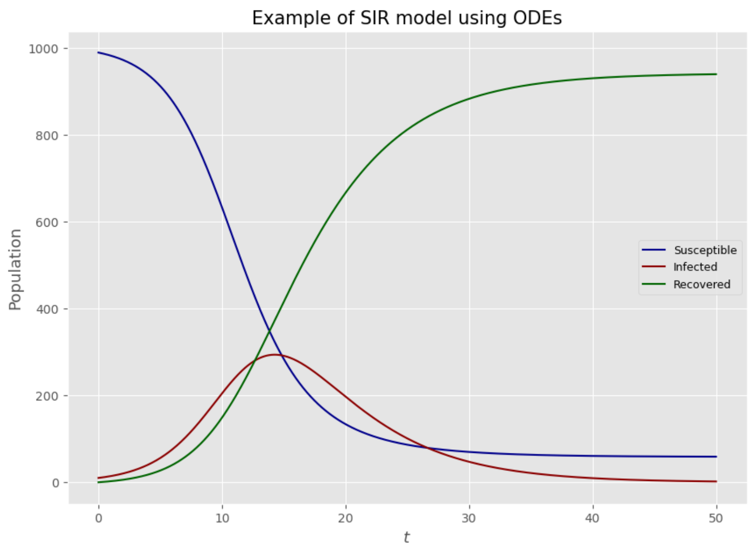


Figure 1: Simple example of a SIR model using ODEs

From the figure, we can see that most people were infected by the disease at around t = 35 so at that point the disease reached its peek. At around t = 65, almost all of the population has recovered.

3.1.2 Stochastic SIR Model

Now the same example is implemented using a stochastic simulation. We utilize an exponential distribution to represent the proportions of infected and recovered individuals. The parameters of this distribution correspond to the infection rate and the recovery rate in the above example. Subsequently, we generate the numbers of infected and recovered individuals by sampling from a binomial distribution. The probability used in this distribution corresponds to the fractions mentioned earlier. The result of the SIR model example using a stochastic simulation is shown in figure 2.

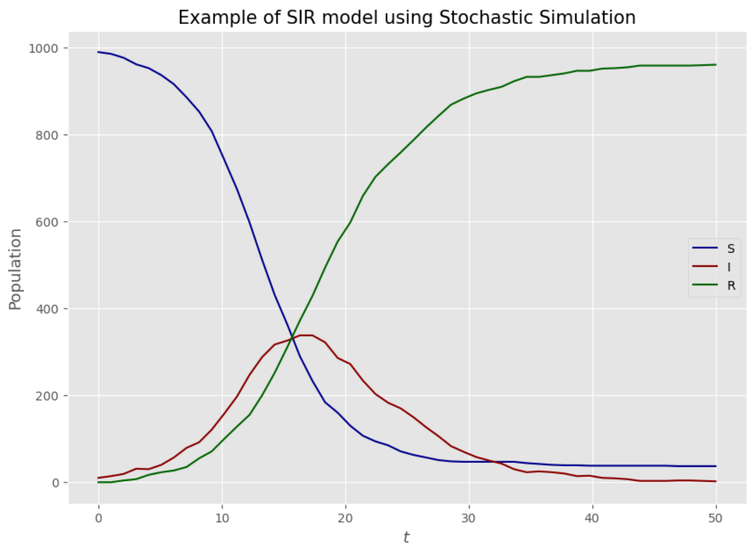
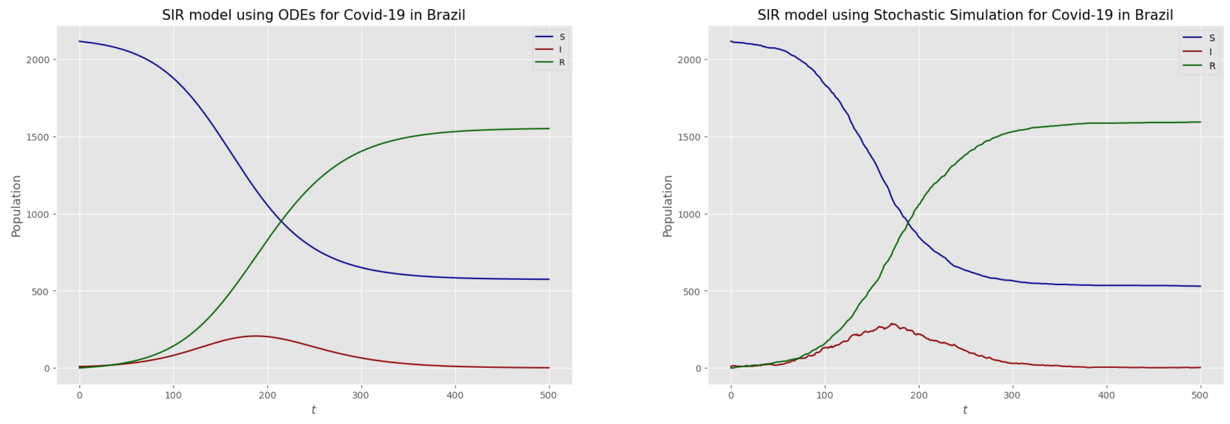


Figure 2: SIR model using Stochastic simulation

In figure 2 we see similar behavior as in figure 1. The disease arrives at the same time as in the ODE example and then eventually disappears. It is to be expected in a simple example like this that both methods show similar results.

3.2 Modelling Covid-19 in Brazil and Japan using the SIR model

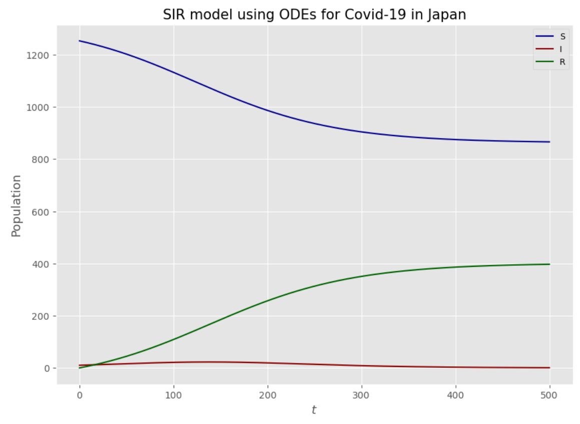
Now, let's examine a real-world example of COVID-19 in Japan and Brazil. In the case of Brazil, the population size at the time of analysis was approximately 212.6 million individuals. However for the purposes of applying the SIR model, we will only use a fraction of the whole population as not all of the population was infected by Covid-19, therefore $N = 2.126$. The estimated parameter values used were $\beta = 0.0756$ and $\gamma = 0.0423$ [5]. The basic reproduction number (R_0) for the disease was calculated as $R_0 = \beta/\gamma = 1.789$. Consequently, the simulation was conducted for a duration of 500 days. The results for the SIR model are shown figure 3. Figure a) shows the deterministic SIR model and figure b) shows the SIR model using stochastic simulation for only one simulation.



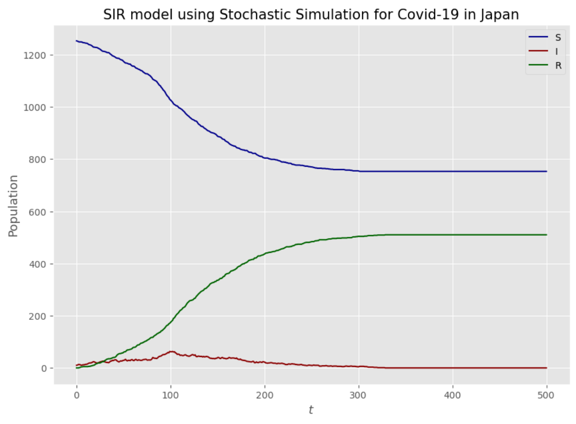
((a)) Deterministic SIR model ((b)) Stochastic SIR model

Figure 3: SIR model of Covid-19 in Brazil with $\beta = 0.0756$ and $\gamma = 0.0423$

Japan has a population of around 126.3 million people, and again only a proportion of the whole population was used in the SIR model ($N = 1.263$). The parameter values used were $\beta = 0.0800$ and $\gamma = 0.0681$ [5]. The basic reproduction number (R_0) for the disease was calculated as $R_0 = \beta/\gamma = 1.173$. The time used spanned a period of 500 days. Figure 4 shows the results of the SIR model for Covid-19 in Japan, both for the deterministic model and the stochastic model.



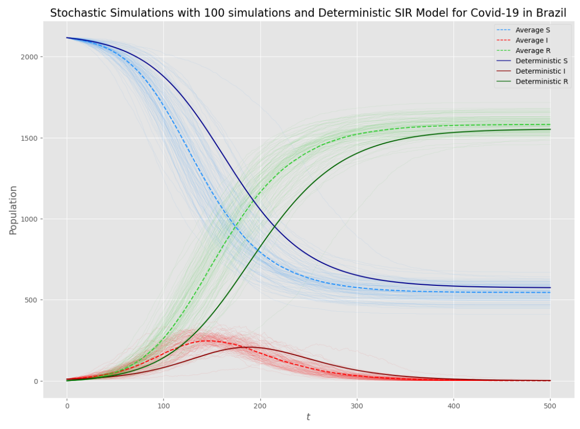
((a)) Deterministic SIR model



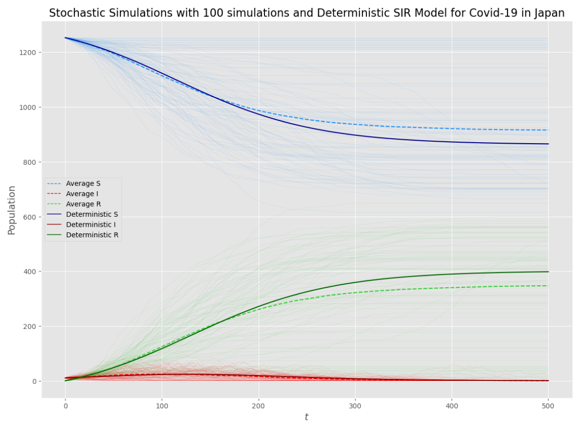
((b)) Stochastic SIR model

Figure 4: SIR model of Covid-19 in Japan with $\beta = 0.0800$ and $\gamma = 0.0681$

Figure 3 (Brazil) shows that the number of infected individuals peeks at after around 190 days, there it reaches around 10% of the population, which seems reasonable. The number of susceptible people seems to significantly decrease around the time that the peeks happens. Figure 4 (Japan) however shows that only a very small part of the population is infected, so it looks like Japan had minimal effect from Covid-19, which is not very reasonable. As Japan has a R_0 value close to 1, it is likely the disease will disappear eventually.



((a)) Deterministic SIR model for Covid-19 in Brazil



((b)) Stochastic SIR model for Covid-19 in Japan

Figure 5: Comparing stochastic SIR model for Covid-19 in Brazil and Japan

Figure 5 shows the stochastic SIR model for 100 simulations. Increasing the simulations should give more accurate results. The error rates were calculated for Covid-19 in Brazil to see the difference of increasing the number of simulations compared to the analytical results. The error rates were calculated as the mean absolute percentage difference between the two sets of values. For a single simulation the error rate was given as:

$$\frac{|sim_1 - analytical|}{analytical} = 0.338$$

For 1000 simulations the error rate was given as:

$$\frac{|sim_{1000} - analytical|}{analytical} = 0.246$$

Showing that by increasing the simulations we achieve better results.

3.3 Likelihood of a disease disappearing

The likelihood of a disease dying out is determined by the basic reproduction number R0, which reveals the average number of secondary infections caused by an infected person. If R0 is much greater than 1, then the likelihood of the disease dying is very.

The SIR model for COVID-19 in Japan has an R0 equal to 1.173. Even though this number is larger than 1, it is not significantly larger, thus the COVID-19 disease will not disappear immediately, however it will not cause a large scale epidemic.

3.4 How does a disease exhibit cyclical behaviour?

In order to check if a disease exhibits cyclical behaviour we should inspect the changes of effective reproduction number, R1, which is obtained by multiplying the basic reproduction number, R0, by the fraction of the population that is susceptible. This will tell us the number of new infections caused by a single person in a population with and without immunity. If there is any period where R1 increases significantly, then we can classify the disease as exhibiting cyclical behaviour.

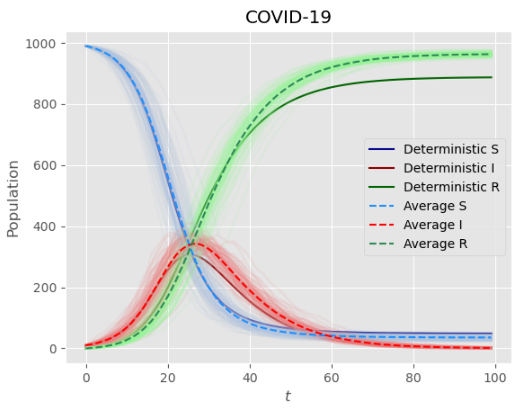
3.5 Can a highly deadly disease extinguish the population?

A highly deadly disease could extinguish a whole susceptible population if it presents an extremely high infection rate and a death rate close to 1. In order to model and demonstrate this, one could add another state D to the basic SIR model, in which people transition from the infectious state to the dead state according to a specific death rate for the disease.

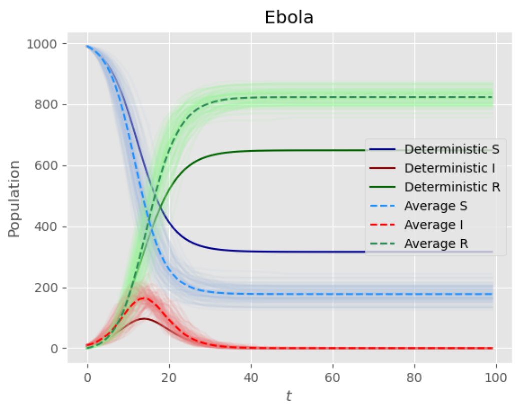
The death of people will alter the value of β (infection rate), because there is a chance that part of the population die soon enough to not infect anybody else. Therefore, if we investigate the evolution of the basic reproduction number, R0, throughout time, if it decreases rapidly before eradicating the whole susceptible population, then there is a low likelihood for the disease to extinguish the whole population.

3.6 Further Modelling of Various Diseases

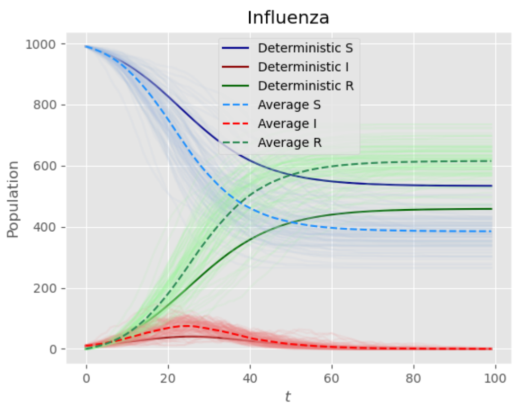
After finding suitable values of β and γ for various diseases, graphs were made comparing the deterministic (ODE) model and the stochastic model (7). The average stochastic S, I and R was calculated after 50 simulations.



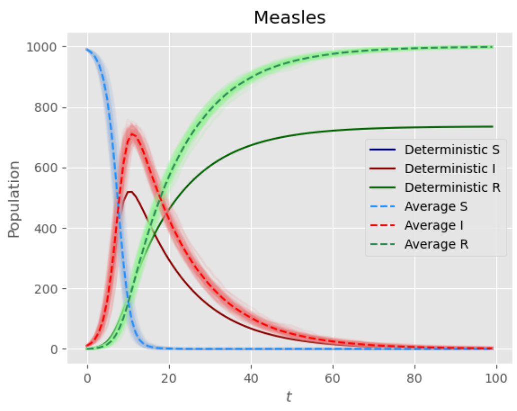
((a)) COVID-19: $\beta : 0.328,\ \gamma : 0.1$ [4]



((b)) Ebola: $\beta : 0.71,\ \gamma : 0.4152$ [2]



((a)) Influenza: $\beta : 0.4,\ \gamma : 0.3$ [1]



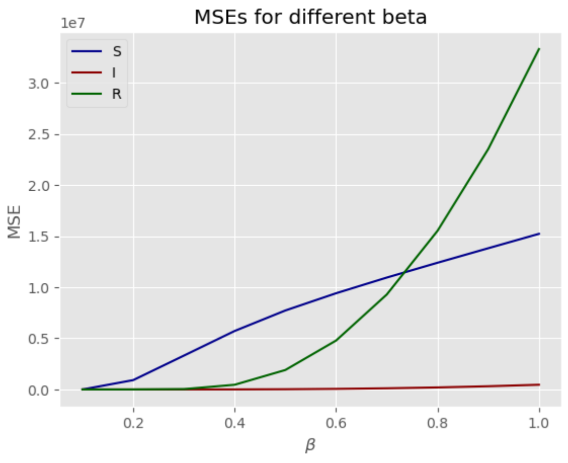
((b)) Measles $\beta : 0.8571,\ \gamma : 0.0714$ [7]

Figure 7: Comparison of Deterministic and Stochastic SIR model for a variety of diseases

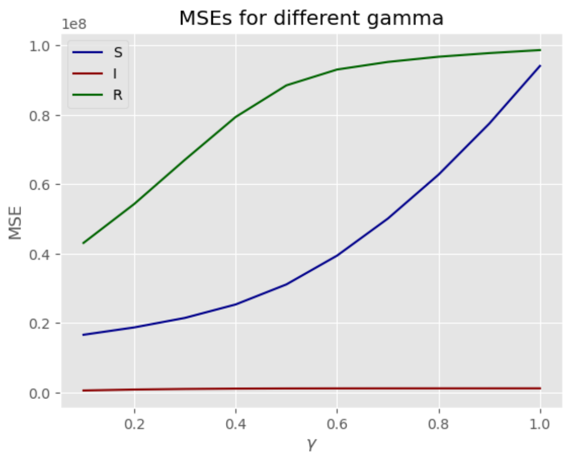
Despite being initially very similar, the two models eventually asymptote at different values. This is most apparent for long term prediction of R. When examining the models and taking the values of β and γ into consideration, it is clear that a relationship between the similarity of the two models and the values of β and γ exists.

3.6.1 Investigating the Effect of Parameters on the Similarity of the Two Models

In order to assess the effect that each parameter has on the similarity of the two models, each parameter was adjusted whilst keeping the others constant. The *Mean Squared Error* was then calculated for each parameter value, between an average of 50 stochastic simulations and the deterministic ODE-based model.

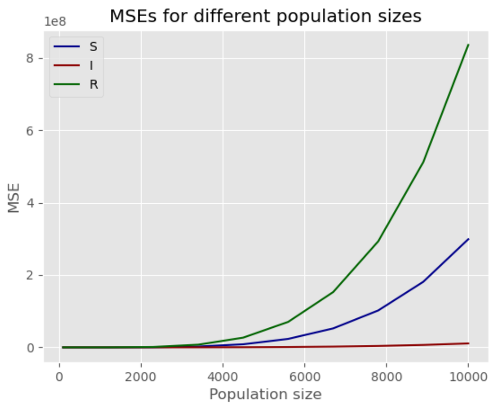


((a)) MSE vs β at $\gamma = 0.2$

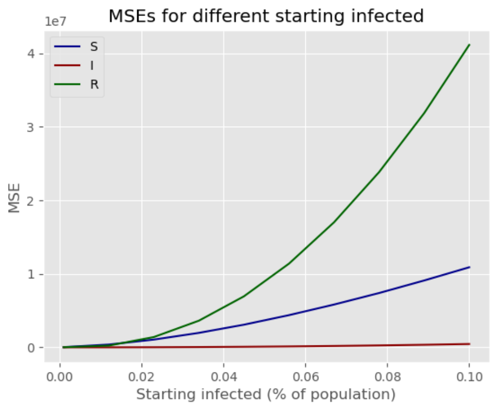


((b)) MSE vs γ at $\beta = 0.5$

Figure 8: Comparison of MSE between the stochastic and determinisitc model for varying βs and γs. Population: 1000, Starting infected: 10



((a)) MSE vs population size at $\gamma = 0.2$ and $\beta = 0.4$, initial infected: 1% of population



((b)) MSE vs initial infected population at $\gamma = 0.2$ and $\beta = 0.4$.

Figure 9: Comparison of MSE between the stochastic and determinisitc model for varying starting populations and initial infected proportions.

There is a clear positive relationship between each of the parameters and the calculated MSE between the two models. The stochastic implementation of the model, the probability of infection and recovery are both proportional to β and γ respectively. The number of new infections and recoveries are modelled using a Binomial distribution. Thus, when β and γ are increased, this also increases variability and randomness in the stochastic model, resulting in a greater deviation from the deterministic model. The same is also true for an increase in population size/initial infected population.

4 Part 2: Further studies

In the second part, we will explore ways to make the SIR model more complex in order to gain deeper insights into disease modeling. We will incorporate the SEIR model as well as looking at the effect of vaccination for various diseases.

4.1 The SEIR model

The SEIR model is an extension of the SIR (Susceptible-Infectious-Recovered) model, commonly used in epidemiology to study the spread of infectious diseases. The SEIR model includes an additional compartment called "Exposed" to represent individuals are infected but are not yet able to transmit the disease.

The equations for the deterministic SEIR model are as follows:

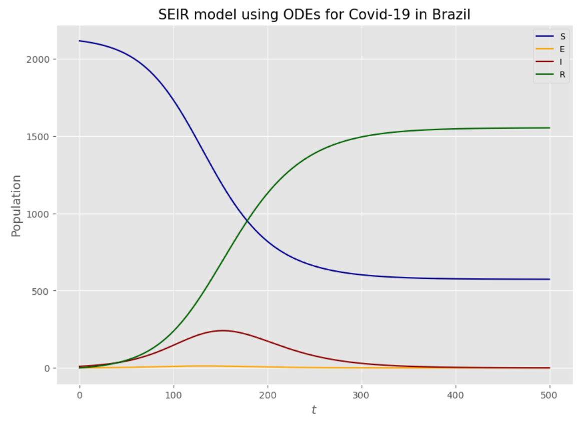
$$\begin{aligned}\frac{dS}{dt} &= -\beta \cdot S \cdot I \\ \frac{dE}{dt} &= \beta \cdot S \cdot I - \sigma \cdot E \\ \frac{dI}{dt} &= \sigma \cdot E - \gamma \cdot I \\ \frac{dR}{dt} &= \gamma \cdot I\end{aligned}$$

In the SEIR model we have an additional parameter σ which represents the rate at which exposed individuals become infectious.

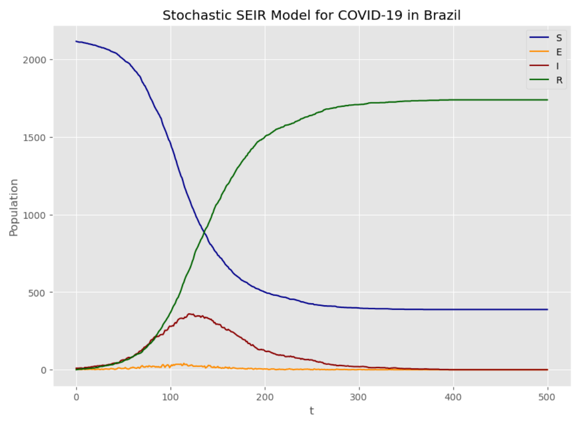
The stochastic SEIR model incorporates randomness in disease transmission and progression. It models the compartments $S(t)$, $E(t)$, $I(t)$, and $R(t)$ continuously over time. Events occur within the system, such as transitions between compartments or changes in disease state, and are associated with specific times. The time until the next event follows an exponential distribution, representing the waiting time. Equations $B(t) = E(t) + I(t) + R(t)$ and $C(t) = I(t) + R(t)$ capture the relationships between compartments. The stochastic SEIR model provides a realistic framework for studying disease dynamics by accounting for inherent variability and uncertainty [3].

4.1.1 Modelling COVID-19 in Brazil using the SEIR model

A deterministic SEIR model was utilized to simulate the spread of COVID-19 in Brazil. The initial values at time zero were set to $(S_0, E_0, I_0, R_0) = (2116, 1, 10, 0)$. The model was then simulated once for maximum time of $t_{max} = 500$ with $\beta = 0.0756$, $\gamma = 0.0423$ [5] and the average incubation period for COVID-19 is between 5 to 7 days on average sigma was set as $\sigma = 1/5$ [6]. The results for the deterministic model and for a single simulation stochastic model are shown in figure 10.



((a)) Deterministic SEIR model



((b)) Stochastic SEIR model

Figure 10: Models for Covid in Brazil with $\beta = 0.0756$, $\gamma = 0.0423$ and $\sigma = 0.2$

At around time $t = 300$ the disease reaches a point where $I(t) = 0$ and $E(t) = 0$, indicating that the disease has completely died out and no further changes occur in the process. When comparing the deterministic model to the stochastic model, they appear to exhibit a comparable trajectory. It is worth noticing that $E < I$ for the whole time period, which could suggests that the disease progression from the exposed state to the infectious state is relatively rapid.

Next, a stochastic simulation model was computed that created 100 stochastic simulations and plotted with the deterministic SEIR model. This was used again for the case of Brazil. Then the average of the 100 simulations was plotted such that it can be compared to the deterministic values These results are presented in figure 11.

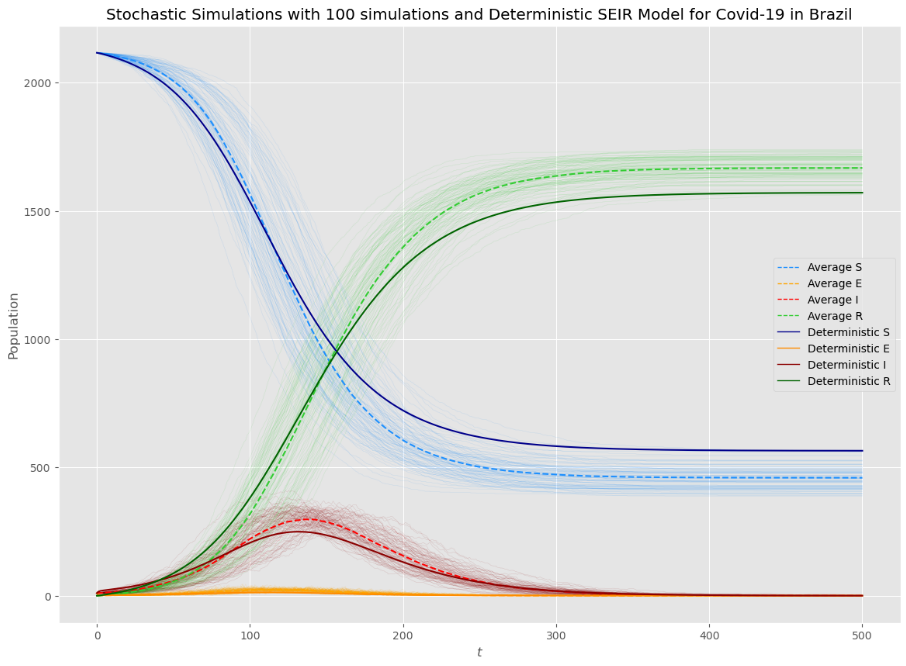


Figure 11: Stochastic SEIR model with 100 simulations and $\beta = 0.0756$, $\gamma = 0.0423$ and $\sigma = 0.2$

Observing the plot, it becomes evident that the average of the stochastic simulations quite closely aligns with the deterministic values. This suggests that the stochastic nature of the model does not introduce significant deviations from the deterministic predictions.

4.2 Effect of Vaccination for Vaccines of Varying Effectiveness

In this section we aim to simulate and analyse the effect of vaccinations with different effectiveness from 2 diseases using the SIRV model. First we simulate the effect of two vaccines for Ebola, with 90% and 40% effectiveness, respectively. The β and γ values of $\frac{1}{3}$ and $\frac{1}{7}$, respectively, were determined according to a source that mentioned that the basic reproduction number for Ebola ranges between 1.4 and 2.8.

This model is an extension of the SIR, by including a fourth compartment for grouping Vaccinated people. Vaccinated people are randomly picked from the Susceptible population according to a vaccination rate, which we determined to be equal to 0.02 and is always constant. From the vaccinated population, only a percentage of it ends up acquiring immunity, according to the effectiveness of the vaccine, and should be grouped into the Vaccinated compartment. The group that is not immune to the disease after vaccination remains in the Susceptible population. In the following graphs the dotted lines represent the deterministic simulations and the continuous lines represents the stochastic simulations.

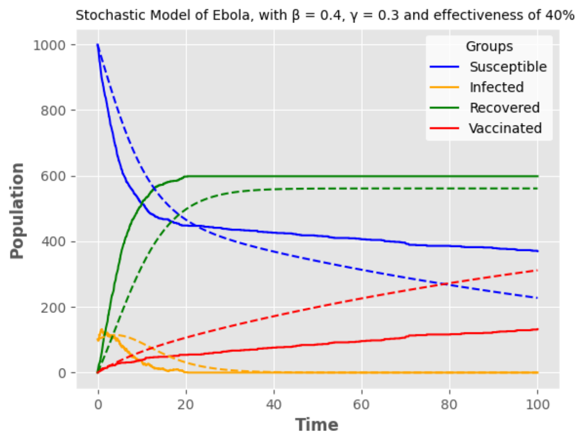


Figure 12: 40% effectiveness

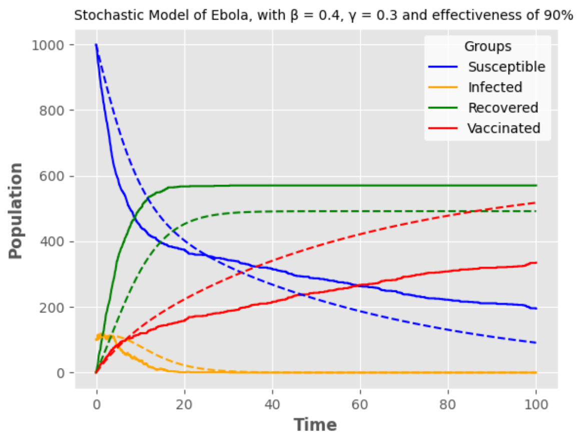


Figure 13: 90% effectiveness

As expected, the number of susceptible people is significantly lower (approximately 2 times lower) for the 90% effective vaccine against the 40% effective one, after 100 days.

Next, we simulate the effect of two vaccines for Influenza, with 90% and 40% effectiveness, respectively.

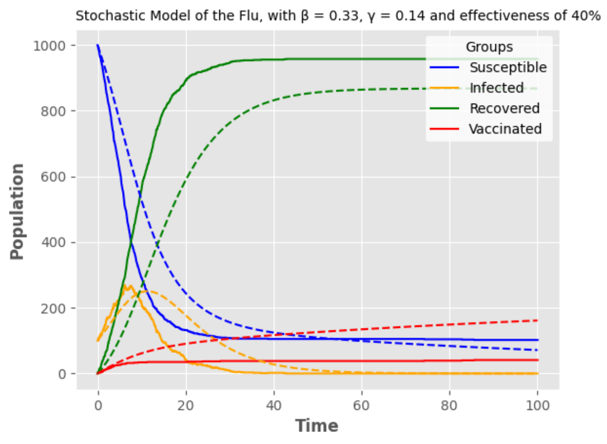


Figure 14: 40% effectiveness

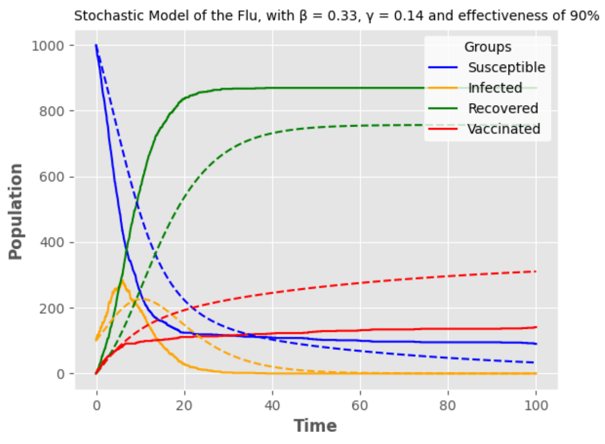


Figure 15: 90% effectiveness

To further inspect the differences, we plot the evolution of the susceptible population for both diseases, with different vaccine effectiveness.

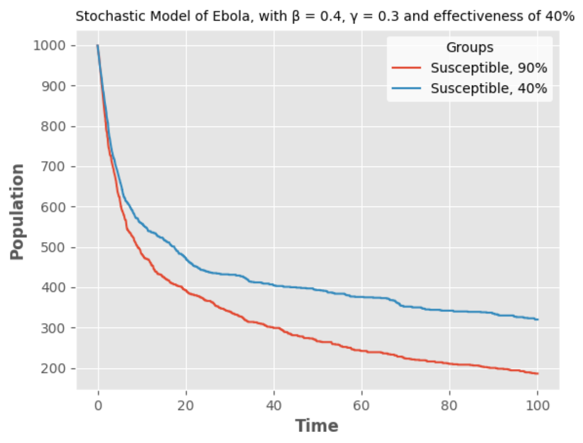


Figure 16: Ebola

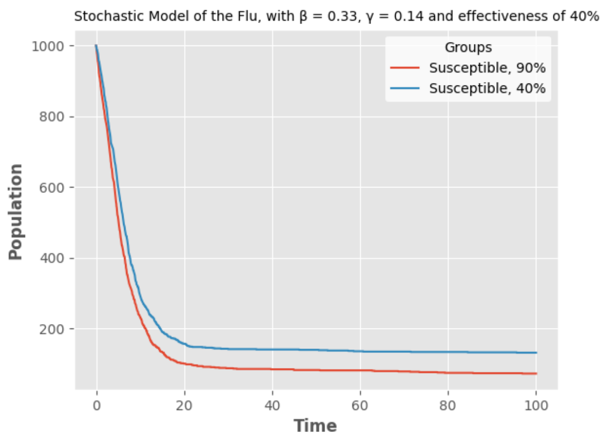


Figure 17: Flu

The evolution of the susceptible population was modelled from two different stochastic simulated samples. Due to the variation of the outputs from stochastic simulations, we collected samples of the susceptible population after 100 days for 50 different simulations for the vaccine effectiveness of 40% and 90% and compared the means to verify the effect of the vaccines.

Table 1: xxx

Disease	**Effectiveness**	**Susceptible**	
		Mean	**Std.**
Ebola	40%	317.94	24.83
	90%	171.48	10.81
Flu	40%	115.40	19.38
	90%	86.32	11.76

The results reflect that there is a significant reduction of the susceptible population in the 100'th day of the experiment, with a reduction of 46% for Ebola and 25% for the Flu by using a vaccine with an effectiveness of 90% compared to 40%.

The model that was implemented only includes people with immunity in the vaccinated group and the group of people who got the vaccination but didn't develop immunity remain in the susceptible population. This assumption opens up the possibility for a vaccinated person who isn't immune to the disease to be vaccinated again and either build immunity after the second shot or remain as a susceptible person. This could be addressed by splitting the susceptible population into two groups: vaccinated and unvaccinated; and the vaccinated people would only be drawn from the unvaccinated group of the susceptible population.

5 Conclusion

During this project we learned that it is possible to simulate and inspect the behaviour of various diseases by modelling different states that are implemented according to specific assumptions. It is possible to create more realistic models by adding more states to the basic SIR model. The most challenging part of creating a reliable, rather complex, SIR model for a disease is in finding appropriate infection rates, recovery rates, and other rates that describe the evolution of states, and stochastically generate new rates for every time interval. This approach would ensure a more realistic model for the behaviour of a disease, because in reality infection and recovery rates, for example, constantly vary.

6 Appendix

Part 1

```
import matplotlib.pylab as plt
import math
import numpy as np
from scipy import stats
from scipy.integrate import odeint
```

```
# differential equatinons
def diff(SIR, t, beta, gamma, N, alpha):
    dsdt = - (beta * SIR[0] * SIR[2])/N
    dedt = (beta * SIR[0] * SIR[2])/N - alpha*SIR[1]
    didt = alpha*SIR[1] - gamma * SIR[2]
    drdt = gamma * SIR[2]
    dSIRdt = [dsdt,dedt,didt, drdt]
    return dSIRdt



def simulate_sir_model(beta, gamma, N, alpha, days):
    S = N - 10
    I = 10
    R = 0
    E = 1
    day = days
    # initial conditions
    SIR_init = (S, E, I, R)
    # time points
    t = np.linspace(0, day, 1000)
    # solve ODE
    SIR = odeint(diff, SIR_init, t, args=(beta, gamma, N,
   alpha))
    return SIR, t
```

```
beta = 0.6 # infection rate
gamma = 0.2 # recovery rate
alpha = 5.5
N = 1000
days = 50
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
plt.style.use('ggplot')
plt.figure(figsize=(10,7))
plt.plot(t, SIR[:, 0], color='darkblue', label = '
   Susceptible')
plt.plot(t, SIR[:, 2], color='darkred', label = 'Infected')
plt.plot(t, SIR[:, 3], color='darkgreen', label = 'Recovered
   ')

plt.legend(fontsize=9)

plt.xlabel(r"$t$", fontsize=13)
plt.ylabel('Population', fontsize=13)
plt.title('Example of SIR model using ODEs', fontsize=15)
plt.grid(True)

plt.show()
```

```
def simulate_stochastic(population, initial_infected,
   max_time, beta, gamma):
    """
    Stochastic simulation of disease progression
    - Population
    - initial number of infected
    - time over which to model

    Returns:
    - S, I, R
```

```
     """

     # iota = 0.01 - external infection rate
     t = np.linspace(0,max_time,max_time)
     S, I, R = [population - initial_infected], [
    initial_infected], [0]

     for _ in t[1::]:

         l = beta*(I[-1])/population # some probability?

         # Success rates
         ifrac = 1.0 - np.exp(-l) # Probability of infection
         rfrac = 1.0 - np.exp(-gamma) # Probability of
    recovery

         infection = np.random.binomial(S[-1],ifrac) # Number
     of new infected
         recovery = np.random.binomial(I[-1],rfrac) # Number
    of new recovered

         S.append(S[-1]-infection)
         I.append(I[-1]+infection-recovery)
         R.append(R[-1]+recovery)
         #Re.append(self.getRe(S[-1], population))

     return S, I, R
```

```
beta = 0.6
gamma = 0.2
population = 1000
max_time = 50
initial_infected = 10
```

```
S, I, R = simulate_stochastic(population, initial_infected,
   max_time, beta, gamma)
```

```
plt.figure(figsize=(10,7))
x = np.linspace(0, max_time, int(max_time/1))
plt.plot(x, S, color="darkblue", label="S")
plt.plot(x, I, color="darkred", label="I")
plt.plot(x, R, color="darkgreen", label="R")

plt.grid()
plt.legend()
plt.title("Example of SIR model using Stochastic Simulation"
   , fontsize=15)
plt.ylabel("Population", fontsize=13)
plt.xlabel(r"$t$", fontsize=13)
plt.grid(True)

plt.show()
```

```
beta = 0.6 # infection rate
gamma = 0.2 # recovery rate
alpha = 5.5
N = 1000
days = 100
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
num_sims = 100

plt.figure(figsize=(14,10))

for i in range(num_sims):
    beta = 0.6
    gamma = 0.2
    population = 1000
    max_time = 100
    initial_infected = 10
    S, I, R = simulate_stochastic(population,
   initial_infected, max_time, beta, gamma)
    x = np.linspace(0, max_time, int(max_time/1))
    S_sim = np.zeros([num_sims, len(S)])
    I_sim = np.zeros([num_sims, len(I)])
    R_sim = np.zeros([num_sims, len(R)])

    if i == 0:
        plt.plot(x, S, color='darkblue', alpha=0.2, lw=0.25,
    label = 'S(t) — simulated')
        plt.plot(x, I, color='darkred', alpha=0.2, lw=0.25,
   label = 'I(t) — simulated')
        plt.plot(x, R, color='darkgreen', alpha=0.2, lw
   =0.25, label = 'R(t) — simulated')

        S_sim[i] = S
        I_sim[i] = I
        R_sim[i] = R
    else:
        plt.plot(x, S, color='darkblue', alpha=0.2, lw=0.25)
        plt.plot(x, I, color='darkred', alpha=0.2, lw=0.25)
        plt.plot(x, R, color='darkgreen', alpha=0.2, lw
   =0.25)

        S_sim[i] = S
        I_sim[i] = I
        R_sim[i] = R
    T_sim = x

plt.plot(t, SIR[:, 0], '—', color='black', label = 'S(t) —
    analytical')
plt.plot(t, SIR[:, 2], ':', color='black', label = 'I(t) -
    analytical')
plt.plot(t, SIR[:, 3], '--', color='black', label = 'R(t) -
    analytical')

legend = plt.legend()
for l in legend.get_lines():
    l.set_alpha(1)
    l.set_linewidth(1.0)

plt.xlabel('Time (t)')
plt.ylabel('Population (N)')
plt.title('1000 stochastic simulations')

plt.show()
```

```
beta = 0.0756
gamma = 0.0423
alpha = 1/5.1
N = 212600000/100000
days = 500
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
plt.figure(figsize=(10,7))
plt.plot(t, SIR[:, 0], color='darkblue', label = 'S')
plt.plot(t, SIR[:, 2], color='darkred', label = 'I')
plt.plot(t, SIR[:, 3], color='darkgreen', label = 'R')

plt.legend(fontsize=9)

plt.xlabel(r"$t$", fontsize=13)
plt.ylabel('Population', fontsize=13)
plt.title('SIR model using ODEs for Covid−19 in Brazil',
   fontsize=15)
plt.grid(True)

plt.show()
```

```
beta = 0.0756
gamma = 0.0423
population = 212600000/100000
max_time = 500
initial_infected = 10
S, I, R = simulate_stochastic(population, initial_infected,
   max_time, beta, gamma)
```

```
plt.figure(figsize=(10,7))
x = np.linspace(0, max_time, int(max_time/1))
plt.plot(x, S, color="darkblue", label="S")
plt.plot(x, I, color="darkred", label="I")
plt.plot(x, R, color="darkgreen", label="R")

plt.grid()
plt.legend()
plt.title('SIR model using Stochastic Simulation for Covid
   −19 in Brazil', fontsize=15)
plt.ylabel("Population",fontsize=13)
plt.xlabel(r"$t$",fontsize=13)
plt.grid(True)

plt.show()
```

```
# Set the parameters and initial conditions for the models
beta = 0.0756
gamma = 0.0423
alpha = 1/5.1
```

```
N = 212600000/100000
days = 500

# Simulate the deterministic SIR model
SIR_deterministic, t = simulate_sir_model(beta, gamma, N,
    alpha, days)

# Simulate the stochastic SIR model with matching time
    points
population = N
initial_infected = 10
max_time = len(t)  # Use the length of t from the
    deterministic simulation
S_stochastic, I_stochastic, R_stochastic =
    simulate_stochastic(population, initial_infected,
    max_time, beta, gamma)

# Calculate the error rate between the models
error_rate = np.mean(np.abs(I_stochastic - SIR_deterministic
    [:, 2]) / SIR_deterministic[:, 2])
```

```
beta = 0.0756
gamma = 0.0423
alpha = 1/5.1
N = 212600000/100000
days = 500
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
num_sims = 100

plt.figure(figsize=(14, 10))

# Arrays to store simulation data
S_sim = np.zeros([num_sims, len(x)])
```

```
I_sim = np.zeros([num_sims, len(x)])
R_sim = np.zeros([num_sims, len(x)])

# Run simulations
for i in range(num_sims):
    beta = 0.0756
    gamma = 0.0423
    population = 212600000 / 100000
    max_time = 500
    initial_infected = 10
    S, I, R = simulate_stochastic(population,
   initial_infected, max_time, beta, gamma)
    x = np.linspace(0, max_time, int(max_time / 1))

    if i == 0:
        plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
   =0.25)
        plt.plot(x, I, color='red', alpha=0.2, lw=0.25)
        plt.plot(x, R, color='limegreen', alpha=0.2, lw
   =0.25)

        S_sim[i] = S
        I_sim[i] = I
        R_sim[i] = R
    else:
        plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
   =0.25)
        plt.plot(x, I, color='red', alpha=0.2, lw=0.25)
        plt.plot(x, R, color='limegreen', alpha=0.2, lw
   =0.25)

        S_sim[i] = S
        I_sim[i] = I
        R_sim[i] = R

    T_sim = x

# Calculate average of simulations
S_avg = np.mean(S_sim, axis=0)
I_avg = np.mean(I_sim, axis=0)
R_avg = np.mean(R_sim, axis=0)

# Plot average lines with dotted lines
plt.plot(x, S_avg, '--', color='dodgerblue', label='Average
   S')
plt.plot(x, I_avg, '--', color='red', label='Average I')
plt.plot(x, R_avg, '--', color='limegreen', label='Average R
   ')

# Plot deterministic lines from your code
plt.plot(t, SIR[:, 0], '-', color='darkblue', label='
   Deterministic S')
plt.plot(t, SIR[:, 2], '-', color='darkred', label='
   Deterministic I')
plt.plot(t, SIR[:, 3], '-', color='darkgreen', label='
   Deterministic R')

legend = plt.legend()
for l in legend.get_lines():
    l.set_alpha(1)
    l.set_linewidth(1.0)

plt.xlabel(r"$t$", fontsize=14)
plt.ylabel('Population', fontsize=14)
plt.title('Stochastic Simulations with 100 simulations and
   Deterministic SIR Model for Covid-19 in Brazil', fontsize
   =16)

plt.show()
```

```
# Set the parameters and initial conditions for the models
beta = 0.0756
gamma = 0.0423
alpha = 1/5.1
N = 212600000/100000
days = 500

# Simulate the deterministic SIR model
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)

# Simulate the stochastic SIR model with matching time
   points
population = N
initial_infected = 10
max_time = len(t)  # Use the length of t from the
   deterministic simulation
S_stochastic, I_stochastic, R_stochastic =
   simulate_stochastic(population, initial_infected,
   max_time, beta, gamma)

# Calculate the error rate between the models
error_rate_I = np.mean(np.abs(I_stochastic - SIR[:, 2]) /
   SIR[:, 2])
error_rate_S = np.mean(np.abs(S_stochastic - SIR[:, 0]) /
   SIR[:, 0])
```

```
beta = 0.0800
gamma = 0.0681
```

```
alpha = 0.2
N = 126300000/100000
days = 500
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
plt.figure(figsize=(10,7))
plt.plot(t, SIR[:, 0], color='darkblue', label = 'S')
plt.plot(t, SIR[:, 2], color='darkred', label = 'I')
plt.plot(t, SIR[:, 3], color='darkgreen', label = 'R')

plt.legend(fontsize=9)

plt.xlabel(r"$t$", fontsize=13)
plt.ylabel('Population', fontsize=13)
plt.title('SIR model using ODEs for Covid-19 in Japan',
   fontsize=15)
plt.grid(True)

plt.show()
```

```
beta = 0.0800
gamma = 0.0681
population= 126300000/100000
max_time = 500
initial_infected = 10
S, I, R = simulate_stochastic(population, initial_infected,
   max_time, beta, gamma)
```

```
plt.figure(figsize=(10,7))
x = np.linspace(0, max_time, int(max_time/1))
plt.plot(x, S, color="darkblue", label="S")
plt.plot(x, I, color="darkred", label="I")
plt.plot(x, R, color="darkgreen", label="R")
plt.grid()
plt.legend()
plt.title('SIR model using Stochastic Simulation for Covid
   -19 in Japan', fontsize=15)
plt.ylabel("Population",fontsize=13)
plt.xlabel(r"$t$",fontsize=13)
plt.grid(True)

plt.show()
```

```
beta = 0.0800
gamma = 0.0681
alpha = 0.2
N = 126300000/100000
days = 500
SIR, t = simulate_sir_model(beta, gamma, N, alpha, days)
```

```
num_sims = 100

plt.figure(figsize=(14, 10))

# Arrays to store simulation data
S_sim = np.zeros([num_sims, len(x)])
I_sim = np.zeros([num_sims, len(x)])
R_sim = np.zeros([num_sims, len(x)])

# Run simulations
for i in range(num_sims):
    beta = 0.0800
    gamma = 0.0681
    N = 126300000/100000
    max_time = 500
    initial_infected = 10
```

```
        S, I, R = simulate_stochastic(population,
    initial_infected, max_time, beta, gamma)
        x = np.linspace(0, max_time, int(max_time / 1))

        if i == 0:
            plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
    =0.25)
            plt.plot(x, I, color='red', alpha=0.2, lw=0.25)
            plt.plot(x, R, color='limegreen', alpha=0.2, lw
    =0.25)

            S_sim[i] = S
            I_sim[i] = I
            R_sim[i] = R
        else:
            plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
    =0.25)
            plt.plot(x, I, color='red', alpha=0.2, lw=0.25)
            plt.plot(x, R, color='limegreen', alpha=0.2, lw
    =0.25)

            S_sim[i] = S
            I_sim[i] = I
            R_sim[i] = R

        T_sim = x

# Calculate average of simulations
S_avg = np.mean(S_sim, axis=0)
I_avg = np.mean(I_sim, axis=0)
R_avg = np.mean(R_sim, axis=0)
```

```
# Plot average lines with dotted lines
plt.plot(x, S_avg, '--', color='dodgerblue', label='Average
    S')
plt.plot(x, I_avg, '--', color='red', label='Average I')
plt.plot(x, R_avg, '--', color='limegreen', label='Average R
    ')

# Plot deterministic lines from your code
plt.plot(t, SIR[:, 0], '-', color='darkblue', label='
    Deterministic S')
plt.plot(t, SIR[:, 2], '-', color='darkred', label='
    Deterministic I')
plt.plot(t, SIR[:, 3], '-', color='darkgreen', label='
    Deterministic R')

legend = plt.legend()
for l in legend.get_lines():
    l.set_alpha(1)
    l.set_linewidth(1.0)

plt.xlabel(r"$t$", fontsize=14)
plt.ylabel('Population', fontsize=14)
plt.title('Stochastic Simulations with 100 simulations and
    Deterministic SIR Model for Covid-19 in Japan', fontsize
    =16)

plt.show()
```

6.0.1 Further Modelling of Various Diseases Investigating the Effect of Parameters on the Similarity of the Two Models

```
    import numpy as np

class SIR:
    def __init__(self, name, beta, gamma):
        self.Name = name
        self.Beta = beta
        self.Gamma = gamma

    def __str__(self):
        s = f"Name: {self.Name}\nBeta: {self.Beta}\nGamma: {self.Gamma}"
        return s

    def simulate_ODE(self, population, initial_infected, max_time):
        """
         Discrete simulation disease progression:
        - Population
        - initial number of infected
        - time over which to model

        Returns:
        - S, I, R
        """

        # initial set up
        N = population
        S, I, R = [N-initial_infected], [initial_infected], [0]
        t = np.linspace(0, max_time, max_time)

        for _ in t[1:]:
            S.append(S[-1]*(1 - self.Beta*I[-1]/N))
            I.append((self.Beta*S[-1]/N + (1-self.Gamma))*I[-1])
            R.append(R[-1]+self.Gamma*I[-1])

        return S, I, R

    def simulate_stochastic(self, population, initial_infected, max_time):
        """
        Stochastic simulation of disease progression
        - Population
        - initial number of infected
        - time over which to model
        Returns:
        - S, I, R
        """

        # iota = 0.01 - external infection rate
        t = np.linspace(0,max_time,max_time)
        S, I, R, Re = [population - initial_infected], [initial_infected],
    [0], [self.getRe(population, population)]

        for _ in t[1::]:

            l = self.Beta*(I[-1])/population # Rate of new infections

            infection_probability = 1.0 - np.exp(-l) # Probability of
    infection
            recovery_probability = 1.0 - np.exp(-self.Gamma) # Probability
    of recovery

            infection = np.random.binomial(S[-1],infection_probability) #
    Number of new infected
            recovery = np.random.binomial(I[-1],recovery_probability) #
    Number of new recovered

            S.append(S[-1]-infection)
            I.append(I[-1]+infection-recovery)
            R.append(R[-1]+recovery)
            Re.append(self.getRe(S[-1], population))

        return S, I, R

    def getR0(self):
        return self.Beta/self.Gamma

    def getRe(self, num_susceptible, population):
        return self.getR0()*num_susceptible/population
```

```
    import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chisquare
from SIR import SIR
```

```
def test_disease(name, beta, gamma, time):
    sir = SIR(name, beta, gamma)

    # Making simulations
    S_stoch_sum, I_stoch_sum, R_stoch_sum = np.zeros(time), np.zeros(time),
   np.zeros(time)
    S_ODE, I_ODE, R_ODE = sir.simulate_ODE(1000, 10, time)
    S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(1000, 10, time)

    # Plotting
    # All on the same plot, with different shades of the same color for each
     S, I, R
    plt.style.use('ggplot')
    plt.plot(S_ODE, color='darkblue', label='Deterministic S')
    plt.plot(I_ODE, color='darkred', label='Deterministic I')
    plt.plot(R_ODE, color='darkgreen',  label='Deterministic R')
    num_sim = 100
    for _ in range(num_sim):
        S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(1000, 10, time)
        S_stoch_sum += S_stoch
        I_stoch_sum += I_stoch
        R_stoch_sum += R_stoch
        plt.plot(S_stoch, color='lightsteelblue', alpha=0.1)
        plt.plot(I_stoch, color='lightcoral', alpha=0.1)
        plt.plot(R_stoch, color='palegreen', alpha=0.1)
    plt.plot(S_stoch_sum/num_sim, color='dodgerblue', label='Average S',
   linestyle='--')
    plt.plot(I_stoch_sum/num_sim, color='red', label='Average I', linestyle=
   '--')
    plt.plot(R_stoch_sum/num_sim, color='seagreen', label='Average R',
   linestyle='--')
    plt.legend()
    # plt.grid()
    plt.title(f"{name}")
    plt.xlabel(r"$t$")
    plt.ylabel("Population")
    plt.show()
    print(f"MSE S: {np.mean((S_stoch_sum/num_sim - S_ODE)**2)}")
    print(f"MSE I: {np.mean((I_stoch_sum/num_sim - I_ODE)**2)}")
    print(f"MSE R: {np.mean((R_stoch_sum/num_sim - R_ODE)**2)}")

def measure_effect_of_beta_gamma():

    betas = np.linspace(0.1, 1, 10)
    gammas = np.linspace(0.1, 1, 10)
    time = 100

    S_stoch_sum, I_stoch_sum, R_stoch_sum = np.zeros(time), np.zeros(time),
    np.zeros(time)
    MSE_beta_S = []
    MSE_beta_I = []
    MSE_beta_R = []



    # Measuring the effect of beta on the MSE

    for beta in betas:
        for _ in range(50):
            sir = SIR("", beta, 0.2)
            S_ODE, I_ODE, R_ODE = sir.simulate_ODE(1000, 10, time)
            S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(1000, 10,
    time)
            S_stoch_sum += S_stoch
            I_stoch_sum += I_stoch
            R_stoch_sum += R_stoch
        MSE_beta_S.append(np.mean((S_stoch_sum/50 - S_ODE)**2))
        MSE_beta_I.append(np.mean((I_stoch_sum/50 - I_ODE)**2))
        MSE_beta_R.append(np.mean((R_stoch_sum/50 - R_ODE)**2))

    plt.plot(betas, MSE_beta_S, label="S", color='darkblue')
    plt.plot(betas, MSE_beta_I, label="I", color='darkred')
    plt.plot(betas, MSE_beta_R, label="R", color='darkgreen')
    plt.title("MSEs for different beta")
    plt.xlabel(r"$\beta$")
    plt.legend()
    plt.ylabel("MSE")
    plt.show()
    # Measuring the effect of gamma on the MSE

    MSE_gamma_S = []
    MSE_gamma_I = []
    MSE_gamma_R = []


    for gamma in gammas:
        for _ in range(50):
            sir = SIR("", 0.5, gamma)
            S_ODE, I_ODE, R_ODE = sir.simulate_ODE(1000, 10, time)
            S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(1000, 10,
    time)
            S_stoch_sum += S_stoch
            I_stoch_sum += I_stoch
            R_stoch_sum += R_stoch
        MSE_gamma_S.append(np.mean((S_stoch_sum/50 - S_ODE)**2))
        MSE_gamma_I.append(np.mean((I_stoch_sum/50 - I_ODE)**2))
        MSE_gamma_R.append(np.mean((R_stoch_sum/50 - R_ODE)**2))

    plt.plot(gammas, MSE_gamma_S, label="S", color='darkblue')
    plt.plot(gammas, MSE_gamma_I, label="I", color='darkred')
    plt.plot(gammas, MSE_gamma_R, label="R", color='darkgreen')
    plt.legend()
    plt.title("MSEs for different gamma")
    plt.xlabel(r"$\gamma$")
    plt.ylabel("MSE")
    plt.show()



def measure_effect_of_total_population():
    populations = np.linspace(100, 10000, 10)
    time = 100

    S_stoch_sum, I_stoch_sum, R_stoch_sum = np.zeros(time), np.zeros(time),
    np.zeros(time)
    MSE_pop_S = []
    MSE_pop_I = []
    MSE_pop_R = []

    for pop in populations:
        for _ in range(50):
            sir = SIR("", 0.4, 0.2)
            S_ODE, I_ODE, R_ODE = sir.simulate_ODE(pop, pop/100, time)
            S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(pop, pop
/100, time)
            S_stoch_sum += S_stoch
            I_stoch_sum += I_stoch
            R_stoch_sum += R_stoch
        MSE_pop_S.append(np.mean((S_stoch_sum/50 - S_ODE)**2))
        MSE_pop_I.append(np.mean((I_stoch_sum/50 - I_ODE)**2))
        MSE_pop_R.append(np.mean((R_stoch_sum/50 - R_ODE)**2))
    plt.style.use("ggplot")
    plt.plot(populations, MSE_pop_S, label="S", color='darkblue')
    plt.plot(populations, MSE_pop_I, label="I", color='darkred')
    plt.plot(populations, MSE_pop_R, label="R", color='darkgreen')
    plt.legend()
    plt.title("MSEs for different population sizes")
    plt.xlabel("Population size")
    plt.ylabel("MSE")
    plt.show()


def measure_effect_of_starting_infected():
    starting_infected = np.linspace(1, 100, 10)
    time = 100

    S_stoch_sum, I_stoch_sum, R_stoch_sum = np.zeros(time), np.zeros(time),
np.zeros(time)
    MSE_starting_S = []
    MSE_starting_I = []
    MSE_starting_R = []

    for infected in starting_infected:
        for _ in range(50):
            sir = SIR("", 0.4, 0.2)
            S_ODE, I_ODE, R_ODE = sir.simulate_ODE(1000, infected, time)
            S_stoch, I_stoch, R_stoch = sir.simulate_stochastic(1000,
infected, time)
            S_stoch_sum += S_stoch
            I_stoch_sum += I_stoch
            R_stoch_sum += R_stoch
        MSE_starting_S.append(np.mean((S_stoch_sum/50 - S_ODE)**2))
        MSE_starting_I.append(np.mean((I_stoch_sum/50 - I_ODE)**2))
        MSE_starting_R.append(np.mean((R_stoch_sum/50 - R_ODE)**2))
    plt.style.use("ggplot")
    plt.plot(starting_infected/1000, MSE_starting_S, label="S", color='
    darkblue')
    plt.plot(starting_infected/1000, MSE_starting_I, label="I", color='
    darkred')
    plt.plot(starting_infected/1000, MSE_starting_R, label="R", color='
    darkgreen')
    plt.legend()
    plt.title("MSEs for different starting infected")
    plt.xlabel("Starting infected (% of population)")
    plt.ylabel("MSE")
    plt.show()










if __name__=="__main__":
    # test_disease("Ebola", 0.71, 0.4152, 100)
    """
    "Estimating the Reproduction Number of Ebola Virus (EBOV) during the
    2014 Outbreak in West Africa"
    published in PLOS Currents Outbreaks in 2014
    (doi: 10.1371/currents.outbreaks.91afb5e0f279e7f29e7056095255b288)
    """
    test_disease("COVID-19", 0.328, 0.1, 100)
    """"The Novel Coronavirus, 2019-nCoV, Is Highly Contagious and More
    Infectious Than Initially Estimated"
    published in the journal MedRxiv in 2020
    (doi: 10.1101/2020.02.07.20021154)."""
    # test_disease("Influenza", 0.4, 0.3, 100)
    """
    Different Epidemic Curves for Severe Acute Respiratory Syndrome Reveal
   Similar Impacts of Control Measures"
    published in the Proceedings of the National Academy of Sciences in 2004
    (doi: 10.1073/pnas.0402950101).
    """
    # test_disease("Measles", 0.857, 0.0714, 100)
    """"Infectious Diseases of Humans: Dynamics and Control" published in
   1992."""
    # test_disease("Test 1", 0.5, 0.02, 200)
    # test_disease("Test", 0.0800, 0.0681, 1000)

    # measure_effect_of_total_population()
    # measure_effect_of_starting_infected()
```

Part 2

SEIR model for Covid-19 in Brazil using ODE

```
def diff(SIR, t, beta, gamma, N, alpha):
    dsdt = - (beta * SIR[0] * SIR[2])/N
    dedt = (beta * SIR[0] * SIR[2])/N - alpha*SIR[1]
    didt = alpha*SIR[1] - gamma * SIR[2]
    drdt = gamma * SIR[2]
    dSIRdt = [dsdt,dedt,didt, drdt]
    return dSIRdt



def simulate_seir_model(beta, gamma, N, alpha, days):
    S = N - 10
    I = 10
    R = 0
    E = 10
    day = days
    # initial conditions
    SIR_init = (S, E, I, R)
    # time points
    t = np.linspace(0, day, 1000)
    # solve ODE
    SEIR = odeint(diff, SIR_init, t, args=(beta, gamma, N,
   alpha))
    return SEIR, t

beta = 0.0756
gamma = 0.0423
sigma = 0.9
N = 212600000/100000
days = 500
alpha = 0.9

SEIR, t = simulate_seir_model(beta, gamma, N, alpha, days)

plt.figure(figsize=(10,7))
plt.plot(t, SEIR[:, 0], color='darkblue', label = 'S')
plt.plot(t, SEIR[:, 1], color='orange', label = 'E')
plt.plot(t, SEIR[:, 2], color='darkred', label = 'I')
plt.plot(t, SEIR[:, 3], color='darkgreen', label = 'R')



plt.legend(fontsize=9)

plt.xlabel(r"$t$", fontsize=13)
plt.ylabel('Population', fontsize=13)
plt.title('SEIR model using ODEs for Covid-19 in Brazil',
   fontsize=15)
plt.grid(True)
plt.show()
```

Stochastic SEIR model

```
 def simulate_stochastic_SEIR(population,
initial_infected, max_time, beta, gamma, sigma):
 """
 Stochastic simulation of SEIR disease progression
 - Population
 - initial number of infected
 - time over which to model
 - beta: transmission rate
 - gamma: recovery rate
 - sigma: rate of transition from exposed to infectious

 Returns:
 - S, E, I, R
 """


 t = np.linspace(0, max_time, max_time)
 S, E, I, R = [population - initial_infected], [1], [
initial_infected], [0]

 for _ in t[1::]:

     # Compute transition probabilities
     l = beta * (I[-1] + E[-1]) / population  # Effective
 transmission rate
     ifrac = 1.0 - np.exp(-l)  # Probability of infection
     rfrac = 1.0 - np.exp(-gamma)  # Probability of
recovery
     efrac = 1.0 - np.exp(-sigma)  # Probability of
transition from exposed to infectious
```

```
        # Determine new cases
        exposure = np.random.binomial(S[-1], ifrac)  #
   Number of newly exposed
        infection = np.random.binomial(E[-1], efrac)  #
   Number of new infected
        recovery = np.random.binomial(I[-1], rfrac)  #
   Number of new recovered

        # Update compartments
        S.append(S[-1] - exposure)
        E.append(E[-1] + exposure - infection)
        I.append(I[-1] + infection - recovery)
        R.append(R[-1] + recovery)

    return S, E, I, R


beta = 0.0756
gamma = 0.0423
sigma = 1/5
N = 212600000/100000
days = 500
alpha = 0.9

SEIR, t = simulate_seir_model(beta, gamma, N, alpha, days)

plt.figure(figsize=(10,7))
plt.plot(t, SEIR[:, 0], color='darkblue', label = 'S')
plt.plot(t, SEIR[:, 1], color='orange', label = 'E')
plt.plot(t, SEIR[:, 2], color='darkred', label = 'I')
plt.plot(t, SEIR[:, 3], color='darkgreen', label = 'R')
plt.legend(fontsize=9)

plt.xlabel(r"$t$", fontsize=13)
plt.ylabel('Population', fontsize=13)
plt.title('SEIR model using ODEs for Covid-19 in Brazil',
   fontsize=15)
plt.grid(True)

plt.show()
```

Creating 100 stochastic SEIR simulation for Brazil

```
beta = 0.0756
gamma = 0.0423
sigma = 1/5
N = 212600000/100000
days = 500
alpha = 1/5

SEIR, t = simulate_seir_model(beta, gamma, N, alpha, days)

def simulate_stochastic_SEIR(population, initial_infected,
   max_time, beta, gamma, sigma):
    """
    Stochastic simulation of SEIR disease progression
    - Population
    - initial number of infected
    - time over which to model
    - beta: transmission rate
    - gamma: recovery rate
    - sigma: rate of transition from exposed to infectious
    Returns:
    - S, E, I, R
    """

    t = np.linspace(0, max_time, max_time)
    S, E, I, R = [population - initial_infected], [1], [
initial_infected], [0]

    for _ in t[1::]:

        # Compute transition probabilities
        l = beta * (I[-1] + E[-1]) / population  # Effective
 transmission rate
        ifrac = 1.0 - np.exp(-l)  # Probability of infection
        rfrac = 1.0 - np.exp(-gamma)  # Probability of
recovery
        efrac = 1.0 - np.exp(-sigma)  # Probability of
transition from exposed to infectious

        # Determine new cases
        exposure = np.random.binomial(S[-1], ifrac)  #
Number of newly exposed
        infection = np.random.binomial(E[-1], efrac)  #
Number of new infected
        recovery = np.random.binomial(I[-1], rfrac)  #
Number of new recovered

        # Update compartments
        S.append(S[-1] - exposure)
        E.append(E[-1] + exposure - infection)
        I.append(I[-1] + infection - recovery)
        R.append(R[-1] + recovery)
    return S, E, I, R

beta = 0.0756
gamma = 0.0423
sigma = 1/5
N = 212600000/100000
days = 500
alpha = 1/5

SEIR, t = simulate_seir_model(beta, gamma, N, alpha, days)



num_sims = 100

plt.figure(figsize=(14, 10))

S, E, I, R = simulate_stochastic_SEIR(population,
   initial_infected, max_time, beta, gamma, sigma)
S_stoch_sum, E_stoch_sum, I_stoch_sum, R_stoch_sum = np.
   zeros([num_sims, len(S)]), np.zeros([num_sims, len(E)]),
   np.zeros([num_sims, len(I)]), np.zeros([num_sims, len(R)
   ])

for i in range(num_sims):
    beta = 0.0756
    gamma = 0.0423
    sigma = 1/5
    population = 212600000/100000
    max_time = 500
    initial_infected = 10
    S, E, I, R = simulate_stochastic_SEIR(population,
   initial_infected, max_time, beta, gamma, sigma)

    x = np.linspace(0, max_time, int(max_time/1))
    S_sim = np.zeros([num_sims, len(S)])
    E_sim = np.zeros([num_sims, len(E)])
    I_sim = np.zeros([num_sims, len(I)])
    R_sim = np.zeros([num_sims, len(R)])

    S_stoch_sum[i] = S
    E_stoch_sum[i] = E
    I_stoch_sum[i] = I
    R_stoch_sum[i] = R

    if i == 0:
        plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
   =0.25)
        plt.plot(x, E, color='orange', alpha=0.2, lw=0.25)
        plt.plot(x, I, color='red', alpha=0.2, lw=0.25)
        plt.plot(x, R, color='limegreen', alpha=0.2, lw
   =0.25)

    else:
        plt.plot(x, S, color='dodgerblue', alpha=0.2, lw
   =0.25)
        plt.plot(x, E, color='orange', alpha=0.2, lw=0.25)
        plt.plot(x, I, color='darkred', alpha=0.2, lw=0.25)
        plt.plot(x, R, color='limegreen', alpha=0.2, lw
   =0.25)

T_sim = x
```

```
S_stoch_avg = np.mean(S_stoch_sum, axis=0)
E_stoch_avg = np.mean(E_stoch_sum, axis=0)
I_stoch_avg = np.mean(I_stoch_sum, axis=0)
R_stoch_avg = np.mean(R_stoch_sum, axis=0)

# Plot average lines with dotted lines
plt.plot(T_sim, S_stoch_avg, '--', color='dodgerblue', label
   ='Average S')
plt.plot(T_sim, E_stoch_avg, '--', color='orange', label='
   Average E')
plt.plot(T_sim, I_stoch_avg, '--', color='red', label='
   Average I')
plt.plot(T_sim, R_stoch_avg, '--', color='limegreen', label=
   'Average R')

plt.plot(t, SEIR[:, 0], '-', color='darkblue', label='
   Deterministic S')
plt.plot(t, SEIR[:, 1], '-', color='darkorange', label='
   Deterministic E')
plt.plot(t, SEIR[:, 2], '-', color='darkred', label='
   Deterministic I')
plt.plot(t, SEIR[:, 3], '-', color='darkgreen', label='
   Deterministic R')

legend = plt.legend()
for l in legend.get_lines():
    l.set_alpha(1)
    l.set_linewidth(1.0)

plt.xlabel(r"$t$")
plt.ylabel('Population')
plt.title('Stochastic Simulations with 100 simulations and
   Deterministic SEIR Model for Covid-19 in Brazil')
plt.show()
```

6.0.2 SIRV Model

```
import numpy as np
from scipy.integrate import import odeint
import matplotlib.pyplot as plt
import math



class vaccination_analysis ():

    # DETERMINISTIC MODEL
    ----------------------------------------------------------------------------


    def __init__(self, beta, gamma, v_rate, effectiveness,
   max_time):
            self.beta = beta
            self.gamma = gamma
            self.v_rate = v_rate
            self.effectiveness = effectiveness

            self.time_passed = 0

            self.max_time = max_time

            # Set the initial conditions and parameters
            self.y0 = [999, 101, 0, 0]  # initial population
   : 999 susceptible, 1 infectious, 0 recovered, 0
```

```
vaccinated

    # Define the SIRV model equations
    def sirv_model(self, y, t, beta, gamma, v_rate):
        S, I, R, V = y
        N = S + I + R + V
        dSdt = -beta * S * I / N - v_rate * S * self.
effectiveness
        dIdt = beta * S * I / N - gamma * I
        dRdt = gamma * I
        dVdt = v_rate * S * self.effectiveness
        return [dSdt, dIdt, dRdt, dVdt]



    def build_deterministic_model(self):
        # Set the time points to simulate
        t = np.linspace(0, self.max_time, 100)

        # Solve the SIRV model equations
        sol = odeint( self.sirv_model, self.y0, t, args=(
self.beta, self.gamma, self.v_rate))

        # Plot the results
        plt.style.use("ggplot")
        plt.plot(t, sol[:, 0], label='Susceptible', color="
blue", linestyle='--')
        plt.plot(t, sol[:, 1], label='Infected', color="
orange", linestyle='--')
        plt.plot(t, sol[:, 2], label='Recovered', color="
green", linestyle='--')
        plt.plot(t, sol[:, 3], label='Vaccinated', color="
red", linestyle='--')
```

```
        plt.xlabel('Time')
        plt.ylabel('Population')



    #
    ------------------------------------------------------------------------------------



    # STOCHASTIC MODEL
    ------------------------------------------------------------------------------------


    def sirv_stochastic(self,u,t):
        S,I,R, V=u

        N = np.sum(self.y0)
        dt = 0.1
        self.time_passed = self.time_passed + dt




        # transmission rate lambda
        lambd = self.beta * I / N

        ifrac = 1.0 - math.exp(-lambd*dt)
        rfrac = 1.0 - math.exp(-self.gamma*dt)




        vfrac = 1.0 - math.exp(-self.v_rate*dt)
        infection = np.random.binomial(S,ifrac)
        recovery = np.random.binomial(I,rfrac)
        #from susceptible, draw a populatrion to be
vaccinated
        vaccinated = np.random.binomial(S,vfrac)
        immune = math.floor(vaccinated * self.effectiveness)

        return [S-infection-immune, I+infection-recovery,R+
recovery, V + immune]

    def simulate(self):



        tf = self.max_time
        tl = 2001
        t = np.linspace(0,tf,tl)
        S = np.zeros(tl)
        I = np.zeros(tl)
        R = np.zeros(tl)
        V = np.zeros(tl)
        u = self.y0
        S[0],I[0],R[0], V[0] = u

        # loop for generating samples
        for j in range(1,tl):
            u = self.sirv_stochastic(u,t[j])
            S[j],I[j],R[j], V[j] = u
        return {'t':t,'S':S,'I':I, 'V':V, 'R':R}

    def build_stochastic_model(self):
        plt.style.use("ggplot")
        sir_out = self.simulate()
        plt.title(f"Stochastic Model of the Flu, with \u03B2
 = {np.round(self.beta, 2)}, \u03B3 = {np.round(self.
gamma, 2)} and effectiveness of {int(self.effectiveness *
 100)}% ", fontsize=10, loc="left")
        sline = plt.plot("t","S","",data=sir_out,label='
Susceptible', color="blue")
        iline = plt.plot("t","I","",data=sir_out, label='
Infected', color="orange")
        rline = plt.plot("t","R","",data=sir_out, label='
Recovered', color="green")
        vline = plt.plot("t","V","",data=sir_out, label='
Vaccinated', color="red")
        plt.xlabel("Time",fontweight="bold")
        plt.ylabel("Population",fontweight="bold")
        legend = plt.legend(loc=1)
        frame = legend.get_frame()
        legend.set_title("Groups")
        frame.set_facecolor("white")
        frame.set_linewidth(0)

    def build_susceptible(self):
        plt.style.use("ggplot")
        sir_out = self.simulate()

        plt.title(f"Stochastic Model of the Flu, with \u03B2
 = {np.round(self.beta, 2)}, \u03B3 = {np.round(self.
gamma, 2)} and effectiveness of {int(self.effectiveness *
 100)}% ", fontsize=10, loc="left")
        sline = plt.plot("t","S","",data=sir_out,label=f"
Susceptible, {int(self.effectiveness * 100)}%")
        plt.xlabel("Time",fontweight="bold")
        plt.ylabel("Population",fontweight="bold")
        legend = plt.legend(loc=1)
        frame = legend.get_frame()
        legend.set_title("Groups")
        frame.set_facecolor("white")
        frame.set_linewidth(0)

    def mean_susceptibles(self):
        means = []
        #collect 50 simulations
        for i in range(50):
            sir_out = self.simulate()
            means.append(sir_out["S"][-1])

        print(f"mean {int(self.effectiveness * 100)}%
    effectiveness: {np.mean(means)}")
        print(f"std {int(self.effectiveness * 100)}%
    effectiveness: {np.std(means)}")

    #
    ----------------------------------------------------------------------




#s1 = vaccination_analysis (1/3, 1/7, 1, 0.01)
#s1 = vaccination_analysis (0.4, 0.3, 1, 0.02, 0.9, 100)

#Flu
s1 = vaccination_analysis (1/3, 1/7, 0.02, 0.9, 100)
s2 = vaccination_analysis (1/3, 1/7, 0.02, 0.4, 100)
```

```
#Ebola
# s1 = vaccination_analysis (0.4, 0.3, 0.02, 0.9, 100)
# s2 = vaccination_analysis (0.4, 0.3, 0.02, 0.4, 100)

s1.build_stochastic_model()
s1.build_deterministic_model()



# s1.mean_susceptibles()
# s2.mean_susceptibles()
plt.show()
```

References

[1] (2004). Different epidemic curves for severe acute respiratory syndrome reveal similar impacts of control measures. *National Academy of Sciences.*

[2] (2014). Estimating the reproduction number of ebola virus (ebov) during the 2014 outbreak in west africa. *PLOS Currents Outbreaks.*

[3] (2019). On the relation between stochastic epidemic models and self-exciting point processes. *Graz University of Technology.*

[4] (2020). The novel coronavirus, 2019-ncov, is highly contagious and more infectious than initially estimated. *medRxiv.*

[5] (2021). Modeling covid-19 data usingan sir model. *Lagrange Collage.*

[6] (2023). Coronavirus incubation period and covid-19 symptoms. Accessed on June 22, 2023.

[7] Anderson, R. and May, R. (1992). *Infectious diseases of humans: Dynamics and control.* AASLD.

7 Sources

Source for finding β and γ values from the basic reproduction number, R0: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2715422/`

Source for SIR models in Python: `http://epirecip.es/epicookbook/chapters/seir/r_desolve`